TAL TECH

DOCTORAL THESIS

Automating Defences against Cyber Operations in Computer Networks

Mauno Pihelgas

TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY
TALLINN 2021

TALLINN UNIVERSITY OF TECHNOLOGY
DOCTORAL THESIS
36/2021

Automating Defences against Cyber Operations in Computer Networks

MAUNO PIHELGAS

TAL TECH PRESS

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Software Science

The dissertation was accepted for the defence of the degree of Doctor of Philosophy in Computer Science on 10 June 2021

Supervisor: Dr. Risto Vaarandi,
Department of Software Science, School of Information Technologies,
Tallinn University of Technology,
Tallinn, Estonia

Co-supervisor: Professor Dr. Olaf Manuel Maennel,
Department of Software Science, School of Information Technologies,
Tallinn University of Technology,
Tallinn, Estonia

Opponents: Professor Dr. Anja Feldmann,
Max Planck Institute for Informatics,
Saarbrücken, Germany

Professor Dr. Jan Vykopal,
Masaryk University,
Brno, Czechia

Defence of the thesis: 11 August 2021, Tallinn

Declaration:
Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology, has not been submitted for any academic degree elsewhere.

Mauno Pihelgas _____
signature

ISSN 2585-6898 (publication)
ISBN 978-9949-83-718-2 (publication)
ISSN 2585-6901 (PDF)
ISBN 978-9949-83-719-9 (PDF)
Printed by Koopia Niini & Rauam

TALLINNA TEHNIKAÜLIKOOL
DOKTORITÖÖ
36/2021

Arvutivõrkude kaitse automatiseerimine küberoperatsioonide vastu

MAUNO PIHELGAS

TAL TECH
KIRJASTUS

Contents

List of Publications

I R. Vaarandi and M. Pihelgas. Using Security Logs for Collecting and Reporting Technical Security Metrics. In *Military Communications Conference (MILCOM), 2014 IEEE*, pages 294–299, October 2014

II R. Vaarandi and M. Pihelgas. LogCluster - A data clustering and pattern mining algorithm for event logs. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 1–7, November 2015

III R. Vaarandi, M. Kont, and M. Pihelgas. Event log analysis with the LogCluster tool. In *MILCOM 2016 - 2016 IEEE Military Communications Conference*, pages 982–987, November 2016

IV B. Blumbergs, M. Pihelgas, M. Kont, O. Maennel, and R. Vaarandi. Creating and Detecting IPv6 Transition Mechanism-Based Information Exfiltration Covert Channels. In *Secure IT Systems: 21st Nordic Conference, NordSec 2016, Oulu, Finland, November 2-4, 2016. Proceedings*, pages 85–100. Springer International Publishing, 2016

V M. Kont, M. Pihelgas, K. Maennel, B. Blumbergs, and T. Lepik. Frankenstack: Toward real-time Red Team feedback. In *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, pages 400–405, October 2017

VI M. Pihelgas. Design and Implementation of an Availability Scoring System for Cyber Defence Exercises. In *14th International Conference on Cyber Warfare and Security (ICCWS 2019)*, page 329–337, 2019

VII P. Théron, A. Kott, M. Drašar, K. Rzadca, B. LeBlanc, M. Pihelgas, L. Mancini, and A. Panico. Towards an active, autonomous and intelligent cyber defense of military systems: The NATO AICA reference architecture. In *2018 International Conference on Military Communications and Information Systems (ICMCIS)*, pages 1–9, May 2018

VIII P. Théron, A. Kott, M. Drašar, K. Rzadca, B. LeBlanc, M. Pihelgas, L. Mancini, and F. de Gaspari. Reference Architecture of an Autonomous Agent for Cyber Defense of Complex Military Systems. In *Adaptive Autonomous Secure Cyber Systems*, pages 1–21, Cham, 2020. Springer International Publishing

IX A. Kott, P. Théron, L. V. Mancini, E. Dushku, A. Panico, M. Drašar, B. LeBlanc, P. Losiewicz, A. Guarino, M. Pihelgas, and K. Rzadca. An introductory preview of Autonomous Intelligent Cyber-defense Agent reference architecture, release 2.0. *The Journal of Defense Modeling and Simulation*, 17(1):51–54, 2020

X R. Vaarandi and M. Pihelgas. NetFlow Based Framework for Identifying Anomalous End User Nodes. In *15th International Conference on Cyber Warfare and Security (ICCWS 2020)*, page 448–456, 2020

XI M. Pihelgas and M. Kont. Frankenstack: Real-time Cyberattack Detection and Feedback System for Technical Cyber Exercises. In *2021 IEEE CSR Workshop on Cyber Ranges and Security Training (CRST)*. IEEE, July 2021. (Accepted paper)

Author's Contributions to the Publications

I I conducted the background study on security metrics research, properties of good security metrics, and how to create an organisational security metrics program. Furthermore, I offered proposals and suggestions in the metrics selection process as well as helped draw conclusions on the example security metrics extracted from the organisational framework implementation.

II I conducted performance evaluation experiments of the LogCluster algorithm on Locked Shields data including both the final and intermediate versions of the algorithm. I also participated in the design and development discussions of the LogCluster algorithm. During the development phase of the algorithm, I conducted a detailed evaluation of various data structures used by the cluster candidate support aggregation procedure, measuring how different data structures influence the algorithm performance under heavy workloads.

III I participated in the discussions and helped create several novel LogCluster usage examples presented in the paper.

IV I was responsible for describing the data exfiltration detection experiment and assembling the set of capture combinations of each exfiltration tool, source and destination port number, transport layer protocol, and IP version. Altogether, 126 unique packet capture (PCAP) files were generated for analysis. We used dedicated monitoring nodes to run a selection of five popular open-source detection tools which would analyse each of these PCAP files, produce flow records, and potentially generate alerts for suspicious activity. We presented the detection results in an extensive table and I provided the detailed discussion of the detection results.

V I was the co-author of the cyberattack detection and feedback system designed to increase the training benefit for the cyber red team participants during cyber exercises. The authors organised a one-week hackathon during the Crossed Swords 2017 Test Run for sprint-developing the Frankenstack framework—the near real-time technical feedback system for the cyber red team. I was responsible for designing the post-processing, filtering and correlation engine to process raw events from intrusion detection system, syslog, Windows logs. These transformed and correlated (i.e., meaningful) events were then displayed on feedback dashboard for the cyber red team to see and learn. The framework was successfully implemented and tested in the NATO CCD COE's technical cyber security exercise Crossed Swords. Furthermore, I provided input for the cyber red team situation awareness (SA) feedback questionnaire prior to the event and later analysed the feedback to extract valuable takeaways for improving Frankenstack in the upcoming years.

VI I was the sole author of this paper and have been the primary designer, developer and implementer of the Availability Scoring system for the Locked Shields cyber defence exercise since 2014. The paper describes the design process and reasoning behind various decisions made in the scoring system that has been used annually since 2014. The system provides automated scoring checks for a variety of systems throughout the exercise. While the framework employs many standard system monitoring practices, one of the primary differentiators is the adversarial environment in which the service availability checks have to be performed. It is in the winning interest of participating teams to mislead the monitoring system in order to obtain a better score for the availability of services.

VII This is the first of three publications on the topic of **A**utonomous **I**ntelligent **C**yber-defence **A**gents (AICA). Publication VII is based on the intermediate results of the work done by the NATO Science and Technology Organisation's IST-152 Research Task Group (RTG) between 2016 and 2018. I was an active member of this research group. Publication VII captures the primary concepts of the initial reference architecture [85] published by the IST-152 group. This preliminary research resulted in the publication of the *Initial Reference Architecture of an Intelligent Autonomous Agent for Cyber Defense* [85], in which I directly contributed to the research and write-up of the *Sensing and World State Identification* chapter. Correspondingly, I was the co-author of the same chapter within Publication VII.

VIII This book chapter published in [68] is the second of three publications on the topic of AICAs. This chapter is largely based on the second revision of the *Autonomous Intelligent Cyber-defense Agent (AICA) Reference Architecture* report [86] that comprises the work done by the NATO IST-152 group between 2016 and 2019. In Publication VIII, I contributed to the research and write-up of section *Sensing and World State Identification* and was the primary author of section *Use Cases* that provides an example scenario of AICAs being deployed within Unmanned Aerial Vehicles (UAV).

IX This journal article is the third of three publications on the topic of AICAs. Publication IX serves as an introductory and overview article of the second revision of the *Autonomous Intelligent Cyber-defense Agent (AICA) Reference Architecture* report [86] that concludes the work done by the NATO IST-152 RTG between 2016 and 2019. Within this second release of the reference architecture, I directly contributed to the research and write-up of the *Rationale of AICA and Scenario* and *Sensing and World State Identification* chapters.

X I provided input for the development and features of the novel detection framework. In later phases, I supported the work by looking through the analysed NetFlow dataset to provide additional observations on the performance of the proposed methods.

XI I was the primary author of this paper. The paper describes the research and development effort of Frankenstack following the initial paper (Publication V) in 2017. This latest publication describes the updated architecture, event normalisation, data enrichment methods, and improved event processing within the newly developed version. In addition to contributing most of the manuscript write-up, I was responsible for the development of the *Exercise asset collection* and *Python event shipper* modules described in the paper.

List of Abbreviations

ACK	Acknowledgement
AHEAD	Attackers Hindered by Employing Active Defense
AICA	Autonomous Intelligent Cyber-defence Agents
AIS	Automatic Identification System
AMD	Advanced Micro Devices
API	Application Programming Interface
APT	Advanced Persistent Threat
ARIMA	Autoregressive Integrated Moving Average
ARP	Address Resolution Protocol
AWS	Amazon Web Services
BT	Blue Team
CCD COE	Cooperative Cyber Defence Centre of Excellence
CDX	Cyber Defence Exercise
CERT	Computer Emergency Response Team
CGC	Cyber Grand Challenge
CIS	Center for Internet Security
CPU	Central Processing Unit
CRS	Cyber Reasoning Systems
CSX	Cyber Security Exercise
DARPA	Defense Advanced Research Projects Agency
DDoS	Distributed Denial of Service
DDR	Double Data Rate
DGA	Domain Generation Algorithm
DNS	Domain Name System
ERSPAN	Encapsulated Remote Switched Port Analyser
EWMA	Exponentially Weighted Moving Average
FIN	Finish
GB	Gigabyte
GiB	Gibibyte
GNU	GNU's Not Unix!
GPL	GNU General Public License
GT	Green Team
HDFS	Hadoop Distributed File System
HPC	High-Performance Computing
HQ	Headquarters
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
ICS	Industrial Control System
ICT	Information and communications technology
IDS	Intrusion Detection System
IMAP	Internet Message Access Protocol
IoC	Indicator of Compromise
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IPFIX	IP Flow Information Export

IPS	Intrusion Protection System
IPsec	Internet Protocol Security
ISP	Internet Service Provider
IST	Information Systems Technology
IT	Information Technology
JSON	JavaScript Object Notation
LCS	Longes Common Subsequence
LS	Locked Shields
LTE	Long-Term Evolution
MB	Megabyte
MiB	Mebibyte
NATO	North Atlantic Treaty Organization
NCIA	NATO Communications and Information Agency
NIST	National Institute of Standards and Technology
NOC	Network Operations Center
NSM	Network Security Monitoring
NTP	Network Time Protocol
PCAP	Packet Capture
PID	Process Identifier
PLC	Programmable Logic Controller
RT	Red Team
RTG	Research Task Group
SA	Situation Awareness
SCADA	Supervisory Control and Data Acquisition
SIEM	Security Information and Event Management
SITREP	Situation Report
SLA	Service-Level Agreement
SMTP	Simple Mail Transfer Protocol
SOC	Security Operation Center
SSD	Solid-State Drive
SSH	Secure Shell
SSL	Secure Sockets Layer
SYN	Synchronise
TCP	Transmission Control Protocol
TICK	Telegraf, InfluxDB, Chronograf, Kapacitor
TTL	Time to Live
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
USB	Universal Serial Bus
VLAN	Virtual Local Area Network
VM	Virtual Machine
VPN	Virtual Private Network
WSI	World State Identification
WT	White Team
XS	Crossed Swords
YT	Yellow Team

1 Introduction

Over the last two decades computer networks have grown significantly. Modern systems are becoming more complex and more difficult to test, operate, monitor, and defend. Along with the accelerated proliferation of interconnected systems, the attack surface for malicious actors has also increased.

In order to discover problems and detect breaches in computer networks and connected systems, it has become common to collect and analyse network traffic and system logs from network devices, servers, and even workstations. However, for large organisations the amount of data collected on a daily basis can exceed hundreds of gigabytes and over the course of weeks terabytes of data accumulate. Just maintaining storage and computational requirements for timely analysis of such amount of data is difficult, however, making sense of this data is even more challenging for the monitoring and security analysts. For the reasons above, it is essential for organisations to operate well-designed monitoring and situation awareness (SA) systems which rely on advanced network monitoring and system log analysis algorithms.

Considering the increasing sophistication and level of automation employed by malicious actors, human operators' reaction times are far too slow to take any defensive action before the adversary has already been able to disrupt the target systems. Defending complex and highly interconnected systems will soon become overwhelming for human operators. To counter this critical tendency, cyber defenders must improve their monitoring techniques and strive to implement better defensive solutions. Automation and advances towards autonomous defences are pivotal in this endeavour.

In recent years, the number of security appliances on the market has increased considerably. While it might be tempting to just connect one of these systems to the network and expect them to just work and provide meaningful output on their own, this is generally not the case. No system can effectively defend a complex organisational network out of the box. Instead, security systems must be customised and continuously maintained in order to meet the organisation's needs. Furthermore, such systems still rely heavily on the expertise of their developers, administrators, and analysts, which is why it is crucial for organisations to ensure adequate training for cyber defender to increase their readiness in this relatively new domain.

Cyber exercises are a great way to enable cyber security experts to enhance their competence and skills in defending IT systems and networks under the stress of real-time attacks. Fictional, yet realistic scenarios turn a cyber security exercise (CSX) into an intriguing experience for all participants. While proper tools and infrastructure are essential for learning, they do not make an exercise successful by default. Instead, the progression and learning experience of the participants has to be continuously measured and adapted in order to maximise their learning benefit from the exercise. Providing participants continuous feedback throughout the exercise is crucial for improving skills in a fast-paced cyber exercise.

In addition to training cyber security experts, cyber exercises offer an excellent opportunity for verifying the design of security monitoring and SA tools and algorithms in a controlled exercise environment. The content of this thesis is intricately connected to cyber exercises, since many of the concepts, methods and algorithms discussed in this thesis have been validated during cyber exercises.

With all kinds of scientific and technological improvements organisations are still struggling with ensuring adequate level of security for their IT networks and systems. Even if standards and best practices are applied when installing and maintaining the systems, the current security posture is not really known unless appropriate full-stack monitoring ca-

pability has been set up. Although the need for security monitoring is generally agreed upon, it is often unclear how to build a useful monitoring solution. It is not possible to go from *zero to hundred* (e.g., from no capability to autonomous defences) with just one step. Rather, it is a rigorous process which has to consider the basic steps before proceeding to more advanced challenges. This thesis aims to describe this process starting from the fundamental capabilities (e.g., specifying security metrics) and working up to developing a tailored security solution that comprises advanced algorithms for log mining and network security monitoring.

Many organisations can operate some basic monitoring capability (e.g., log collection or availability monitoring). First obstacles typically arise when the amount of collected data grows large. People are generally not particularly good at comprehending large amounts of machine-generated data (e.g., event logs or technical metrics) and this data by itself is typically quite useless. It must be processed to extract relevant information and transform it into actionable knowledge. Furthermore, it is often necessary to re-examine some of those basic functions (e.g., event log processing or metrics gathering) to assure that it actually supports the requirements of higher level security incident analysis. Subsequently, with improved data exploration capability operators often develop a particular insight and a raised SA level regarding the environment they have to defend.

While humans are struggling to understand the data, computers should be making this process simpler. However, this is not always the case. One concern is that many traditional log management and network analysis tools are not able to handle big data which creates a serious obstacle for security monitoring and analysis. Moreover, high performance solutions or commercial appliances are often not affordable for many smaller institutions. Furthermore, many standard tools do not provide the required functionality or enough information, so the operator is forced to use multiple tools simultaneously to get a full picture of the situation. Tackling this problem requires a decent methodology, capable tools, and more efficient algorithms. Therefore, this thesis focuses on the use of open-source solutions for building efficient security monitoring frameworks.

This thesis addresses problems in the areas of determining which metrics are relevant for security monitoring, how to build general-purpose and cyber-exercise-specific SA systems, how to raise SA qualification and readiness of security system operators during cyber exercises, how to implement and verify novel monitoring algorithms and frameworks for cyber defence, and improving defence by designing advanced intelligent agents for relieving the burden from human operators. In a long-term future outlook, the goal of this research is to advance monitoring and cyber defence techniques forward to a degree where it would be possible to consider a fully functional autonomous defence system in cyber security. Eventually, leaving the human operator out of the immediate decision-making loop.

A secondary objective of the thesis is to be a valuable source of information not just for researchers but also for practitioners looking to implement similar systems. Therefore, this thesis aims to unite theoretical research with an abundance of practical guidance and implementation examples. A similar approach has also been followed in most of the author's publications presented as a part of this thesis.

1.1 Research questions

This thesis addresses three primary topics. For each topic, a short description and several research questions aiming at specific nuances of the topics are provided. Table 1 outlines the relevant connections between research questions, thesis chapters, publications, and contributions (described in subsection 1.2 below).

Security metrics and SA solutions in organisational and training environments It is often unclear how to set up proper organisational monitoring and situation awareness solution—deciding which metrics to gather, how to process event logs, when to send alerts, what to show on dashboards, etc.

Useful security metrics and SA systems can only be built on a foundation that provides efficient monitoring capability. Although many security monitoring tools facilitate data collection and data representation functions, they often lack the necessary systematic approach for accommodating security metrics reporting and raising operator SA levels. Without proper tools and monitoring capability, operators cannot effectively defend large networks. Furthermore, the tools and solutions have to support the operator's ability to transform data into actionable knowledge.

However, just providing the tools is often not enough—operators require a good learning environment to practice using those tools, develop new skills, and advance existing ones. CSXs provide an excellent training environment for cyber defenders in a realistic setting without the fear of disrupting critical production systems.

RQ1.1. What information is relevant and in which form should it be displayed in the SA system to support operator's ability to perceive and comprehend events and processes?

RQ1.2. How to verify that the SA system and its underlying algorithms are well-designed and provide adequate performance?

RQ1.3. How to design a CSX-specific SA and scoring systems?

RQ1.4. How to measure and improve the quality of operator training and learning experience?

Enhancing security monitoring detection capability Computer networks are growing in size, complexity, bandwidth, and variety of connected devices. Consecutively, the growing number of IT systems and applications are producing progressively larger volumes of system and application logs. Protection of organisational assets and data requires continuous testing and improvement of security monitoring and SA systems. It is often required to improve existing methods and develop new algorithms to meet all the requirements. Verifying novel algorithms is necessary to ensure that the implementation works and is in fact useful in the proposed environment.

RQ2.1. How to discover new knowledge from event logs for improving monitoring systems?

RQ2.2. How to detect covert network communication channels and anomalous network traffic?

Advancing towards autonomous cyber defence With the increasing level of sophistication and automation employed by adversaries, protection of complex interconnected systems may soon become overwhelming for human cyber defenders. The third set of research questions focuses on advancing research towards autonomous cyber defence using deployable autonomous intelligent agents. While this research has been driven by the foreseeable evolution of military systems, it is likely that civil systems, such as complex interconnected systems or the increasing deployment of the Internet of Things (IoT) devices, will soon generate similar demands.

RQ3.1. What is the purpose and rationale of Autonomous Intelligent Cyber-defence Agents (AICAs)?

RQ3.2. What are the required capabilities for AICAs?

RQ3.3. How can an AICA acquire data from the environment to reach an understanding of the current state of the world?

Table 1: Mapping of research questions to corresponding thesis chapters, publications, and contributions.

Research Questions	Chapters	Publications	Contributions
RQ1.1, RQ1.2	3	I, V	1, 2
RQ1.3, RQ1.4	4	V, VI, XI	5, 6, 7
RQ2.1	5	II, III	3
RQ2.2	6	IV, X	4
RQ3.1, RQ3.2, RQ3.3	7	VII, VIII, IX	8

1.2 Contributions

This thesis is based on a collection of publications in a journal and international conferences. The thesis explores the improvement of organisational security monitoring capability and readiness to advance towards autonomous cyber defence systems. The thesis captures recommendations for data collection, transformation, and analysis methods alongside relevant data representation techniques. Furthermore, a data clustering and mining algorithm is proposed and later used to extend a novel NetFlow-based anomaly detection solution. Moreover, Frankenstack and cyber exercise Availability Scoring frameworks are described and verified in several iterations of cyber exercises. Finally, the concept and reference architecture for autonomous cyber-defence agents is proposed for the defence of future military and civil systems.

The main contributions of this thesis are:

1. Recommendations for security analyst's toolkit. The thesis addresses determining essential requirements such as security analysis and reporting tools having important features like drill-down functionality and ability to handle large amounts of data.

2. Recommendations for extracting, transforming, and reporting meaningful security metrics. Defining meaningful security metrics is not easy but following certain guidelines can benefit this process. While the cyber defenders are the primary target audience in this scope, other target groups are briefly discussed.

3. Novel data clustering and pattern mining algorithm (LogCluster) for textual event logs. Publicly available open-source implementation that has been verified with several production system event logs as well as custom logging formats from cyber exercise Availability Scoring logs. The thesis provides various use cases of advanced event log analysis using the LogCluster tool implementation.

4. Covert channel and anomalous network node detection solutions. Extensive analysis of data exfiltration methods using covert channels and their detection capability

by various open-source network security monitoring tools. Additionally, a novel NetFlow-based anomaly detection framework which calculates an anomaly score of each end-user device by analysing network flows using a multi-step and multi-detector process. The framework employs the LogCluster algorithm to mine flow patterns of nodes which had exhibited anomalous behaviour.

5. Review of existing CSX-specific SA and scoring frameworks. While the number of various cyber exercises has been on the rise, there is little information available about the design and implementation of those exercises. More specifically, the thesis demonstrates how SA and scoring systems within the exercises are built.

6. Development of near real-time cyberattack detection, analysis, and representation framework named Frankenstack. The novel framework provides timely feedback and SA for exercise participants to increase the training benefits and learning experience. A cyber security exercise environment was used to verify and test the implementation of the framework.

7. Design of a cyber exercise Availability Scoring system to measure the effectiveness of defences against cyber red team attacks. The scoring system was verified and continuously improved during six annual iterations of cyber defence exercises.

8. Novel reference architecture for an Autonomous Intelligent Cyber-defence Agent (AICA). A concept designed for complex interconnected systems or limited-access networks to which human security operators cannot connect easily or cannot deliver fast enough reactions to cyberattacks. The deployment scope of the agent remains largely for military networks where the variety of authorised network activities are more limited. Nevertheless, those agents need to continuously learn and adapt because the adversary attack patterns are constantly evolving. The thesis author's contribution in this extensive research is primarily represented in the formulation of the agent's *Sensing* and *World State Identification* functions as well as the example AICA use case development.

1.3 Thesis structure

This thesis is divided into eight chapters. The Introduction (chapter 1) provided a brief overview of the problem landscape, presented the reasoning as well as research questions, and described contributions of the thesis.

Chapter 2 provides an overview of related work in subsections following the structure of the topics discussed throughout the thesis. Note, that the following thesis chapters also include a dedicated section for an in-depth comparison of key similarities and differences with relevant work from other authors that has been published after the corresponding original publications in this thesis.

Chapter 3 handles establishing a proper organisational monitoring system in order to start collecting security metrics and integrate data feeds into an open-source SA system.

Chapter 4 describes the use of CSXs to improve and verify complex SA systems and enhance cyber defence operator readiness.

Chapter 5 presents the novel LogCluster log mining algorithm for event log analysis.

Chapter 6 details network security research on the discovery of covert data exfiltration channels and a novel NetFlow-based anomaly detection algorithm.

Chapter 7 describes the reasoning behind the need for autonomous cyber-defence agents and provides a brief overview of the AICA reference architecture.

Finally, chapter 8 provides conclusions and outlines future work.

2 Related work

This section reviews the background and related work within the scope of the thesis. The section is divided into subtopics that are largely inspired by the groups of research questions above. Additionally, research gaps are identified and contributions to address these are emphasised.

2.1 Security metrics, situation awareness systems, and cyber security exercises

Over the years, researchers have struggled to establish what distinguishes a good security metric from a bad one. Furthermore, even good metrics can have distinct meaning for different audiences. For example, a metric that is useful for a technical system engineer, might not mean much to a higher-level executive. Security metrics and situation awareness systems are closely related topics, since after identifying relevant security metrics for a given scope (e.g., company or state), it often makes sense to collect and visualise these metrics with a situation awareness system.

2.1.1 Security metrics research and general-purpose SA systems

There several academic papers, reports, case studies, and documents available on the topic of security metrics. Although the notion of security metrics is defined in a slightly different way in various papers, sources generally agree that a security metric refers to a standard of measurement.

One of the earliest mentions of the term *security metrics* comes from Orlandi [130], however, the term was used in the context of measuring computer software quality in terms of security, risk management and economics of software security. In [44] Geer, et al. identified substantial gaps in measuring information security—questions easily answered in business context had proved exceptionally difficult to answer by technical information security experts. The authors stressed the need for defining rigorous security metrics and establishing a risk-management framework to effectively assess these metrics.

One of the earliest works which describes the implementation of an organisational security metrics program was an article [132] by Payne. Her paper provides the definition and general information on security metrics. The report also discusses potential motivation, as well as some daunting factors regarding committing to a security metrics program. Although the paper does not give technical recommendations for gathering data from IT systems, it does offer general ideas that can help security managers and technical specialists make the first steps towards implementing a security metrics program.

A book [69] by Jaquith (also a co-author of an earlier work [44] mentioned above) describes the properties of a good metric and provides a detailed discussion on how to report and visualise metrics. The book attempts to bridge the management's quantitative viewpoint with the technical approach typically taken by security professionals.

The NIST article by Black et al. [16] emphasises the importance of the surrounding context to realise the full benefit security metrics. They introduce the notion of a *measure*, which is a simple measurable system attribute without a surrounding context, while the metric is composed of one or more *measures* that are combined with relevant *context*.

Security metrics have also been suggested for measuring some specific aspects of security, such as cyber threats. A study paper [42] by Frye et al. discusses possible metrics for characterising cyber threats arising from malicious organizations and individuals. The study conducted in Sandia National Labs proposes the use of a threat matrix model to

assess the threats based on commitment and resources available to the threat actors.

A comprehensive book [22] by Brotby and Hinson offers practical guidance and examples of information security metrics. The authors propose a novel methodology (the PRAGMATIC approach) on how to better define, score and rank security metrics to identify the ones that would be beneficial for various audiences. PRAGMATIC is an acronym of nine criteria for describing and selecting metrics: **P**redictive, **R**elevant, **A**ctionable, **G**enuine, **M**eaningful, **A**ccurate, **T**imely, **I**ndependent, **C**heap. Each criterion assesses different attributes of a metric, using a scale of 0 to 100. Although the scoring is subjective, the authors have provided some guidelines on how this assessment should be carried out. They have also included over 150 metrics as examples, allowing the reader to build his or her own metrics program.

Kotenko and Doynikova [84] propose a technique to dynamically determine appropriate countermeasure selection for ongoing computer network attacks based on relevant security metrics. The approach is established on a set of quantitative interrelated metrics that are calculated based on attack graphs and service dependencies.

In-depth survey papers by Pendelton et al. [134] and Ramos et al. [140] identified the state-of-the-art security metrics and described what properties a desirable metric should exhibit. Both papers agree that quantifying the overall security state of interconnected systems is a difficult problem. Similarly, they conclude that current state of the art only supports decision-making rather than perfectly represents the security level.

Since 2010 the Center for Internet Security has published and continuously developed the CIS Critical Security Controls [24] that provide a set of 20 control groups along with number of proposed measures, metrics, thresholds, and guides that can be used across different organizations to collect and analyse data on security processes and outcomes. In recent versions (since version 6 [23]) the CIS Controls have also adapted their definitions and terminology to differentiate the term measure from the term metrics based on the aforementioned article by Black et al. [16]. Moreover, a recent update to version 7.1 introduced a practical notion of implementation priority depending on the amount of resources and level of expertise available in organizations [25]. In recent years, the CIS Critical Security Controls have also been adopted by the SANS Institute and used in their training courses (e.g., for SIEM and SOC operators) and research [148].

To summarise the main ideas from aforementioned sources, the following common key properties of a good security metric can be identified:

- It is unambiguous and meaningful for predefined purposes while making sense to the human analyst.
- Taking measurements of the metric should not involve significant costs.
- Measurements should be taken consistently using the same methodology, with appropriate time frames between measurements, and preferably with an automated data collection procedure.
- The metric should be expressed on a quantitative, not qualitative scale.

SA systems are typically a set of tools that work together to provide data storage, processing, analytical and visualisation capabilities. SA systems are generally used by monitoring specialists or analysts in various organisational, national, or military security teams (e.g., in CSIRTs, NOCs, or SOCs). Appliances from commercial vendors might offer more advanced functionality compared to free and open-source software solutions, however, since this work focuses on open-source solutions, commercial tools are beyond the scope of the thesis.

A recent paper by [78] Kohlrausch and Brin presented a set of security metrics for quality assurance (QA) and SA that were supplemented by ARIMA (autoregressive integrated moving average) time-series analysis models. They showed that these models can effectively support the security practitioners' typical analysis process (e.g., baselining, detecting trends, forecasting future measurements to detect anomalies) of security and quality metrics. They demonstrated the applicability and usefulness of these metrics by integrating the prototype system implementation into the QA processes and SA systems actively used by CSIRTs and SOC security teams.

A paper [8] by Arendt et al. presented Ocelot, a SA and decision support visualisation tool. The system features multiple viewing modes: visualisations as well as text-based interfaces to facilitate drill-down functionality for investigative purposes. The paper describes four primary modules of the system: a circle-packing visualisation component called Petri Dish, a Filters panel for filtering the view the network based on host attributes, an Alerts panel to display relevant alerts, a Quarantine panel for taking an appropriate defensive action. The authors emphasise their attention for user-centric design for Ocelot. With that in mind, the tool only supports a somewhat limited set of six predefined defensive strategies that the operator can take for any group of network hosts.

Hall et al. have researched the problem of lacking SA communication in cross-domain collaborations. In their work [51] they propose a model and present a use case based on the collaboration between the cyber security and military operations teams. The authors analyse potential caveats and mistakes to avoid when building cross-domain SA systems. Their model relies extensively on Endsley's well-known SA model [39] and aims to aid the design of future collaborative SA systems.

A research paper [45] by Graf et al. aims to replace human input requirement for basic and repetitive SA analysis tasks by proposing a solution that combines modern data analysis techniques with anomaly detection and pre-existing user knowledge base. The authors describe the goal of bridging the critical gap between data collection and situation comprehension—something that the complex human brain can do, but artificial systems currently lack. The fuzzy inference system has a feedback loop that constantly improves existing fuzzy rules based on the decision made. The system was evaluated as a decision-support system in a critical infrastructure in the cyber security domain.

Scaling up from SA systems used by organisational CERTs and SOCs, in a recent paper [15] Bahşi et al. have described a system architecture for a national-level SA that would support country's everyday management and facilitate decision-making at various locations and levels of government, local authorities, as well as the private sector. The paper details how the mapping of information flows between institutions enables the development of such a SA system. Furthermore, their work proposes a systematisation method for individual types of information flows to reduce their heterogeneity. In another recent paper [107], Mõtus et al. discuss the issues related to the modelling process of such a national-level SA system based on the example of [15].

2.1.2 CSX-specific SA systems and the use of CSXs as a proving ground for novel research

Apart from general-purpose SA systems which have been described above, there is one special class of SA systems which has emerged during the recent years—cyber-exercise-specific SA systems. These systems are specifically designed for improving situation awareness during cyber exercises, and while traditional SA systems are oriented to cyber defenders, cyber-exercise-specific SA systems have been designed to provide situation awareness not only to the defending blue teams, but to all involved teams alike (e.g., the attacking red team or the white team in charge of exercise control). A more comprehensive overview

of various team roles is provided below in section 2.1.3.

Moreover, cyber exercises introduce unique requirements. Most importantly, the exercise must support a scoring system which measures the performance of participants. Furthermore, it is important to collect feedback and measure the learning experience of participants during the exercise to improve the future iterations of the exercise.

While cyber exercises introduce several unique requirements, they have nevertheless proven to be an excellent testing ground for validating novel SA systems and security monitoring algorithms. For example, deploying a novel SA system for a live exercise helps to assess its suitability for providing situation awareness during massive cyberattacks. However, cyber exercises are not only useful for conducting live experiments with new systems and algorithms, but they can provide large amounts of valuable data that could often not be acquired from real environments. For example, a day's worth of exercise data might include traces of hundreds of cyberattacks that rarely happen inside organisational networks. Furthermore, the controlled exercise environment enables to better establish *ground truth* that can be later utilised for research validation. While these datasets are exercise specific, they often represent an interesting research topic on their own. For example, the data captured during an exercise enables improving learning efficiency during future exercises or predicting cyber defender's potential performance during real-life situations.

Despite the usefulness of cyber exercises for validating novel SA systems and security monitoring algorithms, they have one unfortunate limitation—the number of large-scale exercises that mimic real-life environments and situations is relatively low. Understandably, designing and organizing such exercises requires a great deal of effort and resources. Since these live exercises typically take place annually over a relatively short period of time, the experiments conducted during an exercise have to be carefully planned and timed, because there might not be an opportunity to re-run the experiment several times. Moreover, elaborate details of military cyber exercises are sometimes not publicised due to potentially sensitive content.

Research by Känzig et al. [71] aims to detect command and control (C&C) channels in large networks without prior knowledge about the network characteristics. The authors leverage the notion that while benign traffic differs, malicious traffic bears similarities across networks. They trained a random forest classifier on a set of computationally efficient features designed for the detection of C&C traffic. They verified their approach using the NATO CCD COE's Locked Shields exercise datasets. Results revealed that if the participants of the LS18 Swiss blue team had used the proposed system, they would have discovered 10 out 12 C&C servers within first few hours of the exercise.

A publication [75] by Klein et al. compares two different machine learning techniques, i.e., the unsupervised autoencoder and the supervised gradient boosting machine methods, on a partially labelled cyber defence exercise dataset. Both techniques were able to classify known intrusions as malicious while surprisingly also discovering 50 previously unknown attacks in the analysed cyber exercise dataset.

In [7], Arendt et al. presented CyberPetri, a redesign of the aforementioned Ocelot tool [8]. CyberPetri was used to provide real-time SA during the 2016 Cyber Defense Exercise. Primary aim of the tool was to provide high-level feedback to network analysts based on exercise target systems' service availability reports. The authors note scaling to large datasets as a limitation. The exercise participants' feedback revealed that the tool was useful for the exercise White Team (WT) high-level decision making, however, technical specialists were more interested in improved exploratory capability for specific events and time windows.

A paper [57] from Henshel et al. proposed a performance assessment model of human cyber defence teams and verified its applicability during the Cyber Shield 2015 exercise. While exercise data was captured during the game, most of the analysis was done post-mortem. For future iterations, the authors stress the need for real-time analysis of the collected data in order to adapt training and assessment methods already during the exercise. The capability to meaningfully analyse the collected data was the primary limiting factor, as operators were not able to keep up with the huge amounts of incoming data.

Research paper [97] by K. Maennel has focused on measuring and improving learning effectiveness at cyber exercises. Furthermore, a recent contribution [96] on team-learning assessment by K. Maennel et al. proposes an unobtrusive method that is based on mining textual information and metrics from situation reports (SITREPs) submitted by teams during cyber exercises. Since SITREPs are regularly filled in by blue teams as a part of the exercise, this approach would enable gathering relevant information without disturbing the teams by conducting regular surveys and questionnaires throughout the exercise. Another recent paper [40] by Ernits et al. discusses how technical event logs and metrics from the exercise game system can be transformed to measure skills and learning outcomes.

The work [26] by Chmelař describes the analysis of Crossed Swords 2020 exercise data using the MITRE ATT&CK framework in order to create reports of red team progress. Although being currently created as part of a post-mortem analysis, the author proposes that the reports could provide in-game overview and visualisation during the exercise.

A paper [72] by Kim et al. proposes a design for a universal cyber-physical battlefield platform to train defending ICS/SCADA systems within national critical infrastructures and smart city solutions. The authors implemented and validated the solution during the 2017 Cyber Conflict Exercise (CCE) as well as the 2018 and 2019 iterations of the Locked Shields exercise [114]. The CCE event itself is described in another paper [73] by Kim et al. In short, the CCE is a newly developed annual live-fire cyber exercise organized by the National Security Research Institute of South Korea. CCE is a competitive exercise between blue and red teams. In addition to explaining different roles and the in-game environment, the paper by Kim et al. discusses the technical and operational challenges related to steering the exercise in the right direction while having only limited control over the red and blue team activities.

There are several recent publications regarding a cyber security exercise called Cyber Czech [28]. A group of authors from the Masaryk University in Brno, Czech Republic have done extensive research on visual analytics, exercise feedback systems, and scoring systems. In a lessons-learned overview [179], Vykopal et al. describe the Cyber Czech exercise which was directly inspired by the larger Locked Shields exercise. Another paper [126] by Ošlejšek et al. evaluates the exercise preparation procedures as well as the systems and data gathered during the exercise using a visual analytics process.

A recent article [164] by Tovarňák et al. describes the benefits of using network and system log data available from cyber exercises. The publication also argues that lack of realistic data in cyber-security research often leads to newly developed methods which are not useful in practice. To remedy this problem, the authors have described and officially released the data [163] collected during the March 2019 exercise iteration.

Further research by Vykopal, Ošlejšek et al. in papers [178, 131] focuses on providing timely feedback for the exercise participants during or directly after an exercise without any delay caused by a lengthy analysis process. The aim is to maximize the learning benefit by providing feedback while the exercise experience is still fresh. They developed and validated an interactive visual feedback tool that presents a personalised timeline of exercise

objectives, relevant events, and scoring data for each exercise participant.

Finally, a recent publication [125] Ošlejšek et al. takes another look at the visual analytics process of hands-on cyber security trainings and proposes a conceptual model to improve such cyber defence exercises. The discussion section brings out four key challenges that must be overcome in order to improve the future visualisation domain: visualisation tools lack post-exercise reflection and SA capability, insight into learner's educational impact is still highly limited, general purpose monitoring and analysis tools lack exercise-specific requirements, and data processing and correlation cannot keep up with data collection.

2.1.3 CSX background

The topic of organising CSXs was handled in a recent PhD thesis [18] by Blumbergs, however, for the sake of completeness, a short description of exercises that are strongly linked to this thesis will be summarised here as well. Blumbergs focused primarily on researching red team responsive operations and developing specialised CSXs to train such skills in an organised form. Comparatively, this thesis complements his work and makes use of such exercises to predominantly verify novel algorithms and integrations of tools. Additionally, an important segment of the thesis discusses the design and implementation of CSX-specific monitoring and scoring systems to measure and improve the learning effectiveness of participants as well as the overall exercise quality in future iterations.

Locked Shields (LS) [115] is an annual game-based real-time CSX that has been organised by the NATO Cooperative Cyber Defence Centre of Excellence (CCD COE) since 2010. Although the primary exercise goals are learning and cooperation, there is still a competitive element in the game—the defending blue teams (BTs) are scored on their operational performance in several different interdisciplinary categories, e.g., defending against cyberattacks, incident reporting, situation reporting, responding to scenario injects, and keeping their systems functional to the end users. Scoring and team feedback are integral elements of the exercise, because participants need to understand how well they performed with regards to the tasks set out for them.

As LS is a defence-oriented CSX the primary training audience of the exercise are the BTs. Other participants represent one of four additional teams: red team (attacks), yellow team (situation awareness), green team (exercise infrastructure), and white team (exercise management, strategy game, media, etc.). BTs undertake the role of rapid-reaction teams sent to assist the fictional country of Berylia, which is in a long-standing conflict with another fictitious country, Crimsonia, represented by the red team (RT). Although all teams have to prepare long in advance, the exercise takes place over just two days (approximately 8 hours per day) of intensive gameplay.

The exercise has been growing remarkably—in just a couple of years the number of participants has risen from a few hundred to more than a thousand. Training audience has comprised more than 20 BTs in the past few years. Each BT is responsible for maintaining the continuous and secure operation of about 150 physical and virtual hosts with several scored services on each host. The exercise scoring and SA system keeps track of all check results and continuously calculates the uptime for each service. The availability points account for approximately one-third of the total positive score in the exercise. This is roughly the same as the amount of negative score that teams can get for successful attacks carried out by the RT. Therefore, teams need to balance the strategy between ensuring availability and enforcing security wisely.

The BTs are tasked to defend a wide variety of conventional IT infrastructure as well as special-purpose industrial systems. For example, Windows and Linux servers, Windows

and Linux workstations, FreeBSD servers and firewalls, industrial programmable logic controllers (PLCs), professional power management systems, drone flight controllers, and 4G/LTE gateways. In addition to system administration and system hardening tasks, teams are also assigned forensic and legal challenges as well as various injects from the white team (WT). Therefore, BTs must incorporate specialists with a variety of skills to encompass all necessary expertise.

Crossed Swords (XS) [119] is another annual CSX that has been developed and organised by the CCD COE since 2014. Overall, XS is closely related to its older sibling LS, however, the principal difference is that the roles of the training audience are reversed—instead of the BTs, it is now the RT that is playing out a responsive cyber-kinetic scenario. RT has to work as a united team, uncover an unknown game network, and complete a set of technical challenges and collect evidence for attribution. All this must be accomplished while staying as stealthy as possible.

On the tactical level, RT consists of multiple sub-teams based on the skills and engagement specifics. Specifically, the network team targets network services, protocols and routing; client-side team exploits human operators and maintains foothold in the internal network segments; web team targets web services, applications and any associated databases; digital forensics team performs data extraction and artefact collection; and kinetic forces team provides support in operations that require a kinetic element such as physical surveillance, intelligence collection, hardware extraction, forced entry, target capture, etc. It is important to note that XS is not a capture-the-flag competition. These sub-teams are not competing with one another, but rather serve as specialised divisions of a single military detachment. The scenario is constructed in a way that these sub-teams must coordinate their actions, share intelligence, pivot between sub-objectives, and cooperate when executing attacks to complete the exercise goals.

2.2 Detection capability improvement and validation

This subsection describes work related to various detection concepts, methods and algorithms presented in this thesis.

2.2.1 Event log analysis and knowledge discovery

Modern systems often produce large volumes of system and application logs. Event pattern mining is an important log management task in order to discover frequent patterns of events. However, in some cases the opposite applies—infrequent events might be the most crucial ones to look out for while being the easiest to miss in a large number of events.

Vaarandi developed one of the earliest event log frequent pattern mining algorithms called SLCT [167] which has been applied in various log monitoring domains such as intrusion detection alert processing [177], detection of recurrent fault conditions [144, 143] and event log visualisation [99]. These publications have also identified several shortcomings in SLCT, such as inability to detect wildcard suffixes in line patterns, sensitivity to word position shifts as well as delimiter noise, and needless overfitting when pattern mining is conducted with low support thresholds.

Makanju developed a hierarchical clustering algorithm IPLoM [100, 98] that starts with the entire event log as a single partition and splits it into sub-partitions during three iterative steps after which a line pattern is derived for each partition. Splitting depends on various criteria, such as the number of words in a line as well as affiliations between word pairs. IPLoM algorithm does not need a user-specified support threshold but takes several other threshold parameters which impose fine-grained control over partition split-

ting. Like SLCT, IPLoM is sensitive to shifts in word positions, however, unlike SLCT, IPLoM can identify wildcard suffixes in line patterns.

Reidemeister et al. developed a methodology that uses event log mining techniques for diagnosing recurrent faults in software systems. The work [144, 143] describes a modified version of SLCT, that handles delimiter noise and shifts in word positions. To achieve this, the results from SLCT are clustered further with a single-linkage clustering algorithm which employs a variant of the Levenshtein distance function. Detected knowledge is then harnessed for building decision tree classifiers to identify potential faults in future events.

A paper [52] by Hamooni et al. describes a fast pattern recognition method for log analysis called LogMine. It is an unsupervised framework that only makes one pass over the input logs and generates a hierarchy of patterns. The framework can be scaled out to multiple parallel processing nodes. For measuring the performance of LogMine, the experiments were conducted using three proprietary and three publicly available log datasets.

Du and Li developed an online streaming event log parser called Spell [34]. The name stands for **S**treaming **P**arser for **E**vent **L**ogs using an **L**CS (longest common subsequence). Searching is accelerated by subsequence matching and use of a prefix tree. The comparative experiments were performed using the well-known and long-lived high-performance computing (HPC) cluster logs from the Los Alamos National Laboratory and the BlueGene/L (BGL) supercomputer logs [166]. The authors state that Spell was able to outperform the efficiency of offline methods such as IPLoM and another clustering-based log parser [43] proposed by Fu et al. The authors recommend pre-filtering to improve efficiency of the method even further.

Another online streaming log parser called Drain [55] was proposed by He et al. The authors report a 51%–81% improvement in runtime compared to the Spell tool. The search process is accelerated by using a fixed depth parse tree, which uses purpose-designed rules for parsing log messages. Evaluation log datasets comprised the aforementioned HPC and BGL logs as well as Hadoop File System, Apache Zookeeper, and Proxifier logs.

Messaoudi et al. have tackled the problem of log message format identification by coming up with approach called MoLFI (Multi-objective Log message Format Identification) [103] to search for a Pareto optimal set of message templates for logs. They employed a search-based solution based on the multi-objective genetic algorithm and trade-off analysis. MoLFI was evaluated with six log datasets of various size and type. Accuracy and performance were compared against the IPLoM and Drain tools. The authors reported higher accuracy and scalability compared to the previous approaches.

Finally, a comparative evaluation study [183] by Zhu et al. offers a detailed overview of 13 automated log parsers released over the last two decades. Among others, they have also re-analysed all of the aforementioned event log analysis tools to offer a broader comparison of their capabilities. Furthermore, the authors have released a public *Logparser* repository that comprises all 13 tools allowing to conveniently compare their performance for different types of logs [94]. Log datasets used in the evaluation process included 16 sources of various size and type. Additionally, the authors have provided an overview of ten industrial log analysis and management tools, making the effort to bridge the gap between industrial and academic research in this field.

2.2.2 Network security and anomaly detection

Over the years, the number and level of sophistication of cyberattacks against end users have grown significantly. Unfortunately, traditional rule- and signature-based technologies (e.g., network IDS/IPS and next generation firewalls) have their drawbacks and are only able to detect previously known attack patterns. Furthermore, the adoption of Inter-

net Protocol Version 6 (IPv6) has opened up a wide scope of new potential attack vectors, such as network IDS/IPS evasion and data exfiltration [11, 108].

Niemi et al. presented Evader [123] which can create combinations of multiple atomic network level evasion techniques (*inter alia*, IPv4/IPv6 dual-stack evasions). They found that combining multiple evasion techniques enables potential evasion of detection by majority of network IDS/IPS solutions.

A recent paper [101] by Mazurczyk et al. investigates various IPv6 covert channels deployed in the wild. The authors evaluated six different data hiding techniques for IPv6 including their ability to bypass some IDS solutions. They conclude that the hiding capacity of real networks (e.g., using cloud providers such as Digital Ocean or Amazon Web Services) is less than what is proposed by theoretical research. They also criticise the lack of related work which would employ IPv6 protocol and real networks in their covert channel detection research.

Considering the aforementioned limitations of traditional rule-based technologies, the use of network traffic pattern analysis and anomaly detection of end user nodes has become an important research problem that has been tackled by many. Furthermore, full packet capture in high-speed networks is often infeasible due to substantial computational requirements. Even with NetFlow, sampled traffic is often used in backbone links instead of analysing all the traversing flows.

In [181], Wendzel and Zander employed supervised machine learning to detect protocol switching covert channels (PSCCs). They monitored the number of packets between network protocol switches and the time interval between switches. Their solution was able to achieve a 98-99% detection accuracy for high bitrate transmissions, however the authors conclude that for busy networks this might still result in a high number of false positives as vast majority of the traffic is usually normal traffic.

An article [21] by Brauckhoff et al. presents an algorithm which generates histograms of flow features (e.g., source or destination ports) between fixed measurement intervals. Each histogram effectively serves as a flow feature distribution. An alert is returned for the operator if the Kullback-Leibler distance between current and previous interval distributions exceeds a given threshold. Suspicious flow records are then processed with the Apriori frequent itemset mining algorithm.

A paper [169] by Vaarandi proposed two unsupervised anomaly detection algorithms that are applicable in organizational networks. The first algorithm analyses recently used network services and maintains a behaviour profile for each node. An alarm is raised if a node connects to an unusual service. The second algorithm detects clusters of nodes that use the same set of services on a daily basis. An alarm is raised if a node's behaviour deviates from its prior group.

A paper [48] by Grill et al. investigates malware that uses domain generation algorithm (DGA). The authors suggest using a statistical approach and measure the ratio of DNS requests to the number of IP addresses contacted by the host. The authors expect the infected hosts to try to resolve more domain names during a short period of time without actually connecting to a corresponding amount of IP addresses.

Zhou et al. have presented ENTvis [182], a tool which divides network flows into time frames, and for each period calculates both source and destination IP and port entropies. The results are visualized with several techniques (e.g., visual clustering) which enables operators to detect and investigate anomalies.

Hofstede et al. published an extensive tutorial [58] providing an in-depth overview of network security monitoring using NetFlow and IPFIX. Moreover, a paper [59] by Hofstede et al. introduces a near real-time intrusion detection and mitigation approach that

integrates with NetFlow and IPFIX flow exporters. The solution uses the exponentially weighted moving average (EWMA) based algorithm for DDoS detection by tracking the number of flows. In case of an unexpected change, the algorithm can create firewall rules for blocking malicious traffic.

NetFlow analysis and machine learning have been used for DDoS detection in high-speed backbone network links. A paper [60] by Hou et al. proposes random forest classifiers for DDoS detection. The solution proved efficient in analysing real ISP NetFlow logs and achieved the classification accuracy of 99% and less than 0.5% false-positive rate in lab experiments containing real benign traffic and simulated DDoS attacks.

Finally, the paper [71] by Känzig et al. which was mentioned under the 2.1.2 cyber exercises subsection is relevant in this section as well. To reiterate, the authors proposed an approach train a random forest classifier on an efficient set of network traffic features to detect C&C channels in large network without prior knowledge of the network. They reported 99% precision and over 90% recall in near real-time detection while keeping the resource requirements within realistic bounds. The authors propose further evaluation in future LS exercise iterations.

2.3 AI and autonomy in cyber security

Topics such as artificial intelligence, autonomy and machine learning have been a research subject for many decades with some underlying foundations (e.g., probability theory) dating back centuries. AI and autonomous systems research has initiated various academic conferences [62, 63, 64] and practical applications such as signal processing, underwater exploration, autonomous navigation, and more recently also cyber defences.

Moreover, during the last decade, the use of these terms has skyrocketed as the high technology industry has begun boosting the technological advances at an accelerated pace. Although not directly related to cyber security, the general tendency of introducing autonomous technology is definitely on the rise. Autonomous vehicles (e.g., Tesla's Autopilot [157]) and robots (e.g., by Boston Dynamics [20] and Starship Technologies [155]) are being tested in public and will likely abide next to us in everyday life even more than now. Additionally, extensive research and development progress is made in the development of military-purpose autonomous vehicles (e.g., by Milrem Robotics [105]). Efforts towards autonomy is not pursued only in vehicles or robots—it can be true for any technological device or even software. So called *smart technologies* and IoT devices are permeating our lives at an increasing speed, resulting in smart household appliances, smart buildings, and smart cities [158].

When looking at all this technology from the cyber security perspective, there is one thing that is in common for most if not all such modern systems—they are largely interconnected and depend on a myriad of ICT components to function properly. Moreover, the security of such highly complex devices is difficult to audit and assure.

Even if technology industry is moving towards developing autonomous systems, the process of detecting and especially countering new cyberattacks and malware infection vectors is unfortunately still largely artisanal. In [87], Kott and Theron claim that today's cyber defence tools are still not capable of planning and executing dynamic responses to attacks, nor are they able to plan and take action after an incident has occurred—these core tasks are still left to human operators (e.g., cyber defenders, incident responders, and administrators). However, given the complexity of today's systems and networks, fully automated attack detection and mitigation would considerably increase the speed of security incident resolution and lessen the damage from such incidents. In this section, related work on automated cyber defence with intelligent agents is discussed.

2.3.1 Autonomous systems research

The focus of the following related work is on the recent use of AI and autonomous systems for cyber security.

In one of the leading textbooks in the field of AI [146], Russell and Norvig present the idea of intelligent agents that can perceive the environment and perform actions. Different agents may fulfil various functions depending on the environment, tasks, and goals. *Learning* is one of the key capabilities of the agents and is necessary to avoid manually constructing large common-sense knowledge bases—an approach that has not fared well up until now.

A paper [137] by Preden et al. presents a distributed sensor system where the primary computation is pushed to the edge of the network where the sensor data is generated instead of collecting everything centrally. This is especially useful for low bandwidth networks (e.g., in deployed military operations). This allows the individual edge systems to maintain a high level of autonomy. In addition to central communication, sensors are able to collaborate among themselves by utilizing the *Fog Computing* paradigm and applying Data to Decision concepts.

De Gaspari et al. argue that instead of building complex honeypots identical to the production systems they are supposed to defend, production systems themselves should be designed in a way to provide active defence and deception capabilities. The paper [30] proposes a system called **A**ttackers **H**indered by **E**mploying **A**ctive **D**efense (AHEAD), which slows down the attacker and provides incident responders with enough time to identify and monitor the intruder in action.

Guarino studied the potential implications arising from the use of autonomous agents in offensive cyber operations. The paper [49] provides an overview of underlying science, taxonomy, and state of the art of intelligent agents as well as the reasoning behind the use of agents. Furthermore, the author has examined the potential legal implications and how the use of agents might fit into existing legal and doctrinal frameworks. The paper concludes that while truly autonomous agents did not exist at the time of writing, it is only a matter of time when they would be developed and deployed, and it would be best to be prepared both technologically as well as legally.

Tyugu discussed the increasingly relevant research topic of intelligent autonomous cyber weapons. The paper [165] considers several unintended critical behaviours of autonomous cyber weapons: situation misunderstanding, command misinterpretation, loss of contact, and establishment of unwanted coalitions. Considering a longer timeframe, with the growing level of intelligence and autonomy these systems will be increasingly difficult to control—essentially leaving the human out of the control loop. The paper stresses that the developer of the weapon must put in appropriate control mechanisms that would allow to override the weapon under any circumstance. Furthermore, in recent years, interest in the topic of autonomous weapons has been extending well beyond technical discussions, engaging policy [56] and legal researchers [180, 92] as well as inspiring workshops [17] and debates [9].

2.3.2 Autonomous systems' tournament

One of the largest autonomous system competitions was set up by Defense Advanced Research Projects Agency (DARPA) in 2016. This subsection describes the DARPA Cyber Grand Challenge as well as the Cyber Reasoning Systems (CRS) and strategies implemented by the top three teams in the DARPA CGC.

Compared to previous endeavours, DARPA took a more practical approach when they launched the Cyber Grand Challenge (CGC) [31], a competition to develop automated,

scalable, machine-speed vulnerability detection and patching systems capable of reasoning about flaws, formulating patches and deploying them on a network in real time. Over 100 participants joined the competition while only seven made it to the final event in the summer of 2016. Prize pool of over 3.5 million USD for the top three teams sparked interest and pushed teams to solve the tough research challenges. Each team had to develop a CRS that would automatically identify software flaws, and scan a purpose-built, air-gapped network to identify affected hosts—all this autonomously without any human interaction. The teams were given approximately two years to prepare for the challenge [14].

For 12 hours CRSs had to autonomously protect the target hosts, scan the network for vulnerabilities, and maintain the correct function of software [32]. According to [122], the score calculation algorithm comprised three components: system availability, security/vulnerability, and attack success evaluation. Each component was actively scored in a five-minute time window to determine the top performing teams. The teams were not provided with exact details of the scoring logic to deter their attempts to trick the scoring system instead of playing the game in a graceful manner. Furthermore, since players could send malicious input to other CRSs during the game, teams had to think of implementing counter-autonomy countermeasures (e.g., minimising attack surface, isolating faults, and sandboxing application) for their systems in order to be resilient to such adversarial techniques.

An article [150] by Shoshitaishvili et al. describes the CRS called Mechanical Phish that was created by the team Shellphish. The authors reveal the modular architecture of their CRS and state that the entire code base for Mechanical Phish was about 100,000 lines of code (LoC). Their central software component was called `angr` [3, 151], a binary analysis framework integrating various different binary analysis techniques. The authors hold their reservation about the efficacy of deploying Mechanical Phish in real world systems, as this was just an initial step, a prototype towards an autonomous cyber agent deployed in a realistic scenario.

Xandra, the CRS created by the team called TECHx, is described in the paper [122] by Nguyen-Tuong et al. The essential component of their system was the Helix binary analysis platform along with Zipr [54], the static rewriter framework capable of patching arbitrary binaries in a space- and time-efficient manner. The paper also describes the network packet capture analysis and use of network IDS to monitor and modify network traffic. The team takes pride in the efficiency of their system by bringing out the fact that Xandra consumed the least amount of power than any of the competitor system.

Mayhem, the CRS developed by the team For All Secure that won the DARPA CGC, is described in a paper [14] by Avgerinos et al. For its defence, the system made use of two different binary patching techniques, namely the full-function rewriting and the more conservative injection multi-patching approaches. Their system did not use a network IDS component because the developers reasoned that maintaining a signature-based ruleset against polymorphic exploits would not prove good enough. Furthermore, the paper includes an extensive discussion on choosing the correct strategy throughout the cyber defence game (e.g., how to make the best use of limited resources or how to choose a correct patching procedure).

The binary patching capability as well as patching strategy selection seems to have played a critical role in determining the winner. All three top teams describe the strategy selection as one of the cornerstones of their success or failure. The teams seemed to have distinguished between three main strategies: always patch everything as soon as possible, patch only if exploited, or never patch. The first (always patch) might seem reasonable,

however, not all services were attacked in every scoring round and many teams caused a service to fail because of a poor patch even if it was not attacked. The second (patch if exploited) seems to have been the more reliable option within the rules and constraints of the game. Mechanical Phish (III place) chose the *always-patch* strategy while Xandra (II place) and Mayhem (I place) opted for the *patch-if-exploited* strategy.

3 Security metrics and situation awareness systems

Measuring organisational security has long been in the interest of security managers, analysts as well as researchers. Having well-established history of quantifiable security logs and metrics allows to better understand organisational security posture, predict future trends based on historic data analysis, support decision-making in setting up proper security mechanisms, and respond more effectively to security incidents. Furthermore, in order to take action, presenting this information to operators (analysts, security experts, administrators, etc.) in a clear, understandable, and actionable manner is of crucial importance. Finally, these systems alone are not particularly useful without skilled operators to constantly observe, maintain and improve these systems. Therefore, operators have to be trained in order to develop their skills as well as kept up to speed with state-of-the-art technology.

However, it is often unclear how to effectively measure security within the context of a state, organisation, computer network, or even a single system. The identification of relevant security metrics for a given scope is a difficult task but does not serve the final purpose without a proper technical solution to collect, analyse, and visualise them. Therefore, implementing an SA system and customising it according to the specific requirements of the security analyst's team is a sensible and important next step. While many technical security monitoring tools provide data collection and generic representation functionality, they often lack a rigorous and systematic approach in the context of security metrics and SA reports.

While the amount of existing publications on this topic (analysed in section 2.1.1) is substantial, the proposed methodologies often provide generic recommendations which lack concrete examples or specifics for implementing a security metrics collection system as well as practical advice for integrating it with SA systems. There is an apparent research gap in utilising the proposed recommendations in practice. Even if modern monitoring and SA tools have the necessary functionality to customise the solutions according to one's requirements, operators seem to be struggling in applying the theoretical advice into practice.

This chapter addresses the gap by describing the process of extracting, transforming, and reporting meaningful security metrics that are facilitated by following certain practical guidelines. While the cyber defenders are the primary operators and target audience in this scope, other target groups are briefly discussed to illustrate the different information requirements of each group. Additionally, the work discusses essential functionality requirements for security analyst's toolkit in order to enable them to efficiently analyse and report the information gathered into the SA system.

Furthermore, a good security metrics and SA system can only be built on a solid monitoring solution that encompasses event logs, network data, and IT asset management. Thus, many of the underlying information sources and data requirements will also be addressed. To exemplify specific use cases, the thesis makes use of an organisational production metrics and SA framework used for extracting and reporting security metrics (described in detail in Publication I). The framework employs four primary data sources—IDS/IPS alert logs, NetFlow data, workstation logs, and other events (e.g., infrastructure, service, or application logs)—that are generally available and used in organisational environments. The framework demonstrates how common data sources can be leveraged for gathering security metrics and how this information can be presented in open-source SA systems.

This chapter addresses research questions RQ1.1 and RQ1.2. The research was originally published in Publications I and V. Thesis contributions 1 and 2 are described.

3.1 Production framework architecture

The described metrics and reporting framework had been implemented in a large institution which had a complex computer network consisting of thousands of workstations, servers, network devices, IDSs, firewalls, and other nodes.

According to the common identifiers of good security metrics (as identified in Publication I as well as in related work subsection 2.1.1), the collection of metrics should not incur significant cost and should preferably be automated in order to have consistent measurement methodology and interval. Thus, the framework leveraged a centralised design, i.e., all of the data was collected centrally for analysis and long-term storage. This approach significantly reduced the complexity of the framework, as edge nodes only had to submit event logs and network data once, instead of the reaching out to edge nodes over the network during the analysis phase. Furthermore, this enabled performing the resource intensive computations on the central dedicated hardware instead of placing additional load on the edge nodes and potentially interfering with active users.

The framework processed approximately 100 million log records per day (excluding NetFlow data). Handling such volumes of data requires deliberate planning and efficient tools. The system makes extensive use of the syslog and NetFlow protocols for collecting event logs and network flow data. The rsyslog tool was used to receive, parse, and forward syslog events. For event logs which do not natively support the syslog protocol, applicable conversion tools were used. For example, NXLog tool [124] was used to convert and transmit Microsoft Windows Event Log messages to the central log collection servers. Logstash [38] was primarily used for parsing more complex syslog events and forwarding NetFlow data to Elasticsearch. All data from the collectors was forwarded to the Elasticsearch [36] for indexing and support for extensive search functionality. The simplest way to access and explore the data stored in the Elasticsearch was using the Kibana tool [37]. It provided end users a flexible interface for writing search queries, drilling down into raw data, defining visualisations, and composing various dashboards. Event correlation tasks were handled by SEC [171], which also extracted several security metrics and sent them to the Graphite tool [159], a numeric time-series storage and graphing tool. In addition to collecting NetFlow data to Elasticsearch and visualising it with Kibana, NetFlow was also visualised with NfSen [121], a simple yet powerful graphical frontend for comprehensive NetFlow analysis.

3.1.1 Extracting security metrics from IDS/IPS alert logs

Organisations typically operate various IDS/IPS solutions to protect their network. Logs from IDS solutions are commonly collected and critical alerts are displayed to the security operators for additional verification and potential escalation. While an IPS can defend a network independently, they still require regular supervision and updates, because any misconfiguration or outdated signature may result in legitimate traffic being dropped or cyberattack left unnoticed. While these alerts are of crucial importance to organisation's security monitoring, it makes sense to analyse these logs in a slightly wider scope to identify trends and patterns in time. For example, a steady number of low severity alerts over a longer period can signify a low threshold probing activity by an APT. In such security systems it is common to determine the amount IDS alarms per hour, day, week, and present this as a time-series data to the security analyst. Additional filtering attributes can provide more meaningful results. For instance, describing additional metrics based the type of alert (e.g., malware-related alerts, perimeter scanning activity, etc.) or classification of affected services and networks (e.g., public services, internal services, workstations, etc.). Any unexpected change or anomaly in the IDS-related metrics should be investigated by

the security analyst.

There are several caveats that one has to consider when working with IDS/IPS rules and alerts. Many available rulesets (e.g., the Emerging Threats ruleset [138]) are published with a large number of generic rules that are enabled by default. For instance, there are rules that alert on various software update services or even the ICMP ping activity. Depending on the monitored environment these rules can make sense—for example, in a network segment with just Microsoft Windows workstations it would be wise to monitor for network activity emanating from non-Windows machines. The presence of such activity could indicate a rogue or malicious device connected to the network. However, having such rules might prove to be too noisy in some volatile environments where people are allowed to connect their personal devices into the network. To clarify, these alerts should not be considered as false positive, because they work as intended, however, depending on the type of nodes and activity allowed on the network, some rules would have to be tuned or disabled for specific environments. This is a normal rule-management process typical for any IDS/IPS ruleset.

Unfortunately, IDS signatures are also known to cause false-positive alerts. For example, a rule that was previously known to function as intended might suddenly start to produce many alerts. This might be inadvertently caused by external factors such as software updates or changes in configuration. For example, some anti-virus definition updates have been known to cause false-positive alerts because the downloaded definitions are incorrectly classified as malware retrieval. Consequently, if too many false positive IDS alerts occur, it can seriously affect the quality of collected security metrics. This introduces a requirement to be able to exclude certain alerts from further processing. Additionally, there is the issue of background noise produced by frequent bad traffic emanating from well-known internet scanners and worms. To handle such cases, an IDS alert classification system could be used to distinguish noise from more relevant alarms. Vaarandi and Podiņš have published an algorithm in [177] for that purpose based on frequent itemset mining and data clustering techniques. This method would allow to present the security analyst only with relevant alarms.

3.1.2 Extracting security metrics from NetFlow data

Collecting network traffic statistics using NetFlow can provide several interesting metrics. Although IDS/IPS solutions can analyse and collect data in an analogous format compared to the NetFlow protocol, the latter does provide some advantages over the more comprehensive IDS/IPS counterpart. The NetFlow protocol does not analyse and store full packet payloads and is therefore more lightweight in terms of computational and storage requirements. Furthermore, the NetFlow protocol is supported on most professional network devices (e.g., switches, routers), whereas installing a full-blown IDS/IPS would typically require dedicated hardware. This makes collecting network flows more feasible even inside smaller private networks. On high-speed backbone links, where full NetFlow collection would consume too many resources, traffic sampling can be set up to only process a small portion (e.g., 0.01%) of all packets. Contrariwise, sending sampled network traffic to an IDS/IPS would not provide a similar result and would instead hamper the system. This happens because those tools are designed to assemble full network sessions in order to reconstruct and analyse the payload that was transmitted [91, 147].

NetFlow monitoring enables to set up blacklist-based security metrics that are useful indicators of any data exchange with known compromised, malicious, or suspicious IP addresses or domains in the internet. The first potential step would be to correlate NetFlow data with publicly available blacklists such as Proofpoint's Emerging Threats Intelligence

[138] and abuse.ch [1]. Another possible method includes observing the NetFlow field which holds the union of TCP flags in order to detect illegal flag combinations indicating abnormal or malicious traffic. For instance, in a legitimate network flow a TCP FIN flag should never appear without a TCP ACK flag. This method would allow to specify a metric indicating the number of distinct sources (i.e., IP addresses) of abnormal traffic per hour (or day). Furthermore, such metrics should separately keep track of external (i.e., internet) and internal hosts because malicious activity from external sources can typically be expected but detecting such activity in an internal network should be considered even more alarming.

3.1.3 Extracting security metrics from workstation logs

Workstations hold a crucial role in organisational security. Since end-user workstations can typically access restricted internal services and information, they are often the initial targets for malware and targeted attacks. Therefore, it is essential to monitor and create security metrics of their activity. Workstation logs contain a significant amount of security information (e.g., login failures, antivirus alerts, installation of new software and/or services, modification of protected system files, etc.) that can be used to craft valuable security metrics. The relevance of some metrics depends largely on organisational policies—for instance, hardware events (e.g., plugging in an USB stick) might be normal in most organisations, however, for organisations handling classified information, such events might be extremely critical.

While there is a wide variety of metrics that can be produced from workstation logs, for the sake of brevity we will only discuss the most relevant type: user account monitoring and control. In many cases it enables effective detection of system compromise, malware propagation as well as malicious insider activities. According to the CIS Critical Security Controls [24] it is a critically important cyber security discipline. As such, it is common to monitor events indicating both successful and failed login attempts into workstations. For example, consider the metric observing the number of unique workstations or accounts with login failures in each timeframe. A sudden upsurge in the number of hosts or accounts might signify an unauthorised account probing activity, possibly to gain unauthorised access to data. Furthermore, monitoring successful remote logins from unusual (e.g., non-administrative) sources can potentially reveal lateral movement of an attacker or a malware spreading in the network.

Although the event log format and collection techniques differ between Windows and UNIX-like platforms, the mentioned event types are not specific to Microsoft Windows and can be similarly observed for other workstation platforms (e.g., Linux, MacOS[1]). Furthermore, it often makes sense to extend the default security logging functionality. For instance, Windows Event Log can be supplemented using the Sysmon [104] utility provided by the Microsoft Sysinternals team. There are a number of recommendations as well as example configurations available for Sysmon [53, 156]. Similarly, for Linux there are various built-in security tools (e.g., AppArmor [6] or SELinux [141]) that provide additional security and can log relevant events.

Unfortunately, the analysis of workstation logs is often under-utilised. Primarily because workstations often generate large number of logs which makes collecting them for analysis an expensive process, especially in large organisations with hundreds or thousands of workstations. This situation is further exacerbated by extending the default logging functionality with additional tools, therefore careful considerations are necessary

[1]Due to limited access to MacOS-based hosts, the security monitoring of MacOS is not discussed it this thesis.

when designing and configuring the logging infrastructure. Collecting all available logs would of course allow for a more in-depth analysis, however, identifying the most relevant types of logs enables creating filtering conditions before transmitting and storing the logs at the central log server, which in turn can help keep the log volumes under control [4].

3.1.4 Extracting security metrics from other event logs

Furthermore, there are several other categories of logs available in most organisational networks. For example, this includes events from hardware device logs, server OS logs, service-specific logs. Extracting hardware logs from routers, switches, servers, but also from other network-capable devices is often overlooked as a source of information and a security measure. While these logs mostly contain messages about hardware health and regular maintenance procedures, they may also reveal unauthorised physical access and tampering with the device. Although acquiring logs from some proprietary device may prove challenging, hardware health messages should be kept track of nevertheless, as these provide a metric of the overall infrastructure health and potentially avoiding critical hardware failure resulting in loss of availability.

Server OS logs might be collected in a similar OS-dependent manner compared to workstation logs, however, when it comes to analysis and correlation, the characteristics that are induced by user activity in the logs are usually quite different. When workstations typically operate in a graphical environment and run a variety of applications then servers are commonly operated in a *headless* fashion and are managed via a remote terminal. Furthermore, users ordinarily do not log directly into servers, but rather access remote services hosted by the server. Therefore, any user activity on servers should always correspond to administrative activity. There are of course exceptions such as organisational terminal servers or administrative *jump hosts*.

Regarding the myriad of potential services hosted on the servers, it is reasonable to make use of the IDS/IPS solution to monitor for known security vulnerabilities being exploited over the network (this approach was already explained in subsection 3.1.1). Additionally, it also worthwhile to look for suspicious event patterns or unusual combinations of events. Consider a web server scenario where under normal circumstances most of the HTTP client requests receive the normal 2xx and 3xx (i.e., successful responses and redirections respectively) response from the web server, while non-OK (e.g., 4xx client error) response codes might occur from time to time when the clients request a non-existing or a forbidden resource. However, when the rate of error messages increases compared to normal traffic, it might signify a reconnaissance scan. Although this kind of scanning happens regularly on public facing web sites, it is still sensible to keep an eye on the rate of such scans, because they can help in assessing the threat level for individual services.

3.2 Discussion of open-source SA system capabilities

Our organisational framework takes a step further from generating static metrics reports (e.g., sent regularly via e-mail). The thesis argues that organisations should be operating an SA system that is capable of searching, analysing, aggregating, and visualising large amounts of diverse data.

Although processed events and metrics are often generalised and correlated with multiple data sources, one essential requirement for such tools is that they should enable the operator to understand the presented results and the underlying data transformations by facilitating additional queries and a drill-down capability into the raw data. Consecutively, this allows the operator to spot anomalies, potentially revise collected metrics, and

improve their meaningfulness.

Ideally, a single tool or interface would be the best for keeping the focus of operators. However, as the number of various data types (e.g., time-series data, network data, event logs) builds up, it becomes increasingly difficult to create an SA system which excels at every data type and task. While this may be possible in case of limited scope or specific tasks, the unfortunate reality is that security operators often have to query or cross-reference multiple tools while inspecting collected data or investigating a security incident.

Technology has been advancing at an increasing pace and is continuously improving the potential to provide better security, monitoring capability, and autonomy. However, when it comes to cyber security monitoring and especially the more advanced metric and situation awareness systems handled in this thesis, technology alone is not yet enough to operate an effective security metrics and situation awareness programme. Operators manning those systems require training to maintain and further develop these complex systems on a regular basis. Therefore, the need for more skilled cyber security specialists is increasing. CSXs provide a way to provide a training in a realistic environment, yet without the risk of breaking live production systems. The operator training aspects will be discussed in more detail in section 4.2.

3.3 Verifying system capabilities

For development and research purposes it generally makes sense to test the systems and their components in a more controlled environment. For direct performance comparisons between various algorithms the use of static or generated datasets is justified, however, their use can also lead to inherent optimisation for that specific dataset and potentially deterring the focus from volatile production environments. This may result in the creation of systems and algorithms that work well with specific type of data but experience difficulties as any unexpected or non-standard input is received.

To converge research and practicality, this thesis actively promotes the use of realistic yet supervised environments in research and development (R&D). Environments for such research endeavours are potentially available during various CSXs, however, their use for research purposes should be carefully planned beforehand. To ensure practicality and support closing the aforementioned research gap, most algorithms and tools developed as a part of this thesis have been tested and verified under realistic conditions during annual CSXs or with the datasets collected during those exercises.

It has to be considered that the primary purpose of many monitoring and SA tools is to operate in production environments. Thus, contemplating their use in CSX environments is bound to have some shortcomings that must be addressed and overcome. We applied prior research and expertise with organisational monitoring and situation awareness systems to design and implement exercise-specific systems. These systems were presented in Publications V, VI, and XI, and will be described in detail in section 4.

Although the exercises described in this thesis depict a realistic scenario, there are numerous nuances that introduce several influential differences due to the nature of the exercises and the requirement to provide meaningful feedback to participants in a short timeframe. For example, prior to the exercise the deployed target systems are just idling, hindering any kind of experimentation and baselining required to tune detection methods. Contrariwise, due to the limited timeframe of the exercise, when participants access the systems and the red team starts their attack campaigns, the rate of incoming security events far exceeds typical production environments. Thus, the actual testing and verification of the systems can only happen effectively during the short timespan when the exercise takes place.

3.4 Comparison with related work

This section compares the work presented in this chapter with the related work described in section 2.1. As stated above, one of the primary goals of the research behind chapter was to bridge the gap between published theoretical suggestions and practical implementations. This was achieved by offering practical recommendations and example organisational implementation for a metrics and SA system.

While a number of security metrics papers were discussed in the related work section 2.1.1, due to differences in how the topic is addressed, a direct (i.e., side-by-side) comparison is not applicable for any of the publications. However, the following publications discussed below took a similar approach or proposed a somewhat similar solution for bridging this gap.

Similarly to our work, Jaquith provided recommendations for reporting and visualising metrics while also identifying the gap between managerial metrics requirements and technical metrics typically provided by technical security professionals [69]. An earlier work [44] by Geer et al. noted in surprise that in the field of business metrics, these problems have already largely been solved, while information security specialists seem to be still struggling.

The CIS Critical Security Controls have been actively developed for more than a decade and currently describe 20 groups of comprehensive security controls. Furthermore, like in our paper, the CIS publishers have just recently started to publish more detailed implementation and practical guidance for specific types of systems (e.g., Windows, network devices, mobile devices, and IoT) [24].

In [8], Arendt et al. discussed the requirements of an SA system based on the example of their Ocelot tool. The authors developed Ocelot with a clear user-centric emphasis, meaning that the operator should have everything at hand to explore and visualise data, drill down for investigating raw data, and act when deemed necessary. Although it is a customised SA solution, Ocelot had several critical limitations—the presented version of Ocelot only supported six predefined defensive strategies the user can execute on a selection of hosts. It remains unclear whether any additional progress has been made to continue the development of Ocelot.

Alternatively, a recent paper [78] by Kohlrausch and Brin discusses a similar security metrics problem, but proposes a somewhat different solution (i.e., the ARIMA method for time-series analysis) to the issue. In contrast, Publication I suggests the use of frequent itemset mining and data clustering algorithms for various anomaly detection tasks based on NetFlow and IDS alert data. Similarities include identifying the need for novel advanced algorithms and approaches to detect anomalies in gathered security metrics.

4 Situation awareness systems for cyber security exercises

This chapter discusses a multi-faceted endeavour to enhance the learning process by providing CSX participants improved training feedback, verify the performance and applicability of standard monitoring and SA systems during CSXs, and advance the newly developed CSX-specific SA systems (i.e., Frankenstack and Availability Scoring systems). Although we implemented the particular SA systems for use during the XS and LS exercises, the architecture of these systems still provides a clear point of reference for other researchers and cyber defenders in need of building such monitoring frameworks in their specific exercise or production environments.

This chapter addresses research questions RQ1.3 and RQ1.4. The research was originally published in Publications V, VI, and XI. Thesis contributions 5–7 are described.

4.1 Author's involvement with CSXs

The author of the thesis has been in the organising team of the LS and XS exercises (described in section 2.1.3) since 2014 and has been directly involved with providing SA and training feedback to CSX participants. The author has been involved with the XS exercise as a YT member during its entire existence, however, LS exercise was already a well-established exercise when the author joined the NATO CCD COE and the exercise green team.

Note, that the author's primary aim has been to advance technical research, therefore the discussion of participant learning experience serves as side topic that provides the reasoning behind some of the technological decisions. Although the author of this thesis was involved with the preparation of questionnaires for the LS exercise, none of the author's contributions directly handle the assessment of the participant feedback collected from the LS exercise. The author was directly involved with analysing the feedback for the XS exercise, however, the questionnaires and interviews conducted during the LS exercise have been assessed by Maennel in [97].

4.2 Enhancing operator training quality during CSXs

The exercise infrastructure and the provided toolset are critical for efficient learning, however, they do not make the exercise a success by default. In many cases the human factors, such as how the training audience perceives the environment and uses the tools, have a significant impact.

A key factor in improving the training quality and improving participants' learning experience is asking direct feedback about the CSX toolset that was provided during the exercise. One fundamental part of the training experience assessment was to observe the behaviour of the training audience and their interaction with Frankenstack and Availability Scoring system during respective exercises to gain further awareness into their training experience. Since 2016, we have carried out interviews and questionnaires among exercise participants during the CSXs about the monitoring and CSX-specific SA tools that we have provided to the exercise target audience. Based on the feedback, we have continued the development and attempted to iteratively improve the Frankenstack framework and the Availability Scoring system.

Therefore, one objective of this work was to assess and improve the quality of training and learning experience during CSXs. One of the primary concerns that exercise participants had expressed during interviews was the amount of time it took to receive feedback about the current situation and ongoing activities. For example, it was previously unknown for the participant, whether a recent action affected the current security pos-

ture of the adversary or not. While this indeed remains unknown in the real-world use cases, it hindered the learning pace and experience of participants.

Another issue was the lack of consistency when human-generated feedback or reports were provided by multiple people on short notice to the participants during the exercise. Manual analysis performed by different human security operators in a constrained time window may result in different interpretations for the same set of events. Thus, another objective was to provide uniform feedback that would be clear and similarly understandable for all CSX teams and participants.

4.2.1 Crossed Swords

The first iterations of the XS exercise (since 2014) revealed several issues with RT learning experience. Traditionally, YT feedback sessions took place at the end of each exercise day, however, this was not best suited to the fast-paced and technical nature of the exercise. The time it took from RT member triggering an alert to the YT briefing session at the end of the day was just too long. Most likely the RT member had already forgot the exact details of the conducted attack.

Furthermore, due to limited time, the briefing addressed only the most noteworthy observations from that day, however, the RT needs explicit and immediate feedback about detected activity to learn from their mistakes as they happen. The feedback observations need to be well described and detailed, so that the RT can quickly understand why and how a specific attack was detected, and then try again with an improved approach.

Finally, in the first few XS iterations most of the data analysis in YT was done manually by several different operators. This proved to be slow and sometimes inconsistent in which attacks were followed up on. Therefore, the slowest and most inconsistent element (i.e., the human operator) in the feedback loop needed to be eliminated and replaced with automation as much as possible.

To solve this problem, for the 2017 iteration of XS we re-used the same ensemble of open-source tools that we had already previously used in production environments (described in section 3.1) as well prior XS iterations. However, we included event correlation, a novel query automation tool, and a newly developed visualisation solution to automate the feedback process. The resulting open-source framework was called Frankenstack and is published on GitHub [117].

Frankenstack is an SA system designed to increase the training experience and learning benefit of the cyber-RT participants during CSXs. Over the years the technical feedback framework has developed into an integral part of the XS exercise. This work directly relates to thesis contributions 5 and 6, described in Publications V and XI. The technical description of Frankenstack follows in section 4.4.

On occasion, we discovered that the feedback we provided to the RT revealed too much information too soon. There is a fine balance between enhancing the learning experience and ruining the game by spoiling the fun of exploratory discovery. We remain committed to perfecting and maintaining this balance for future iterations.

4.2.2 Locked Shields

Prior to the 2015 Locked Shields event, detailed availability scoring status information was not available to BTs. Although the predecessor to the presented Availability Scoring system already existed along with the overall exercise scoring capability, there were no live availability dashboards or direct feedback to participants during the game. As a result, BTs had no way of knowing what was causing them to lose points for availability. They could see the overall score summary, but no detailed breakdown of individual services.

The development of the Availability Scoring system that is presented in this thesis started in February 2014. After the newly developed Availability Scoring system had been successfully implemented and verified at the 2014 LS exercise, the organising team was convinced that the new solution is stable enough to provide live availability feeds and dashboards to all participants in the upcoming years.

In short, Availability Scoring framework is an SA system designed to measure BT performance during defence-oriented CSXs by adopting several common SLA metrics for IT services. The framework has become an integral part of the overall LS exercise SA system. This work directly relates to thesis contributions 5 and 7, originally described in Publication VI. The technical description of Availability Scoring system follows in section 4.5.

4.3 CSX network layout

The XS and LS exercises are mimicking realistic computer networks with a variety of different hosts (e.g., network devices, servers, workstations, laptops, and other specialised equipment). See Figure 1 for a network map that was prepared for LS 2015. Since a considerable amount of exercise environment is reused from year to year, the use of an outdated image is intentional to not reveal anything sensitive. When examining the figure, it is important to note that every BT gets its own full set of hosts that are indicated in the network map—hence the *xx*, *X*, and *XX* notations within hostnames, VLAN IDs, and IP addresses. Furthermore, to scale up the in-game *office network segments* into more realistic organisational networks, each workstation icon actually represents five sequential workstation nodes deployed into the corresponding network.

Although the training audience for XS is the RT, the main elements of the BT infrastructure remain largely similar. Therefore, the network illustrated in Figure 1 can also be effectively used to describe the XS exercise. However, the primary difference is that during the XS, RT starts out with only minimal information about the target environment and has to discover the entire network themselves.

4.4 Frankenstack

The Frankenstack framework features a near real-time feedback loop for the RT participants: any RT action that is discovered on the game network and hosts is analysed and reported back to the RT dashboard as an indicator of compromise. This allows RT members to immediately try again to improve their methods to avoid at least basic detection methods. Furthermore, the framework provides SA about RT progress to the exercise leadership (i.e., the white team) allowing them to precisely control the pace of scenario advancement. See Figure 2 for an high-level overview of the Frankenstack feedback cycle between various in-game teams.

Although Frankenstack was developed as an SA tool for the XS exercise, the open-source modules are publicly available on GitHub and can be materialised by others running a similar environment. Frankenstack makes use of several input data sources: full network traffic mirror, numeric metrics, and event logs. These sources are largely overlapping with the data sources that are used by the production security metrics system described in chapter 3 and Publication I.

4.4.1 Input data sources

The network traffic capture was provided as ERSPAN (Encapsulated Remote Switched Port ANalyser) mirror sessions from the switches in the virtualised game network environment. This meant that Frankenstack's IDS component (i.e., Suricata [128]) also had visibil-

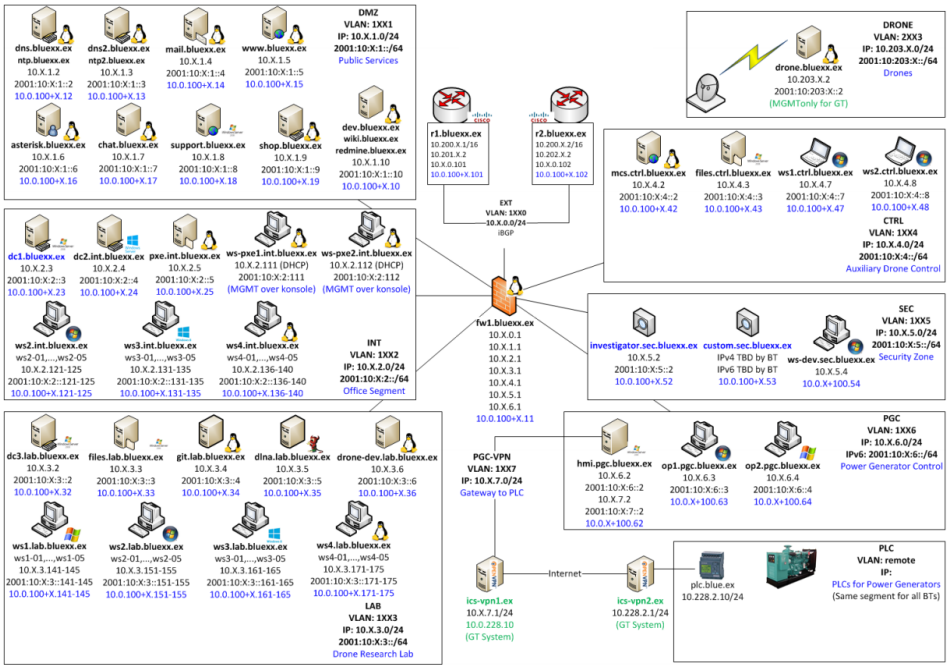

Figure 1: An example CSX network layout.

ity within internal networks not just traffic that was traversing between network routers and perimeter firewalls. Furthermore, all traffic was captured and indexed using the full packet capture and analysis tool called Moloch[2] [10]. Note, that NetFlow data from network routers was available but was used only during the first iteration of Frankenstack. As was discussed in section 3.1.2, IDS/IPS solutions can output information that is very similar to NetFlow. There is a computational overhead compared to NetFlow, however, within XS the amount of network traffic is relatively low, so we finally opted for Suricata to collect network flow data. Another upside was the improved network visibility from the virtual game network switches compared to network perimeter routers.

Moreover, we configured the in-game target systems to collect numerical time-series metrics (e.g., system load, CPU, memory usage, and network interface statistics) using the Telegraf tool [65] that is part of the TICK stack [66] developed by InfluxData. We collected event logs from all in-game systems wherever possible (e.g., Event Logs from Windows, Apache and nginx web server logs, syslog from Linux). We extended Windows Event logging with additional rules [53, 156] for Microsoft Sysinternals Sysmon [104]. Instead of implementing AppArmor and SELinux for enhanced Linux auditing, we opted for using a small library called Snoopy Logger [70]. This was because configuring AppArmor or SELinux typically incorporates increasing the base level of security on the system, however, we did not want to interfere or impair any of the pre-configured vulnerabilities that were planted on the target systems. According to [70], Snoopy Logger is not designed as a reliable security auditing tool for production systems, however, it fits the exercise scenario where the YT instrumentation should not interfere with the in-game systems.

The host instrumentations mentioned above are very difficult to implement in a standard defence-oriented CSX with BTs as the training audience: if the objective is to give

[2]Moloch was renamed to Arkime in the second half of 2020.

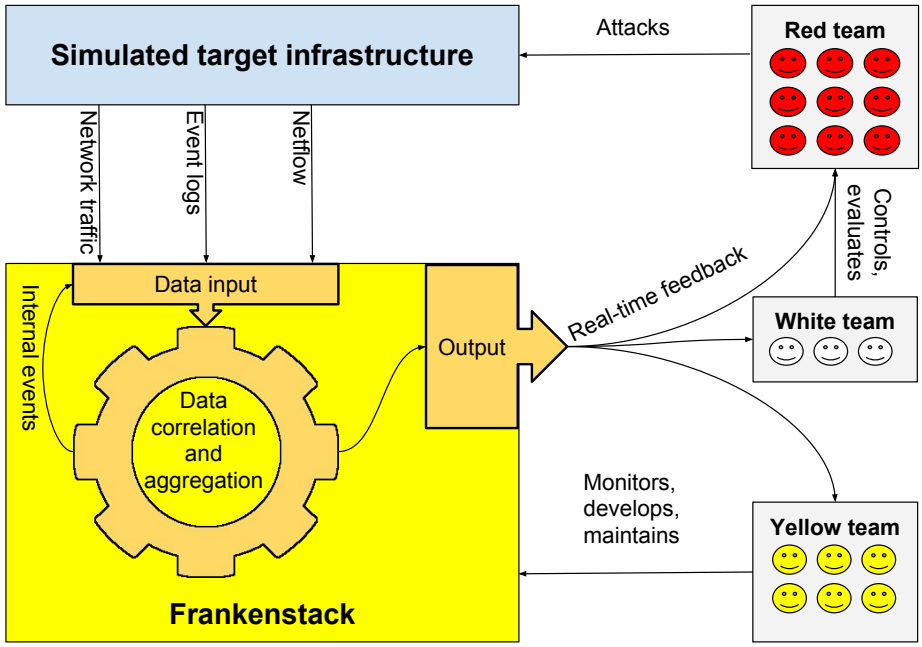

Figure 2: High-level overview of Frankenstack. [82]

BTs full control of their in-game network infrastructure, then they also have full power to disable or reconfigure these tools for any reason. However, as the XS training audience is the RT, the YT could maintain supervisory control of all BT systems and ensure a constant stream of incoming monitoring data.

4.4.2 Data processing components

Post-mortem analysis of available datasets is an irreplaceable method during incident analysis. Although it often reveals valuable insight about cyberattacks, it requires a substantial amount of time and manual work. Unfortunately, this conflicts with the short time span of the exercise and is not a viable method to keep track of the RT. Furthermore, during such incident investigations cyber defenders often write ad-hoc search queries (e.g., in Moloch or Kibana), making analysis results challenging to reproduce later. Thus, Frankenstack comprises Otta [112], a novel query documentation and automation system for executing user-defined queries on large datasets at predetermined intervals. As such, Otta transforms aggregated search query results into time-series security metrics, enables operators to graph trends, define alerts, and detect anomalies for user-defined queries. This directly reduces time spent on analysis, automates detection, and provides reproducible queries to retrieve relevant results.

Frankenstack processes data from the aforementioned input sources and applies event normalisation, enrichment, and correlation for combining various information sources into a single stream of meaningful events. The system employs pre- and post-processing scripts for several event normalisation and enrichment tasks, however, the main data conversion and correlation pipeline is handled by SEC [171, 172]. This exercise-specific configuration and ruleset called frankenSEC has been published in GitHub [113].

Although Frankenstack itself has been kept open source, we have not excluded cooper-

ation with existing analytical platforms, SIEM systems, or commercial vendor appliances. Over the years many security vendors (e.g. Cymmetria [29], Greycortex [47], and Stamus Networks [154]) have joined the exercise YT to test their products in a unique live-fire environment [116]. We do not treat any security product as a all-in-one solution, but just as another data source that produces a separate feed back to the data pipeline.

4.4.3 Visualisation components

During the XS exercise numerous large screens are installed in the training room directed at the RT. The purpose of those screens is to provide visual feedback from various tools without necessarily taking up any of the valuable screen real estate from the RT members. Furthermore, it is not possible to give RT members direct access to some of the SA tools (e.g., Moloch and Kibana) used for creating dashboards, because being able to directly query those tools would expose too much information that the RT is tasked to discover by themselves. The machines displaying the dashboards were directly connected to the YT network segment which was not available for the RT members. Of course, by having physical access to the machines that were connected to the large screens in the same room, it was possible for them to circumvent this network separation.

Frankenstack comprises a set of open-source tools for visualising log data, time-series metrics, and alerts. There are slight differences in handling various types of alerts: for example, alerts for CPU and memory usage trigger and recover automatically based on a predefined set of thresholds, however, security events (e.g., IDS/IPS alerts) are only triggered based on some detection condition but lack the concept of a recovery threshold. Thus, such security alerts will never receive a recovery event, leading to an overflow of information on the feedback dashboard.

Correlation and deduplication of recurring events is crucial for creating usable visualisations. Due to the volatile nature of CSXs and simulated network traffic generation can at times potentially overflow visualisation tools with too much information for users to follow. For example, a network scan using the `nmap` tool can trigger a large amount of security events over a period of time. While event correlation can collect and combine those events, it does not make sense to wait indefinitely before emitting the alert to the dashboard. The aim is to notify the RT of their activity in near real-time, therefore the length of the correlation window has to be kept relatively short. With effective deduplication functionality sending the same alert multiple times does not cause any issues.

Alerta [149] is used as the primary feedback dashboard to present detected RT activity back to the RT. The RT feedback cycle is completed by emitting the transformed event from the correlation engine to the RT dashboard. Each RT member has access to the Alerta API and web interface to create personal filtering rules for limiting the displayed information only to what is relevant in the current attack campaign. To address the issue of security alerts not recovering and leading to an overabundance of events on the dashboard, we set a timeout to automatically archive events that had no correlated activity within the last 15 minutes. Although some event thresholding and deduplication was implemented within the frankenSEC ruleset, Alerta also featured deduplication based on a set of fields within the event. Based on the 2017 XS statistics, the RT dashboard displayed 691 unique events out of 28660 (i.e., 97.59% of all events were de-duplicated).

Kibana [37] and Grafana [46] are used to for presenting more overarching analytical dashboards that provided insight over the entire duration of the exercise, not just the recent events view available in Alerta. For instance, summary of detected RT attack types or statistics of IP addresses that have been generating the most alerts, etc. In addition to RT members, the large-screen dashboards were often observed by WT members who

were interested in the progress of the exercise and overall performance of the RT.

Attack maps are often used to provide a condensed way of visualising events. Unfortunately, they are typically not usable during CSXs because they rely on geographical data which is largely fictional in exercise environments. To address this problem, the author proposed the concept of **E**vent **V**isualization **E**nvironment (EVE), a novel web-based tool for visualising detected attacks in relation to CSX-specific game infrastructure [102]. Source code of EVE has been made publicly available in GitHub [111]. Although, the development of EVE was largely carried out by another YT member, the author of this thesis was the principal user and responsible for providing the data feeds to EVE.

EVE displays attacks carried out by the RT with a customisable game network map as the application background. In a short time window, EVE is able to correlate multiple events that have the same source and destination addresses into a single attack. Attacks are displayed as arrows connecting the source and target hosts on the network map. Furthermore, detailed attack information and a list of previous attacks is displayed next to the network map. The use of the original exercise network map makes EVE a very intuitive tool that enables both participants and observers to easily comprehend CSX events on a high-level. During the first experiment, the EVE tool was not shown to RT members, as it revealed too much of the exercise network map that they had to discover on their own. However, EVE featured a dedicated replay mode to display all the attacks condensed into a predefined time period (e.g., 15 minutes). The entire exercise dataset was replayed to the RT after the exercise. This allowed RT participants to obtain an overview of their entire attack campaign—revealing the most critical mistakes, such as the surge of alerts generated by several network attacks and periodic beaconing during otherwise quiet night periods.

4.4.4 Frankenstack developments since 2017

This section describes the more recent R&D efforts of Frankenstack following the initial paper publication in 2017. This subsection summarises the recent Publication XI detailing the revised architecture, event normalisation process, data enrichment techniques as well as updated event processing pipeline.

Distributed event streaming The initial design of the event processing pipeline relied primarily on syslog-ng [127] to collect, store and forward events. This works well for systems and applications that are set up and configured beforehand, so that proper syslog rules can be created. However, during the annual hackathons that preceded the XS event, contributing participants within the YT would often integrate their own tools and scripts that also needed to analyse the same incoming data feeds or a subset of past events to provide an alternative assessment in addition to the main data processing pipeline. Having events stored as files on the central log collection server is not ideal for this purpose. Alternatively, Elasticsearch can be used to query historic data, but there is no good way to continuously stream all incoming events in real time.

We analysed various distributed message streaming tools and opted for using Apache Kafka [5] as a central collection point for all emitted messages. Kafka fulfils the requirement for a multi-producer and multi-consumer event feeds. Since 2018, there is a general decision within the Frankenstack framework that all events should be produced to Kafka and for further analysis all tools should stream the corresponding topics from Kafka.

Determining the attacker One of the primary problems the YT faced was automatically determining the directionality of the attacks, i.e., identifying the victim and the attacker in a particular cyberattack. For a moment we planned to rely on IDS alerts from Suricata

which always contain *source* and *target* fields. However, this approach did not turn out quite as planned—the *source* and *target* fields in IDS rules just signify the direction of traffic for which the detection match conditions are written for. This meant that whenever the rule writers had written a rule that detects a response of an attack (e.g., sensitive data leaving the victim node), we would have erroneously classified the victim as the attacker. With this approach, we were only able to connect the relevant nodes, but lacked the directionality between them. Unfortunately, there was no other reliable metadata within the IDS rules that could have revealed the directionality of the attack.

Fortunately, Eric Leblond, one of the core Suricata developers, had been part of the XS YT since 2016. He raised this issue with the Emerging Threats rule writers and for the next XS iteration there was already a preliminary solution available, which has now been adopted into the mainstream version of the rules [138]. Emerging Threats rules now contain a metadata field called *attack_target*[3], which reveals the victim-side counterpart of the attack. Although not specifically published, this development effort has been presented at several security conferences, e.g., Hack.Lu [89] and SuriCon [90].

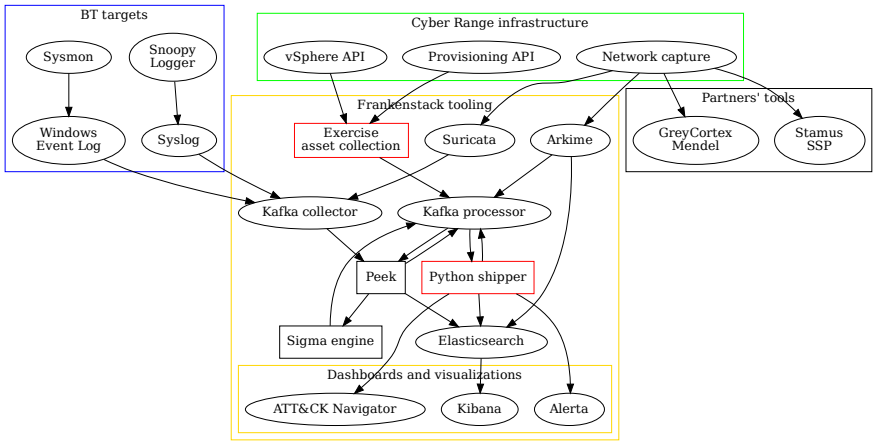

Figure 3: Yellow team's updated technological perspective from XS 2020. Green area denotes data sources originating from the exercise backend infrastructure, i.e., Green team assets. Blue area signifies YT tools and data sources deployed on BT networks and hosts. Black area indicates industry partner tools. Yellow area describes Frankenstack components and the data flow between them. Rectangle-shaped nodes represent novel tools developed for Frankenstack. Red rectangles signify tools developed by the author of the thesis.

Improving event processing Frankenstack employed SEC as the primary normalisation, enrichment, and correlation tool until 2019. As discussed above, all relevant event streams within Frankenstack were already configured to output structured JSON events at the source or transformed into JSON already at the pre-processing phase. Unfortunately, the JSON events emitted by distinct sources featured different structure which had to be individually handled. Bearing in mind that SEC and its rule language was initially designed for

[3]At the time of writing in September 2020, there are nearly 15,000 rules which contain the *attack_target* metadata keyword.

complex event correlation tasks on unstructured messages, it soon became cumbersome to handle complex nested data structures within the SEC rule language. For example, any changes in input JSON key values resulted in the need to edit numerous textual rules. Therefore, instead of using the conventional SEC rule syntax, the author had to write Perl code into most frankenSEC rules. Although it was possible to accomplish what we required using Perl functions, the rule writing and management soon became infeasible to maintain.

Alternatively, processing complex nested data structures within a fully-fledged programming language seemed more approachable. In hindsight, it seems that our primary issue was that we attempted to correlate events too soon in the data processing pipeline—SEC is an event correlation tool, not a programming environment or a data normaliser. Unfortunately, in our ruleset we tried to accommodate many of the data processing and transformation tasks which should have been completed prior to pushing events into SEC. This hindered the rule-writing process and the lack of proper post-processing meant that even minor changes in the input event structure resulted in the need to rewrite a large portion of the rules.

As a replacement we developed a novel data normalisation and enrichment tool called Peek [118]. Peek enriches each atomic message with metadata to determine the event sender, event directionality (i.e., inbound, outbound, lateral, or local), and, if applicable, the attack source and target. This automated enrichment is facilitated by the *exercise asset collection* tool developed by the thesis author. Furthermore, asset information collection enables us to for map hostnames and IP addresses from alerts to known assets which in turn simplifies threat hunting.

The frankenSEC ruleset was largely replaced by the Sigma ruleset [145]. An example Sigma rule is provided in Listing 1. Similarly to Peek, our Sigma match engine was also written in Golang and is available in GitHub [79]. A comprehensive description of the Sigma rule engine is available in a recent whitepaper [81] by Kont and Pihelgas.

Listing 1: Sigma rule to detect base64 encoded PowerShell scripts. [136]

```
title: Encoded ScriptBlock Command Invocation
author: Mauno Pihelgas
description: Detects suspicious PowerShell invocation command parameters
detection:
  condition: selection
  selection:
    winlog.event_data.ScriptBlockText:
    - ' -FromBase64String'
falsepositives:
  - Penetration tests
  - Very special PowerShell scripts
fields:
  - winlog.event_data.ScriptBlockText
id: 697e4279-4b0d-4b14-b233-9596bc1cacda
level: high
logsource:
  product: windows
  service: powershell
status: experimental
tags:
  - attack.execution
  - attack.defense-evasion
  - attack.t1059.001
```

Moreover, the author of the thesis redeveloped the part of frankenSEC that interfaced with the Alerta dashboard as a comprehensive *Python event shipper* script which ensured that all events sent to Alerta conform to a uniform structure and are mapped to the MITRE ATT&CK adversary tactics and techniques knowledge base. This emitter script also imple-

mented a simple baselining functionality to identify security events that occur under normal system use (e.g., execution of scheduled tasks, system updates, etc.). Later, during the exercise, such benign events were suppressed and not displayed to the RT dashboard. Figure 3 illustrates Frankenstack's technological outline from the latest iteration of XS in December 2020.

4.5 CSX Availability Scoring system

To measure the performance of the LS exercise training audience, the organisers must establish proper situation awareness over more than 20 BTs involving hundreds of individual players. Assessing and scoring the performance of BTs is essential in providing them meaningful feedback. Although one of the goals of LS is to improve learning and cooperation between nations, there is still a competitive element in the game that compels all BTs to give their best during the CSX. The primary task of the Availability Scoring system is to measure the availability and functionality of individual services provided by BT systems to provide availability scores and detailed technical feedback to all exercise participants.

4.5.1 Basics of availability scoring

This section describes the basic design and operation of the availability scoring system. Note, that Frankenstack-like host instrumentations are exceedingly difficult to implement in a standard defence-oriented CSX with BTs as the training audience: if the aim is to give BTs full control of their gamenet infrastructure, then they also have full volition to disable or reconfigure these tools for any reason.

The central component of the Availability Scoring framework is Nagios Core [110]. Nagios Core is a stable and mature open-source monitoring software that has been developed since 1999[4]. It features a modular design that supports the use of both active and passive service checks. Although the research considered several other standard monitoring tools (e.g., Centreon, Shinken, Opsview, Nagios XI, op5, and Zabbix), the author concluded that Nagios provides most of the basic monitoring functionality that we required from the scoring system. Additional issue with some monitoring suites was that they included too much unnecessary functionality for our environment, for example, incident tracking and helpdesk functions are not needed in our case.

Listing 2: Service check log format and example output.

```
# Defined in Nagios configuration as:
# $HOSTNAME$|$SERVICEDESC$|$SERVICESTATE$|$SERVICEOUTPUT$|$LASTSERVICECHECK$|$TIMET$

"mail.blue05.ex"|"http"|"OK"|"HTTP OK: HTTP/1.1 200 OK - 322 bytes in 0.005 second
response time"|"1524721248"|"1524721248"

"hmi.pgc.blue13.ex"|"http.ipv6"|"OK"|"HTTP OK: HTTP/1.1 200 OK - 954 bytes in 0.014
second response time"|"1524721248"|"1524721249"

"wiki.blue13.ex"|"https"|"CRITICAL"|"HTTP CRITICAL: HTTP/1.1 200 OK - pattern not
found - 53048 bytes in 0.649 second response time"|"1524721248"|"1524721249"

"ws4-02.int.blue15.ex"|"ssh"|"CRITICAL"|"No route to host"|"1524721244"|"1524721249"
```

The scoring system checks and reports the state (OK, Warning, Critical, or Unknown) of the services that the BTs have to keep functional during the two-day exercise. Any interruption (i.e., any non-OK state) in the service will cause loss of uptime and, more

[4]The project initially used the name NetSaint, but was renamed to Nagios in 2002 due to a dispute over the name *Saint*. [106]

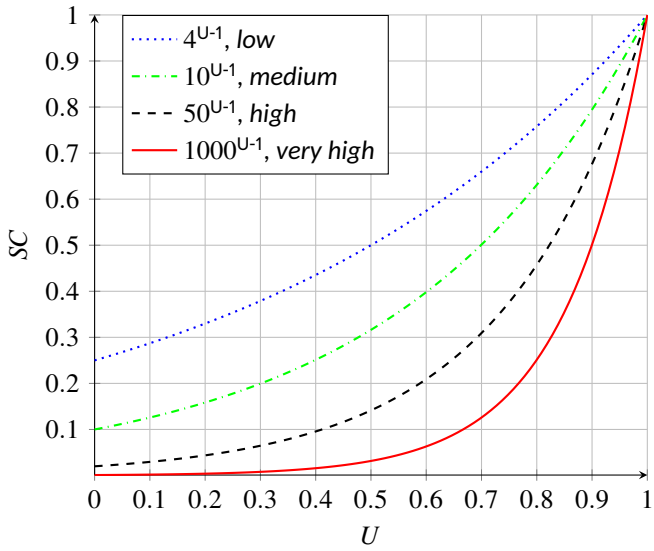

Figure 4: Score coefficient (SC) with regards to uptime (U) and four example service weight values (W).

importantly, valuable score points. Similarly to Frankenstack, the output (see an example in Listing 2) produced by these service checks is submitted to Apache Kafka for further processing and uptime calculation. The custom output log format involving the vertical bar separator was adopted from the previous scoring system. However, it fitted the new solution ideally since the vertical bar symbol is treated as a special character in Nagios to separate the emitted message from the check script performance metadata and thus cannot accidentally appear in the output message.

4.5.2 Availability calculation

During the two days of LS exercise gameplay BTs can gather up to 20,000 availability points (i.e., approximately $\frac{1}{3}$ of total positive score). Individual scored services are distributed into various sub-groups based on their importance. Typically four or five importance levels (e.g., from *low* to *very high*) have been defined during each LS event, albeit, specific weight values (W) have been tuned from year to year. Note, that $\mathtt{W} > 1$ for each weight value. Any loss of uptime (U) will affect the score coefficient (SC) exponentially:

$$\mathtt{SC} = \mathtt{W}^{\mathtt{U}-1}, \quad 0 \leq \mathtt{U} \leq 1$$

Evidently, higher service weight values (W) result in steeper score penalty for any lost availability (see Figure 4). Actual scored points are calculated according to $\mathtt{P} \times \mathtt{SC}$, where P is the point budget for that particular service. For example, if the point budget (P) for a service is 800 points and it has been assigned the highest service importance (represented by the solid line in Figure 4) it will only yield circa 400 score points if its uptime is 0.9 (i.e., 90%). This is due to the score coefficient (SC) quickly declining to approximately 0.5 if the uptime (U) drops to 90%.

This harsh score penalty is derived from real-life approximation—many information technology service providers are commonly expected to maintain production systems' availability value in the order of 99.9% or in some cases even higher. For instance, this

means that a corresponding service can be unusable for just about 1.44 minutes per day or 10 minutes per week. Of course, exact requirements are generally settled in written service level agreements.

4.5.3 CSX-specific challenges

In a typical organisational monitoring scenario, the monitored system is either under the control of the monitoring operator or the monitoring is provided as a consensual service to the system operator. In other terms, the target system operator has no reasonable motivation to deceive the monitoring system. Furthermore, within organisations there are often some best practices and rules that are followed by all system operators. However, this is not the case in LS as all teams can more or less choose their own defensive strategy. Furthermore, as there is a competitive element within the game, so it is in the interest of the BTs to maximise their score. Although tricking the scoring system is forbidden according to the exercise rules, over the years some BTs have attempted to use questionable means to block the RT from attacking and delude the scoring system into thinking the service is up and running when it is not. If RT attacks are not averted by good defensive skills, then BTs are actually diminishing their own potential learning experience.

Large number of teams and replicated networks First and foremost, the exercise implementation differs from real-world environment as each BT (totalling over 20 since 2018) is given an identical network of systems to defend. Although setting up the scoring system for cloned environments seems like a simplifying aspect at first, it has to be considered that from the moment teams are allowed to access their machines each team will pick a slightly different defensive strategy and system configuration. Therefore, the scoring system has to be capable of adjusting to changing configurations on the fly. For example, many teams often change the IP addresses of their hosts or modify the network structure, so the Availability Scoring system must keep track of those changes. Team-specific DNS server has proved to be the most trustworthy source of information for keeping track of such modifications.

Moreover, BTs often upgrade their software packages to newer versions and configurations which can naturally affect how the service behaves, sometimes erroneously breaking the scoring check. Such rare cases require a fix to be developed and applied as quickly as possible. However, it may be that only a subset of the teams decides to upgrade to a newer version, which means the scoring check has to be flexible in choosing the correct approach. As a result, by the end of the exercise most BT systems have become slightly different and potentially using a distinct configuration.

Active service checks Although using host-based agents and submitting passive check results to a central collector is the preferred approach in many monitoring solutions[5], the Availability Scoring system makes extensive use of active checks (i.e., checks initiated by the server) to verify the availability and proper functionality of remote targets. Active checks are more reliable in adverse environments, such as competitive CSXs, where BTs have full control of their infrastructure and could theoretically take advantage of passive checks initiated by the monitored host itself. As mentioned, active checks are initiated by the scoring server and BTs have no control over what exactly is being checked or when the checks are executed.

[5] In Nagios context, passive check means that the host-based agent initiates the check and communicates results back to the central monitoring server

Listing 3: Typical unmodified Nagios checks examined from a BT web server access log.

```
10.0.228.111 - - [23/April/2018:06:39:58 +0000] "GET / HTTP/1.1" 200 437 "-"
"check_http/v2.2.1 (nagios-plugins 2.2.1)"

10.0.228.111 - - [23/April/2018:06:40:58 +0000] "GET / HTTP/1.1" 200 437 "-"
"check_http/v2.2.1 (nagios-plugins 2.2.1)"

10.0.228.111 - - [23/April/2018:06:41:58 +0000] "GET / HTTP/1.1" 200 437 "-"
"check_http/v2.2.1 (nagios-plugins 2.2.1)"

10.0.228.111 - - [23/April/2018:06:42:58 +0000] "GET / HTTP/1.1" 200 437 "-"
"check_http/v2.2.1 (nagios-plugins 2.2.1)"
```

Listing 4: Web server access log entries created by a modified check script and a varying check interval.

```
10.0.228.111 - - [23/April/2018:06:45:43 +0000] "GET / HTTP/1.1" 200 437 "-"
"Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0"

10.0.228.111 - - [23/April/2018:06:46:25 +0000] "GET / HTTP/1.1" 200 437 "-"
"Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0"
```

Hiding identification strings Production monitoring systems typically do not need hide their identity, on the contrary, many such tools contain identification strings that enable operators analysing the logs or network traffic to easily distinguish monitoring checks from other traffic. For instance, many web monitoring plug-ins specify a user-agent string that specifies the name and version of the script: see Listing 3 for an example. Similar situation applies for checking SSH and e-mail services, where the client commonly identifies itself to the server upon setting up the connection.

However, the Availability Scoring is meant to monitor the BT services in a way that is not clearly recognisable as the scoring check. This is meant to avoid BTs whitelisting the scoring checks while blocking all other traffic, *inter alia*, the RT attack traffic. Therefore, the author had to edit and recompile several scoring checks to better mimic end-user tools deployed on regular workstations. See Listing 4 for an example of a modified web monitoring script that was using the user-agent string of the Mozilla Firefox browser. Note, that this specific version of the browser was in fact present on the Linux-based BT workstations that were deployed in the game and used by the User Simulation subteam.

Randomising check intervals Furthermore, having a fixed checking interval is a reasonable approach in standard monitoring situations. Thus, default configuration for Nagios schedules service checks at 60-second intervals, however, more adept BT operators observing event logs or network traffic are likely able to notice the heartbeat-like sequence of incoming checks (observe the timestamps in Listing 3). This enabled BTs to predict to an exact second when the scoring system will execute the next availability check. Therefore, the author introduced a periodic change of the check interval at which individual scoring checks are initiated. The exact interval was randomly generated in the range of 40 and 55 seconds (observe the timestamps of the modified check interval in Listing 4). Shorter intervals also provide more accurate data about the status of various services in any given time.

Randomising IP addresses Even if BTs can identify and whitelist some of the Availability Scoring IP addresses, the system is configured to periodically acquire new public IP addresses that are used to communicate with BT services from the public networks. Note, that RT attacks also originate from the same public network ranges. To safeguard against IP conflicts when assigning new addresses, arping (for IPv4) and ndisc6 utilities (for IPv6) were used to verify that the newly generated IP address is in fact vacant. Within internal BT network segments, the Availability Scoring nodes have fixed IP addresses. In early years, the system also generated new IP addresses within internal BT networks, however, it was deemed unrealistic to have a rogue host in an internal segment that keeps changing its IP address.

Countering removed service functionality Even when taking all aspects above into account, there are still several ways the Availability Scoring can be deceived. For example, BTs have set up fake services (e.g., portspoof [35] or short-lived Docker containers [33]) that attempt to mimic the monitored service, but in fact do not offer the entire required functionality. To counter this, scoring checks executed within the Availability Scoring system cannot check only the basic connectivity (e.g., ICMP ping or establishing a TCP connection), but rather need to implement more advanced functionality checks that mimic the usage patterns of end-users. This is a continuous process that has been addressed and improved by the author over the years.

Another method has been to employ versatile network redirections (e.g., using DNS or ARP spoofing) so that instead of the intended target the scoring checks end up at some other host. Such activity can also be used to reduce the number of hosts the BT has to maintain and just redirect all identical services at a single host. For example, multiple load balanced NTP servers have been reduced to just one single host providing the service. The rationale behind such activity is often that the BT is attempting to reduce the potential attack surface for the RT.

Moreover, web service functionality is often intentionally crippled to stop RT from attacking and gaining foothold via insecure websites. To counter this approach, we have used the open-source Selenium WebDriver [153] which is a comprehensive testing framework for web applications. Selenium enables writing test cases for automated web browsing to check the proper functionality of various websites. For instance, mimicking a user visiting the page of an online store, logging in, selecting a few items, and adding them to the shopping cart, then finalising the purchase, and verifying that the transaction was actually saved under previous orders. Such test cases have their pros and cons: they provide an extensive toolset for writing checks to verify that required functionality is present, but due to the complexity of the test cases they are problematic to debug if the test case erroneously flags the site as non-functional.

4.5.4 Community contributions

In the early spring of 2014, after starting to implement the Availability Scoring system based on the latest stable version of Nagios (v4.0.3 at the time), the author unfortunately discovered some bugs in the Nagios Core software when the overall timeout for all checks was set to 20 seconds, which is low for any normal monitoring environment, but this low threshold actually fits well with simulating regular end-users, who most likely do not have the patience to wait more than 20 seconds for a single request to complete. The problems surfaced when the timeout of 20 seconds was reached. First, the status of the timed-out service check was considered to be OK, not CRITICAL, as it should have been. Second, there was an extra newline symbol present on one of the built-in timeout messages that

Nagios writes into its log file. This caused problems with the log parser that published messages to Apache Kafka for the uptime calculation of BT services. The author was able to identify the bugs from the source code and recompile the Nagios Core software for personal use, however, the author also reported the bugs to Nagios developers along with a suggested fix for the problems. The first bugfix (*Fixed bug #600*) was released in Nagios Core v4.0.6, and the second (*Fixed bug #608*) in v4.0.7 [109].

4.5.5 Availability Scoring system developments since 2018

The R&D of the scoring engine has also continued after the initial description in Publication VI. Although not academically published, the following developments of the Availability Scoring system aid in providing a more relevant comparison with some of the more recent related publications from other authors (e.g., papers [131, 125] by Ošlejšek, et al. and paper [164] by Tovarňák, et al.).

Extensive use of container technology The Availability Scoring framework first employed Docker containers in 2018 to verify the functionality of 22 BT VPN services in parallel without one interfering with the other. Testing a VPN tunnel connectivity typically involves applying network settings retrieved from the corresponding VPN server onto the connecting client. Furthermore, establishing the connection and testing if required services work takes some time, so checking 22 teams in sequence would have taken too long. Alternatively, deploying multiple VMs for this lightweight yet specific task seemed to be excessive in terms of resources consumption and management overhead. Thus, deploying Docker containers for each BT was a sensible and lightweight method of executing all checks in parallel and independent of one another. The following years brought along even larger adoption of Docker container technology in order to conveniently proxy scoring checks within numerous BT internal network segments.

Integrations of the Availability Scoring data feed The live data feed from the Availability Scoring system has been integrated into RT tools to signal the RT members regarding the current status of particular services. For example, the Availability Scoring data can indicate that the targeted service is unavailable before the RT launches an attack against that resource. This forewarning can help RT to remain stealthy and not reveal their attack methods before the target services are actually available.

Furthermore, the Availability Scoring data feed has become an authoritative source of information for the GT as well. The information about the state of all monitored BT systems provides an excellent overview the general BT infrastructure. Prior to the start of the game, GT has to assure that all services are functional and configured identically for all BTs.

4.6 Comparison with related work

This section compares the work presented in this chapter with the related work described in section 2.1. Discussion has been grouped by individual exercises and compares the systems which aim to increase the training experience by providing immediate and improved feedback to CSX participants. Furthermore, this section provides observations for readers seeking to implement such frameworks. The purpose of this section is not to compare exercises in detail.

Note, that since the primary training audience of XS is the RT, the entire purpose and arrangement of the exercise seems to be rather unique. Alternatively, the reason for this

could be that training the red team responsive capability is considered a sensitive topic and as such is often not publicly discussed. However, there is slightly more material available on CDXs.

4.6.1 Cyber Conflict Exercise

The South Korean Cyber Conflict Exercise (CCE) described in [72, 73] follows the CDX model similar to the LS exercise. Unfortunately, the papers do not reveal much technological detail about the scoring system. Therefore, the primary developer of the CCE scoring system was interviewed to gain more insight about the entire framework[6]. The scoring system designed for the exercise is called TeSLA (short for **Te**sting **SLA**). The author of TeSLA had no prior organisational monitoring background, but rather was an experienced software developer.

Architecture Even though the authors of the two systems have different backgrounds, TeSLA still bears many similarities with the Availability Scoring solution described in this thesis. For example, there is a central scoring server that holds the configuration of all necessary scoring checks, but many of the checks are executed on proxy nodes that are located in numerous internal BT segments. Similarly, these proxies are running in Docker containers. The public-range IP addresses used by TeSLA are not changed during the game, however, this is compensated by deploying a large number of scoring proxy nodes the public network segment and using hundreds of IP addresses simultaneously.

Sustainability While the Availability Scoring framework is composed of many standard IT monitoring tools that are combined with custom integrations and data processing tools, the author of TeSLA decided to implement most of the components (i.a., standard monitoring functionality) in Node.js [129]. Although this approach ensures applicability to the corresponding CSX, it also results in a very particular software solution where the developer may very well be the only person who is able to fix bugs and problems, should they be encountered. According to the developer, since this was a newly developed software, there were indeed several bugs that had to be fixed during the game or improved between exercise iterations.

Score calculation TeSLA used a function linear to the corresponding uptime value to calculate the availability score points. Compared to the exponential function used in the Availability Scoring framework, a linear function is simpler for participants to understand and predict, however, the downside of the linear score coefficient is that BTs with relatively different uptime values experienced only modest differences in scored points. For instance, when considering the prior discussion about high-availability service providers, a BT with an average uptime of 75% performed far worse than a team with an average uptime of 90%, however, the difference in accrued points would average to 15% (given identical service point budgets).

4.6.2 Cyber Czech

The research group from Czech Republic working on the R&D of the Cyber Czech exercise has published a number of papers [131, 125, 126, 164, 178, 179] that describe both the exercise and the SA systems deployed during the event. The authors acknowledge that the Cyber Czech exercise is similar to the LS exercise which serves as good example of a

[6] The interviews with YoungJae Maeng took place on multiple occasions in August 2020

CSX. The publications are rather detailed and describe the technical background for better understanding and reproducibility.

In addition to information described in numerous publications, the author of this thesis also contacted[7] the Cyber Czech technical and analytical team and received a brief demonstration of their systems and visualisation tools.

Architecture Similarly to the Availability Scoring system, their central check processor is also based on automated checks implemented in Nagios. Furthermore, they have also implemented network traffic analysis and event log monitoring. These make up the essential components of their exercise-specific SA system as well as the primary input for technical exercise scoring. The systems integrate with the RT attacks to provide more detailed a timeline of in-game events.

Visualisations The Cyber Czech team has developed comprehensive visualisations as well as a separate highly integrated visual analytics tool to provide BTs detailed feedback about their progress in the game and skills in various areas of expertise (e.g., web, Windows, Linux, etc.) [131, 125]. However, according to the interview they have not tested their visualisation solutions with a large number of participating BTs. The number of participating BTs has generally been between four and six which is much less compared to LS where the number of BTs has exceeded 20 for the past several years.

Capturing and storing exercise data A paper by Tovarnak [164] provides detailed insight into the systems used for network traffic (PCAP) and log data capturing during the Cyber Czech exercise. This paper compares well with some of the prior discussions regarding Frankenstack in section 4.4. Their team has released the exercise dataset collected during a CSX that took place in the KYPO Cyber Range Platform in March 2019 [163].

The Cyber Czech team has opted to capture the PCAP from the central gateway node, however, they are additionally processing the network PCAP and exporting it as IPFIX[8] (**I**nternet **P**rotocol **F**low **I**nformation **E**xport) flows. Capturing traffic from the central gateway nodes resembles more the ISP point of view for traffic analysis. Comparatively, we decided to not use NetFlow or IPFIX within our exercise environments and relied on Suricata IDS to provide flow information. Moreover, network traffic in XS and LS is captured from all in-game networks segments using the Cisco ERSPAN functionality, resulting in full network visibility.

Logs were collected using respective OS-based tools and forwarded to a central collector running Logstash. Similarly, the collected logs are already stored in structured JSON format. Modifications to this log collection by the BT was forbidden according to exercise rules, nevertheless, the paper states that any unsanctioned actions taken by BTs might have resulted in loss of events in the log.

[7] The interview with the Cyber Czech research team took place via Zoom on June 15, 2020.

[8] An IETF protocol that is based on NetFlow version 9.

5 Event log analysis and knowledge discovery

This chapter describes the novel data clustering and event pattern mining algorithm called LogCluster. The algorithm and its use cases were initially introduced in Publications II and III. Furthermore, in October 2020 the author conducted a follow-up performance comparison of our algorithm alongside several other pattern mining algorithms which have been released over the past few years. The experiment and the results are described in section 5.3 below. This chapter addresses research question RQ2.1. The research was originally published in Publications II and III. Thesis contribution 3 is described.

Modern systems often produce large volumes of system and application logs, rendering manual review of those events infeasible. Understanding event patterns is an important step in log analysis process in order to discover *frequent patterns*, develop event log monitoring and correlation rules. However, in some cases the opposite applies—*outliers* (e.g., infrequent or unusual events) might be the most important ones to discover as a critical error or a log entry describing a security breach might exhibit itself just once. Motivation for engaging in log analysis and clustering effort can vary between different industrial and research applications. For instance, log exploration, duplicate issue discovery, failure or incident diagnosis, analysis automation, anomaly detection, and performance evaluation are common endeavours addressed by both researchers and industry practitioners.

5.1 Description of LogCluster

The LogCluster algorithm was designed to analyse textual event logs to discover both *frequent line patterns* as well as *outlier* events. LogCluster addresses several shortcomings of pre-existing event log clustering algorithms discussed in papers [168, 144, 100]. In particular, LogCluster resolves several issues of the SLCT algorithm, such as detection of wildcards after the last word in a proposed *line pattern*, sensitivity to delimiter noise and discovering shifts in word positions. The LogCluster tool is distributed under the terms of GNU GPL and is available from [170]. At the time of writing, the latest version of LogCluster is 0.10 which was released on March 20, 2019. LogCluster processes logs in a single-threaded fashion, however, multiple processes can be launched for independent analyses.

LogCluster uses *frequent pattern mining* to address the log clustering problem. LogCluster expects the *support threshold* `s` as user-defined input parameter (if the event log contains `n` lines, then $1 \leq s \leq n$). To detect cluster candidates, LogCluster identifies line patterns by parsing the textual event logs. In the default operating mode, LogCluster makes two passes over the log data: first to identify *frequent words* (i.e., words that appear in at least `s` event log lines) and second to generate *cluster candidates* (assigning one or more event log lines to each candidate). Configuring certain additional features (e.g., *outlier detection*) invokes an additional pass over the dataset.

Listing 5: Example dataset for log clustering.

```
Interface eth0 down
Interface eth1 up
Interface HQ link down
```

Each detected cluster candidate is uniquely identified by its line pattern which matches all lines assigned to the cluster candidate. Line patterns consist of frequent words and wildcard notations. The *support* of a particular cluster candidate is defined as the number of event log lines assigned to the cluster candidate. Finally, after cluster candidates have been detected, candidates with the support of at least `s` are selected as *clusters*. To illustrate, an example dataset is provided in Listing 5. If `s=2`, the detected line pattern

`Interface *{1,2} down` consists of frequent words *Interface* and *down*, and the wildcard notation **{1,2}* which matches at least one and at most two words. Events `Interface eth0 down` and `Interface HQ link down` both belong to the cluster represented by the pattern `Interface *{1,2} down`.

Outliers are lines which do not belong to any of the detected clusters (i.e., cluster candidates with the support of at least `s`). In the example above, when `s`=2, the event `Interface eth1 up` would be considered an outlier. Detection of outliers is not enabled by default but can be enabled by using the relevant command line parameter.

In addition to specifying the input log file to process, the support threshold is the only mandatory input parameter the user has to provide when using the LogCluster tool (see example #1 in Listing 6). To ease the exploration of unfamiliar log files, LogCluster also enables the user to define a *relative support threshold* which denotes the percentage from total lines read from the input. Although results vary with different types of logs, our experiments indicated that a relative support value in the order of 0.1% and 1% is a sensible starting point.

The experiments also revealed some use cases where it might not be optimal to attempt clustering the events with just one LogCluster execution. For instance, when clustering event logs with a large variety of log messages from many hosts, higher support thresholds are likely to yield too many outliers. Contrariwise, lower support thresholds will produce a large number of clusters. For addressing this issue, LogCluster can be run iteratively to cluster the outliers that were detected from the previous execution at each consecutive step (see example #2 in Listing 6). Such multi-step approach enables to identify clusters of events which occur at various frequency levels.

LogCluster has several data pre-processing capabilities to improve the clustering outcome. For example, it is possible to apply additional line filters to only process lines that match the user-defined regular expression (see example #3 in Listing 6). Furthermore, when the line filter regular expression is set to store match variables and used in conjunction with the line template function, it is possible to alter the original input log line before it is processed. For instance, example #4 in Listing 6 removes the preceding syslog timestamp and hostname, leaving just the syslog tag prefix along with the message. This approach can be used to remove excessive noise arising from changes in the timestamp and hostname segment of syslog events.

Moreover, words often share the same format that is not detected during clustering due to some changing parameters. For instance, the program name of the syslog message is followed by a process ID that can change frequently, resulting in many infrequent words for the same program (e.g., `sshd[40515]:` and `sshd[261017]:`). For addressing this issue, LogCluster enables to mask particular words or word parts. This can be achieved using a regular expression filter and search pattern to replace the matching section of the log line with a given string. Such masking can be used to replace values (e.g., process ID numbers, IP addresses, port numbers, timestamps, etc.) that often change between otherwise similarly formatted words. See example #5 in Listing 6 for replacing SSH daemon process ID with a string *PID*. Note, that this approach actually sets up another word class *sshd[PID]:*, which is treated like a regular word by LogCluster. If a corresponding word class is frequent, it replaces all infrequent words during the clustering process. However, if both the original word (e.g., `sshd[40515]:`) and the corresponding word class (i.e., `sshd[PID]:`) are frequent, the original word is given preference during the clustering process.

Finally, in addition to regular expression based line parsing and word class creation, the LogCluster tool supports defining custom Perl functions for these tasks and enables

Listing 6: Sample LogCluster invocations in a log discovery process (omitting output).

```
#1 - Simple invocation using an absolute support threshold value
logcluster.pl --input=auth.log --support=100

#2 - Iterative clustering using a relative support threshold
logcluster.pl --input=auth.log --rsupport=0.5 --outliers=outliers-i1.log
logcluster.pl --input=outliers-i1.log --rsupport=0.5 --outliers=outliers-i2.log
logcluster.pl --input=outliers-i2.log --rsupport=0.5 --outliers=outliers-i3.log

#3 - Process only SSH daemon messages from auth.log
logcluster.pl --input=auth.log --rsupport=1 --lfilter='sshd\[\d+\]:'

#4 - Process sshd messages from auth.log, remove syslog timestamp and hostname
logcluster.pl --input=auth.log --rsupport=1 \
              --lfilter='(sshd\[\d+\]: .+)' --template='$1'

#5 - Process sshd messages, remove timestamp and hostname, replace process ID
logcluster.pl --input=auth.log --rsupport=1 \
              --lfilter='(sshd\[\d+\]: .+)' --template='$1' \
              --wfilter='^\w+\[\d+\]:$' --wsearch='\[\d+\]' --wreplace='[PID]'
```

loading such functions from external Perl modules. This allows for addressing complex data pre-processing tasks which cannot be handled by regular expressions alone.

When using lower support thresholds, LogCluster can sometimes overfit and needlessly split meaningful line patterns into too specific ones. For example, LogCluster can output the following two line patterns which both exceed the desired support threshold: `Interface *{1,1} up` and `Interface eth0 up`. To address such overfitting, LogCluster supports two heuristics for joining clusters to produce fewer and more generic line patterns that are likely more comprehensible for human operators.

First heuristic to join clusters detects cluster overlap and aggregates supports of relevant cluster candidates before final clusters are formed. For each cluster candidate other candidates are analysed to identify the ones with more specific line patterns. Consequently, the lines corresponding to more specific candidates are assigned to the more generic cluster candidate.

Second heuristic employs various word weight functions and is applied after clusters have been selected. LogCluster defines several word weight functions[9] which measure how closely (ranging from 0 to 1) each word in the *line pattern* is correlated to other words in this pattern. For identifying words with insufficient weights, the heuristic requires a user defined *word weight threshold* ($0 < t \leq 1$). When *clusters* are joined, the *supports* of original clusters are summed and the resulting *line pattern* is adjusted to describe the lines in all original clusters: words with insufficient weights are combined into compact lists of alternatives and wildcards are joined. Such approach enables knowledge discovery from original patterns in a concise manner. When evaluating this cluster joining heuristic, we found that word weight thresholds ranging between 0.5–0.8 produced the most reasonable joint clusters.

The LogCluster usage example in Listing 7 brings most of the mentioned features together. The output represents a *joint cluster* produced from the LS15 Availability Scoring system log file containing a total of 3,400,185 events. The example cluster represents the union of twelve clusters which have been combined using the word-weight-based *cluster joining heuristic*. Contextually, this cluster combines all successful SSH service checks using both IPv4 and IPv6 for 192 different hosts.

[9] LogCluster v0.10 includes four different word weight functions.

This usage example makes use of several input parameters, some of which might require an additional explanation. The clustering is performed with a relative support threshold of 0.1% which results in the support threshold of 3,400 lines. To accommodate parsing the custom vertical bar (|) separated format of the LS Availability Scoring logs (see Listing 2 in the section 4.5 above for examples), the word separator regular expression has been set to match the double quote and vertical bar symbols as well as all whitespace characters. Line filtering and template options are used to remove the check start and end timestamps at the last two positions of each event. Word classes are used to mask specific BT numbers (e.g., 04 or 16) within the hostnames with a generic *NN* notation. The outliers are written to the *ls15-outliers.log* file for further analysis.

Listing 7: Example of using LogCluster to analyse LS Availability Scoring system logs.

```
logcluster.pl --input=ls15.log --rsupport=0.1 --separator='["|\s]+' \
              --lfilter='^(.*)(?:\|"\d+"){2}' --template='$1' \
              --wfilter='blue\d\d' --wsearch='blue\d\d' --wreplace='blueNN' \
              --wweight=0.5 --outliers=ls15-outliers.log

(ws4-01.lab.blueNN.ex|ws4-04.lab.blueNN.ex|ws4-03.int.blueNN.ex|ws4-04.int.blueNN.ex
|ws4-02.int.blueNN.ex|ws4-05.lab.blueNN.ex|ws4-05.int.blueNN.ex|dlna.lab.blueNN.ex
|ws4-01.int.blueNN.ex|ws4-02.lab.blueNN.ex|ws4-03.lab.blueNN.ex|git.lab.blueNN.ex)
(ssh|ssh.ipv6) OK SSH OK - (OpenSSH_6.6.1p1|OpenSSH_5.9p1|OpenSSH_6.6.1_hpn13v11)
(Ubuntu-2ubuntu2|FreeBSD-20140420|Debian-5ubuntu1|Debian-5ubuntu1.4) (protocol 2.0)
```

Rare events may easily go unnoticed in large log files. Iterative processing of the outliers file (described in example #2 in Listing 6) may reveal interesting events or unexpected situations that only occur rarely during the CSX. For instance, further processing of the outlier file *ls15-outliers.log* from Listing 7 resulted in an even smaller set of outlier events. Listing 8 describes one problematic check result from a time when the BT was updating the SSL configuration of its mail server. Notably, the event triggered an exception state along with fairly verbose error message within the check script. This was something that needed to be followed up and improved within the script itself. Moreover, such verbose error messages should not be printed into the Availability Scoring data feed but rather logged separately for further investigation.

Listing 8: Problematic error message detected from the outliers of the Availability Scoring log.

```
"mail.blue06.ex"|"imap_receive"|"CRITICAL"|"IMAP RECEIVE CRITICAL - Could not connect
to 10.6.1.4 port 993: IO::Socket::IP configuration failed error:00000000:lib(0):
func(0):reason(0) (if you get this only when using both --ssl and --ssl-ca-file, but
not when using just --ssl, the server SSL certificate failed validation) at
check_imap_receive line 138."|"1429686177"|"1429686177"
```

5.2 Discussion of related work

This section summarises prior comparisons of LogCluster and other relevant log mining tools.

Comparison with SLCT SLCT is a direct predecessor to LogCluster. Before proceeding to conduct new comparisons, it would make sense to first recap our prior experiment in Publication II where we performed a thorough comparison of SLCT[10] and LogCluster. Both algorithms use frequent pattern mining to detect log clusters. Performance evaluation

[10]The SLCT algorithm was re-implemented in Perl language for a more suitable side-by-side comparison with LogCluster.

with Perl-based implementations indicated that the SLCT algorithm was 1.28–1.62 times faster than LogCluster. This is because SLCT does not check and adjust the line patterns of cluster candidates, resulting in a simpler candidate generation procedure. However, this relates directly to the shortcomings of SLCT (e.g., not discovering shifts in word positions) that were mentioned above.

Nevertheless, a recent comparison of SLCT (originally written in C) with a simplified C-based prototype[11] of LogCluster revealed that C-based implementations of both algorithms have roughly the same speed [184]. The reason for this is the fact that SLCT encodes position information into words which increases the length of words and therefore also the computational cost of word hashing. With C-based implementations, this extra cost can even exceed the cost of more complex candidate generation procedure of LogCluster.

Perl-based implementations of both algorithms proved to be relatively efficient in terms of processing speed. However, processing remarkably large log files still consumed a considerable amount of time. In case of the largest event log file (16.3GiB and over 49 million lines) SLCT implementation needed about 1.5 hours to complete, while LogCluster's runtime took slightly over 2 hours. Note, that C-based implementations were significantly faster—for example, the original version of SLCT written in C processed the same 16.3GiB log file in 19 minutes. Thus, the Perl implementation was approximately 4.6 times slower.

Automated benchmarking framework by LogPAI As mentioned in the Related Work section 2.2.1, the comparative analysis and automated benchmarking framework described in [183] by Zhu, et al. is likely one of the most comprehensive comparisons of various log parsers that has been published within the past few years. While their extensive research endeavour provided plenty of valuable insight to the author of this thesis, there are several shortcomings that were encountered while reviewing their implementation of the automated benchmarking framework.

When describing the experiment setup, the paper provides an extensive description of the Loghub project [93] which comprises 16 different log datasets and contains a total of 440 million log messages that sum up to about 77GiB in size. However, the benchmarks were conducted using only a sample of 2000 lines from each log type. On one hand, this limitation is justified, because the accuracy evaluation methodology used manual data labelling for establishing event templates as ground truth. However, on the other hand, assessing the efficiency and performance of various log parsers based on just 2000 events is hardly sufficient. Moreover, for some log types there are additional regular expression pre-filters which further reduce the number of events that are passed on to log parsers as input. To alleviate the situation, the second phase of the experiment selected six log parsers (MoLFI, LenMa, AEL, Spell, IPLoM and Drain) for further testing on three larger 1GiB datasets—Android, BGL, and HDFS log files.

Furthermore, the relative support threshold values used for LogCluster are not optimal. As mentioned, the benchmark iterates over a set of 16 different log types. LogCluster configuration for six out of 16 log types specify an abnormally high relative support value of 10% and higher, in some cases 30–40%. While moderately higher support values might be justified in some cases, it is important to consider that we have suggested using significantly lower relative support thresholds (e.g., between 0.1% and 1%) with LogCluster. For example, specifying a relative support threshold of 40% for a 2000-line Linux syslog event log file would mean that the resulting cluster(s) would have to contain at least 800 similar events. This is rather unrealistic considering the large variety of Linux syslog events.

[11] The simplified C-based LogCluster prototype implements only a subset of the main Perl-based LogCluster tool and is based on its older version.

Finally, the paper claims (as shown in Table II in [183]) that LogCluster tool output does not provide a full coverage of analysed log files and lacks log pre-processing capabilities. As described above, this is not true. First, if LogCluster is executed with the optional *outlier detection* functionality, then its output covers the entire input log file. Second, LogCluster offers many event pre-processing functions (e.g, see Listings 6 and 7), however, there are no mandatory pre-processing steps and logs can also be parsed just as they are.

5.3 Comparison with newer log mining algorithm implementations

This subsection describes the experiment with LogCluster (published in 2015) and several more recently published log parsing tools to understand how LogCluster compares with similar but newer competitors. The experiment provides an assessment of the following primary aspects:

- Efficiency: Time and resource consumption evaluation;
- Features: Discussion of common and unique features;
- Usability: Ease of use and applicability of log mining results.

5.3.1 Experiment setup

To reduce any potential interference and uncertainty arising from using shared environments and virtualisation, all benchmark experiments were conducted on a dedicated physical Linux host[12] built on an ASUS motherboard[13] coupled with the 10th generation Intel Core i9 CPU [14] and 256GB of memory[15]. The storage device used in the experiments was a 2TB NVMe SSD drive[16]. Software version for Perl was 5.32 and for Python 3.8.3.

The experiments were conducted using two different log types from CSXs: Availability Scoring system logs (from Locked Shields) and Linux syslog server logs (from Crossed Swords). First, the Availability Scoring system log file (hereafter indicated as LF1) contains the results of all service checks (both OK and not-OK) from LS15. As already described above, the Availability Scoring system events use a custom vertical bar separated format. Therefore, a custom regular expression that matches the vertical bar symbol as well as whitespace characters as word separators has to be defined (see Listing 7). The 392MiB file contains a total of 3,400,185 events spanning approximately 16 hours of LS gameplay. Second, the 4.6GiB Linux syslog file (hereafter indicated as LF2) contains 27,365,365 events from the XS19 central syslog server that has collected logs from 54 unique hosts over the course of 10 days. The syslog events were logged with rsyslog using the high-precision timestamp format.

The experiment executes a log exploration scenario where a log mining utility is used to parse large unstructured log files without applying any kind of advanced masking or pre-filtering which is possible only when the analyst has already gained a comparatively good overview of the contents of corresponding log files. The comparison investigates efficiency, available features, ease of use in various applications, and an assessment of each tool's output.

[12] Manjaro Linux 20.1.1 (*Mikah*), kernel version 5.8.11-1-MANJARO
[13] ASUS Prime X299-Deluxe II
[14] Intel Core i9-10920X
[15] Crucial DDR4 2666MHz BL32G32C16U4B.M16FB1
[16] Gigabyte NVMe SSD GP-ASM2NE6200TTTD

5.3.2 LogCluster results

For the experiment, the latest available version of the LogCluster tool (LogCluster v0.10) from the LogCluster homepage [170] was used. The LogCluster tool is executed with the *outlier detection* to provide a full coverage of the input log file.

LogCluster was able to parse the LF1 log file with good efficiency and concluded in 78 seconds (i.e., 1m 18s). Since LogCluster implementation is single-threaded and its CPU-thread utilization was 100% according to the GNU *time* utility, then each LogCluster runtime closely matches CPU time that was consumed during this and the following experiments. Memory use peaked at 228MB. LogCluster produced a total of 39 clusters using the relative support threshold of 0.5% (s=17000). The resulting 39 line patterns match 1,750,405 input events (71.7%), the rest (i.e., 962,866 events) were considered as outliers. An example of four clusters is depicted in the #LF1 section of Listing 9. The leading hostnames and the timestamps at the final two positions are replaced by corresponding wildcard notations in all line patterns.

Listing 9: Example of LogCluster output.

```
# LF1
# logcluster.pl --input=ls15.log --rsupport=0.5 --outliers=ls15.outliers \
#               --separator='["|\s]+'

*{1,1} rdp OK x224 OK. Connection setup time: *{1,1} sec. *{2,2}
Support: 420322

*{1,1} rdp.ipv6 OK TCP OK - 0.001 second response time on *{1,1} port 3389 *{2,2}
Support: 271926

  --- 35 patterns omitted ---

*{1,1} ntp.ipv6 OK NTP OK: Offset *{1,1} secs *{2,2}
Support: 18209

*{1,1} ntp OK NTP OK: Offset *{1,1} secs *{2,2}
Support: 17747

# LF2
# logcluster.pl --input=syslog.log --rsupport=0.1 --outliers=syslog.outliers

*{1,1} vts-server.gdt.clf.ex vhf-radioserver[929]: (VHF AIS2) Received AIS sentence
  from tcp:10.242.10.192:9010: *{1,1}
Support: 1084528

*{1,1} vts-server.gdt.clf.ex vhf-aisclient[859]: AisClientInterface: AIS out: *{1,1}
  (VHF AIS2)
Support: 1084489

*{1,1} nagios.clf.ex *{1,1} [login:nagios ssh:((undefined)) username:nagios uid:105
  group:nagios gid:107 sid:407 tty:(none) cwd:/tmp filename:/bin/ping6]:
  /bin/ping6 -n -U -w 15 -c 3 *{1,1}
Support: 735860

  --- 291 patterns omitted ---

*{1,1} nagios.clf.ex *{1,1} [login:nagios ssh:((undefined)) username:nagios uid:105
  group:nagios gid:107 sid:407 tty:(none) cwd:/tmp filename:/bin/ping]:
  /bin/ping -n -U -w 33 -c 3 10.242.6.7
Support: 27968
```

When assessing the clusters detected from LF1, it becomes evident that 37 (i.e., 94.9%) of them are representing events with an *OK* state. As these events describe the uptime of services, then *OK* states are naturally more frequent and less likely to change compared to events describing fault or critical states. This means that many fault events were classified as outliers. Based on this observation, it might make sense to take additional steps when analysing this log file, e.g., attempt iterative clustering of outliers as we described

in example #2 in Listing 6 or employ more advanced heuristics to aggregate the supports of less frequent cluster candidates.

Parsing of the LF2 took slightly longer: 549 seconds (i.e., 9m 9s). However, considering the larger size (11.7 times) of the input file compared to LF1, this could still be considered highly efficient. The memory consumption of the LogCluster process was also notably higher, peaking at 6GB. However, LogCluster's memory use can be reduced by enabling various memory optimisation techniques (e.g., sketching) at the expense of an additional pass over the input data. For instance, when using a word sketch with 100,000 counters (`--wsize=100000`) the process runtime increased to 14m 54s (increase of 5m 45s) due to the additional pass over the data, however, LogCluster's memory consumption dropped by approximately 98.7% to just 80MB. LogCluster produced a total of 295 clusters using the relative support threshold of 0.1% (`s=27365`). The 295 line patterns match 20,409,430 input events (i.e., 74.6%), the rest (i.e., 6,955,935) were considered as outliers. An example of four clusters is depicted in the #LF2 section of Listing 9. The leading timestamp has been replaced by the wildcard notation in all 295 line patterns. The same applies for the syslog tag (e.g., *sshd[261017]:*) in many cases where the process ID number has changed over time. When assessing the line patterns from LF2, it seems that some systems (e.g., AIS - the vessel tracking **A**utomatic **I**dentification **S**ystem) and programs (e.g., Snoopy logger events for Nagios) have been extremely verbose. Although using additional cluster joining heuristics would merge some of those clusters, in some cases it might make sense to set up a line filter to exclude these lines from the analysis to better focus on other events.

5.3.3 Comparison with Drain

For the experiment, the latest available version of the Drain tool (Drain3 v0.7.9) from the Python Package Index [139] was used. Drain3 is an updated version of the original Drain implementation that did not support python version 3 [61]. Drain is an online log parser, meaning that it can analyse a stream of events. It can be configured to periodically dump its current state to Apache Kafka, Redis [142] or a file on disk. The data dump also enables Drain to maintain a persistent state across restarts. Processing takes place in a single-threaded manner and the output offers full coverage of the input log file—even line patterns with just one input line are returned. This behaviour is intentional since Drain is meant to process streams rather than complete files, so under normal operating conditions a new line matching an existing cluster with a single member could come in at any time.

First thing to note is that instead of using command-line parameters, Drain loads a configuration file which specifies three user-defined input parameters which come with default pre-defined values[17]. Additionally, the configuration file enables the user to specify regular expressions for masking commonly occurring patterns (e.g., process ID, IP addresses, date, and time values) with a fixed string placeholder before the log mining commences. For LF2 no regular expression masking was used, however, for LF1 masking feature was used to handle the custom word delimiter conversion to single whitespaces. Since Drain is designed for stream processing, the most reasonable method was to feed the input event logs via the program's standard input (*stdin*) stream.

Drain demonstrated good efficiency for LF1 which was processed in 113 seconds (i.e., 1m 53s). Since Drain3 implementation is single-threaded and its CPU-thread utilization was 100% according to the GNU *time* utility, then the actual runtime closely matches CPU time that was consumed. Maximum process memory usage was only 16MB. The process

[17] Default parser configuration for Drain3: `sim_th=0.4, depth=4, max_children=100`

yielded 90 clusters (see four examples under #LF1 in Listing 10). However, when comparing Drain's output with LogCluster, only 17 clusters detected by Drain contained more (i.e., 17,000) members than what was set by the support threshold for LogCluster.

Drain's processing speed deteriorated when file size grew larger. LF2 was processed in 1 hour 21 minutes 21 seconds. Maximum process memory usage was only 26MB. The mining yielded 4962 patterns (see four examples under #LF2 in Listing 10). When compared with LogCluster, 91 clusters detected by Drain contained more (i.e., 27,365) members than what was set by the support threshold for LogCluster.

Cluster IDs are assigned incrementally (e.g., A0001, A0002, etc.) as new line pattern templates are identified. When messages are added to an existing cluster, the corresponding line pattern is updated, as necessary. The output is not sorted by the size of individual clusters, but rather by the order of cluster IDs.

Produced line patterns are understandable, however, several identified line patterns seem too specific and should have rather been joined together. For example, in case of LF1, nine line patterns (10%) were formed around the single term *OK* and the two UNIX timestamps at the end of each message (e.g., *1429684088< >1429684088*). In case of LF2, similar issue could be observed. For instance, the changing PID number of the mail server's SMTP daemon has resulted in a total of 214 distinct line patterns where the only changing element is the process ID number (e.g., *postfix/smtpd[**10948**]:*). Unfortunately, Drain's functionality does not seem to include any post-processing heuristics to join similar line patterns. Therefore, to avoid such unwanted behaviour the end-user should study the log file and prepare a set of regular expressions to mask such varying elements in log messages. Furthermore, according to the paper [55] describing the original algorithm, Drain assumes that log messages describing a similar event contain the same number of words. This means, for example, that Drain is unable to correctly handle the log clustering scenario presented in Listing 5 where one of the log messages has an additional word in it.

Listing 10: Example of Drain3 output.

```
# LF1
# sim_th=0.4, depth=4, max_children=100
A0001 (size 1316465): < <*> <*> >OK< >TCP OK - <*> second response time on <*> port
  <*> <*> <*> >
A0002 (size 200035): < <*> <*> <*> >HTTP <*> HTTP/1.1 <*> <*> - <*> bytes in
  <*> second response time< <*> <*> >
A0003 (size 35957): < <*> <*> >OK< >NTP OK: Offset <*> secs< <*> <*> >
  --- 86 patterns omitted ---
A0090 (size 1): < >dev.blue06.ex< >wiki_weblogin< >CRITICAL< >HTTP_AUTH CRITICAL -
  Cannot connect to http://wiki.blue06.ex/index.php?title=Special:UserLogin&action=
  submitlogin&type=login&returnto=Main+Page: write failed: Connection reset by peer
  at /usr/local/share/perl/5.14.2/LWPx/TimedHTTP.pm line 251.
  < >1429706566< >1429706567< >

# LF2
# sim_th=0.4, depth=4, max_children=100
A0001 (size 2276810): <*> nagios.clf.ex <*> [login:nagios ssh:((undefined))
  username:nagios uid:105 group:nagios gid:107 sid:407 tty:(none) cwd:/tmp
  <*> <*> <*> <*> <*> <*> <*> <*> <*>
A0002 (size 23177): <*> <*> <*> <*> DEBUG [SerialGpsLocationListenerHandler.java:47]
  Tried to open serial, didn't find any port with given name, have these ports: []
A0003 (size 1084528): <*> vts-server.gdt.clf.ex vhf-radioserver[929]:
  (VHF AIS2) Received AIS sentence from tcp:10.242.10.192:9010: <*>
  --- 4958 patterns omitted ---
A4962 (size 1): 2019-02-02T04:37:01.543519+00:00 vts-server.gdt.clf.ex su[31211]:
  - ??? root:vhf
```

5.3.4 Comparison with LogMine

For the experiment, the latest available version of the LogMine tool (logmine v0.2.2) from the Python Package Index was used. Similarly to LogCluster, LogMine is also an offline log parser, meaning there is no support for stream processing. Subjectively, LogMine is easy to use and provides basic usage information within the tool's help function.

LogMine enables the user to configure several command line parameters. Besides the input log file, none of them are mandatory for the user to specify, since they all come with some default values[18]. By default, LogMine is configured to detect clusters which have at least 2 members. In the experiments, this threshold was set to 1, so LogMine output would cover the entire input file similarly to LogCluster and Drain in prior experiments. Note, that LogMine is the only multi-threaded log processor in this experiment.

LogMine was able to parse the LF1 log file with relatively good efficiency and concluded in 100 seconds (i.e., 1m 40s). The process yielded 67 clusters with *min-members* set to 1 (see four examples under #LF1 from Listing 11). When compared with LogCluster, 16 clusters detected by LogMine contained more (i.e., 17,000) members than what was set by the support threshold for LogCluster. Since LogMine implementation is multi-threaded, it was able to utilise all available (i.e., 24) CPU threads, resulting in 38m 20s of consumed CPU time and an average of 96% CPU utilisation according to the GNU *time* utility. Process memory use peaked at 86MB. Furthermore, it seems that LogMine tends to produce more generic line patterns when compared with LogCluster or Drain. For example, full coverage of LF1 was achieved with just 67 clusters out of which 64 had more than one member, additionally, the pattern `* * * * * * * * * * *` containing just eleven wildcard notations matched 35,934 input lines. Alternatively, parsing LF1 with a single thread took 21m 43s (i.e., approx. 13 times longer) to complete.

Parsing LF2 took noticeably longer and concluded in 36 minutes and 52 seconds. The process yielded 3012 clusters with *min-members* set to 1 (see four examples under #LF2 from Listing 11). When compared with LogCluster, 77 clusters detected by LogMine contained more (i.e., 27,365) members than what was set by the support threshold for LogCluster. LogMine analysis resulted in 5.0 hours of consumed CPU time and an average of 34% CPU utilisation per thread according to the GNU *time* utility. The average CPU utilisation is rather low because some of the threads finished processing much quicker than others. Process memory use peaked at 909MB. Again, the performance in single-threaded mode is very slow—parsing LF2 took approximately 22 hours to complete.

LogMine demonstrated remarkably high CPU usage. With the default settings, LogMine exhibited relatively fast processing speeds on modern multi-core CPUs. However, except for modifying the number of minimum cluster members (i.e., *min-members* value), most command-line parameter (e.g., the *max-dist* or the *K1* and *K2* weight values) modifications resulted in highly increased processing times. For example, when setting *line-distance-weight-K1=0.5* the parsing of LF1 took 13h 54m. An attempt to parse LF2 with the same parameters exceeded 22 hours before the process was manually interrupted. Although this change drastically improved LogMine output for LF1, it is just not feasible in case of larger log files.

It is important to note that LogMine output differs when it is executed in multi-threaded mode compared to the single-threaded mode. The authors have acknowledged this difference and claim that this is an expected side-effect. However, this raised some concerns regarding the reliability and reproducibility of LogMine results in various situations. The experiments revealed an output variation in results not just when switching between

[18] Default values of relevant LogMine command line parameters: `max-dist=0.6, min-members=2, line-distance-weight-K1=1, variable-weight-K2=1`

different operating modes, but also when switching between different test machines. It seems that in order to speed up the log mining process, input log lines are distributed between different threads based on some characteristics, so each thread sees only a fraction of the input file. The more threads a machine has, the less data each thread actually receives. Clusters that form on a small subset of data can be much different from clusters that would form when analysing the entire dataset. To verify this variation in practice, an additional side-experiment using another physical server with two highly multi-threaded CPUs[19] was conducted. Even with identical LogMine configurations the resulting clusters from LF1 were noticeably different when compared to results from the primary testing machine with just 24 CPU threads. Not only were there differences in cluster sizes, but some smaller clusters had not formed at all.

Listing 11: Example of LogMine output.

```
# LF1
# logmine --pattern-placeholder '*' --min-members 1 --delimiters '["|\s]+' ls15.log
# Output format: Cluster_members Line_pattern

1336365 * * OK * OK - * * response time * * * * * * * * * *
 420322 * rdp OK x224 OK. Connection setup time: * sec. * * *
 338388 * * OK * OK - * * * * * *
  --- 63 patterns omitted ---
      1 mail.blue10.ex smtp.ipv6 WARNING recv() failed 1429687172 1429687177

# LF2
# logmine --pattern-placeholder '*' --min-members 1 syslog.log

6270207 * appsrv.gdt.clf.ex start.sh[576]: #011at * *
2956762 * * * * ssh:((undefined)) username:nagios * group:nagios * * tty:(none)
  * * * * * * * * * *
1588265 * * unbound: * info: * * * IN
  --- 3008 patterns omitted ---
      1 2019-02-01T15:24:52.433481+02:00 fw.gdt.clf.ex smbd[23363]: PANIC
  (pid 23363): internal error
```

5.3.5 Comparison with Spell

For the experiment, the latest available version of the Spell tool (spellpy v0.0.9) from the Python Package Index was used. Similarly to Drain, Spell is also an online log mining algorithm capable of stream processing. However, the tool implementation seems to lack any directly configurable settings for loading, periodically outputting or storing its runtime state. Processing takes place in a single-threaded manner and the output provides full coverage of the input log file. Spell has one user provided parameter called *tau* (by default `tau=0.5`).

Spell demonstrated the lowest efficiency for parsing LF1 with default settings. LF1 was processed in 657 second (i.e., 10m 57s). Since Spell is single-threaded and its CPU-thread utilization was 100% according to the GNU *time* utility, then the actual runtime closely matches CPU time that was consumed. Maximum process memory usage was 3.5GB, which is the highest in the experiment. The process yielded 1148 clusters (see four examples under #LF1 in Listing 12). Spell provides full coverage of the input file and outputs all detected clusters. When compared with LogCluster, 22 clusters detected by Spell contained more (i.e., 17,000) members than what was set by the support threshold for LogCluster.

The low efficiency proved to be detrimental for Spell. At first, Spell could not finish loading LF2, instead a *Timeout exception* was raised. After inspecting the Python source

[19] Two AMD EPYC 7452 32-Core Processors that sum up to a total of 128 processing threads for the entire system.

code the *operation timeout* was manually increased from 1 second to 10 seconds for the following attempts. After this, LF2 was loaded successfully and even the process of parsing LF2 reached 100%, but unfortunately failed to produce an output. The process seemed to be stuck in some internal function timeout loop. The Spell process was manually interrupted after 22 hours. No output was produced. Memory usage peaked at 22.4GB.

Based on the results from LF1, the quality of line patterns is relatively good, comparable with Drain and LogCluster. Spell seems to suffer from one of the drawbacks discussed for SLCT above, particularly the ability to detect multiple wildcards at the end of the line pattern. By default, the output is not sorted based on the rate of occurrence, but the CSV formatted file can be easily loaded and sorted using other tools.

Listing 12: Example of Spell output.

```
# LF1
# tau=0.5
# Output format: EventId,EventTemplate,Occurrences
60e656d4,<*> <*> OK TCP OK - <*> second response time on <*> <*> <*> <*> <*>
  port <*>,1316438
e383799d,<*> <*> OK TCP OK - <*> second response time on 2001 10 <*> <*> <*>
  port <*> 1429684081 1429684081,27
2f190908,<*> <*> OK HTTP OK HTTP/1.1 <*> <*> - <*> bytes in <*> second response
  time <*>,247976
  --- 1144 patterns omitted ---
9b34512e,dc1.blue04.ex <*> CRITICAL Connection refused <*>,4

# LF2
# tau=0.5
  --- No output ---
```

5.3.6 Summary

Table 2 summarises the primary aspects (runtime, highest recorded memory use, total amount of clusters detected, and the amount of clusters where the member count exceeds the support threshold that was used for LogCluster) of the experiment. Qualitative analysis and discussion about detected line patterns is not feasible to be summarised in a table form and is only provided in the respective paragraphs. The following summary generalises some more prominent observations from this experiment. Note, that LogCluster results for LF2 also include performance data for running LogCluster with the word sketching memory optimization technique.

Table 2: Summary of the experiment comparing LogCluster with newer log mining tools.

Test type	**LogCluster**	**Drain**	**LogMine**	**Spell**
Runtime (LF1)	1m 18s	1m 53s	1m 40s	10m 57s
Runtime (LF2)	9m 9s / 14m 54s	1h 21m 21s	36m 52s	-
Max. memory use (LF1)	228 MB	16 MB	86 MB	3.5 GB
Max. memory use (LF2)	6.0 GB / 80 MB	26 MB	909 MB	22.4 GB
Total clusters (LF1)	39	90	67	1148
Total clusters (LF2)	295	4962	3012	-
Clusters if $s \geq 17000$ (LF1)	39	17	16	22
Clusters if $s \geq 27365$ (LF2)	295	91	77	-

LogCluster is the fastest log mining tool in this experiment. It has the most descriptive wildcard notation that also indicates how many words a particular wildcard can match. The high performance was accompanied by higher memory use with default settings compared to Drain and LogMine. However, by enabling word sketching, the peak memory use

of 6GB can be reduced to just 80MB at the cost of an additional pass over the input data. Therefore, users can easily adjust the balance between LogCluster's memory consumption and processing speed based on available computational resources. LogCluster has by far the most comprehensive set of user-configurable parameters for modifying the log mining process and presented outcome.

Drain is fast for smaller amounts of input data but gets significantly slower as the amount of input data increases. It consumed by far the least amount of memory in all experiments. The overall quality of line patterns was acceptable, however, many identified line patterns were anchored to highly varying elements of log messages (e.g., timestamps and PID numbers). Moreover, the large number of line patterns that were returned posed a problem of assessing the output. This aspect might be something that can potentially be tuned by the three user-configurable parameters. However, as there was no clear guidance on the effect that each of these parameters would have on the outcome, then this process was considered out of scope for this experiment.

LogMine is fast in multi-core mode but the mining results are not reliably reproducible on multi-threaded CPUs. The amount of available processor threads has an effect on the outcome of the mining process (i.e., the input data is split between processing threads). Single-threaded mode does not suffer from this adverse side-effect but is unfortunately exceedingly slow. Furthermore, LogMine tends to produce very generic line patterns (e.g., * * * * * * * * * * *) that can match a substantial amount of events.

Spell was the slowest log mining tool in this experiment. It was only capable of loading smaller files and timed out when LF2, the larger 4.6GiB syslog file, was analysed. Furthermore, it also used the most memory in the experiments—consuming over 22GB of RAM might become troublesome even for some modern workstations. Based on the mining results of LF1, the line patterns seemed optimal and comparable to LogCluster and Drain.

A notable problem that affected Drain, Logmine and Spell was the large number of patterns that was returned during the experiments. For example, in the case of LF2 data set, several algorithms generated more than 1,000 patterns (see *Total Clusters* in Table 2), with many of them being either too specific or having redundant nature. This issue is not specific to log pattern mining, but is a general data mining problem known since the 1990s [77, 76], and requires special techniques for post-processing detected patterns. LogCluster addresses this issue by implementing a number of post-processing techniques for eliminating too specific patterns and other redundancies [175]. As other tested algorithms did not include such features, this drawback will make them less suitable for mining patterns from large log files with many different message types, likely requiring a lengthy manual review of detected patterns from the end user.

Finally, the experiment demonstrated that LogCluster compares favourably to more recently released algorithms and is the fastest of the tested algorithms. Leaving few shortcomings aside, Drain also demonstrated reasonably good results and can be considered a viable alternative to LogCluster, especially when there is a need to analyse continuous log streams instead of complete log files. Both tools indicated their strengths in analysing security log files from cyber security exercises.

6 Covert data exfiltration and network anomaly detection

In this chapter, subsection 6.1 describes the research on advancing organisational network security by being able to discover covert channels used to exfiltrate sensitive data. This research along with details of the experiment and the release of two novel proof of concept data exfiltration tools were originally published in Publication IV.

Subsection 6.2 describes a novel NetFlow-based framework designed for detecting anomalous behaviour of end-user nodes within organisational computer networks. Detailed description of the framework was initially published in Publication X.

This chapter addresses research question RQ2.2. Thesis contribution 4 is described.

6.1 Covert channel data exfiltration detection

This subsection discusses the research on improving organisational network security regarding the discovery of data exfiltration channels. Although the first half of Publication IV focused heavily of exploiting various IPv6 transition mechanisms, within the context of this thesis, the primary focus lies in comparing the detection capability and IPv6 readiness of various Network Security Monitoring systems (NSM). The data exfiltration detection experiment involved extensive testing of five different NSM tools to detect 126 unique data exfiltration attempts. Apart from the detection aspect, this research resulted in the release of two novel proof of concept data exfiltration tools (*nc64* and *tun64*) that took advantage of IPv6 transition methods. Notably, these tools were later used within the NATO CCD COE CSXs. Particularly, the detection of the nc64 was integrated as a side challenge into a bigger digital forensic challenge of LS 2016. Within the 2016 XS exercise the tools were used to exfiltrate sensitive in-game data as part of the scenario.

At the time of conducting this research, the support for IPv6 protocol in many security solutions had not yet reached the level of acceptable maturity and readiness. Furthermore, the lack of practical experience resulted in IPv6 being often considered as a *backdoor* protocol which may allow an attacker to bypass security mechanisms. This notion is particularly important when an attacker already resides within the network perimeter. There is still a common misconception that malicious actors always originate from outside, thus NSM devices are commonly placed on the perimeter to detect and stop unauthorised access from the outside. Unfortunately this is not the case—in addition to insider threats (e.g., as described in [83]) who already operate within a privileged space, persistent and resourceful adversaries can often find alternative means (e.g., gaining physical access, infecting portable devices, planting a malicious USB stick, etc.) to bypass the outward-facing network security mechanisms. This research focused on measuring the ability of network sensors to detect tunnelling attempts originating from seemingly safe organisational networks.

Any kind of protocol tunnelling can pose a security risk, as it allows bypassing poorly configured or unsupported network security devices. IPv6 transition mechanisms, as well as more generic tunnelling techniques (e.g., HTTP, SSH, DNS, ICMP, IPsec), can often hinder detection or bypass network protection mechanisms. Furthermore, different tunnelling approaches can be employed to establish a covert channel by encapsulating exfiltrated information in various networking protocols. Covert communication channels based on DNS, HTTP(S), ICMP, and SSH protocol are the most common approaches for evading network detection mechanisms, due to both their widespread use and inclusion in standard network policy, which allows these outbound protocols and ports for end-user requirements and remote administration needs. For the NSM experiment test cases we considered fully developed and publicly available open-source tools.

Our research demonstrated that NSM solutions had several drawbacks in parsing and analysing IPv6 traffic. While there have been many improvements in handling IPv6 protocol traffic over the past few years, some of the shortcomings regarding inspecting multi-protocol channels are more fundamental and would require serious effort to redevelop the principles how NSM tools correlate disguised or seemingly separate network sessions to detect such malicious behaviour involving protocol switching, tunnelling or encapsulation. For instance, the *nc64* and *tun64* (the two IPv6 transition-based covert channel approaches) use both IPv4 and IPv6 implementations simultaneously making it harder to attribute them to the same covert channel and analyse them as a single session. In comparison, common tunnelling approaches (e.g., SSH, DNS, ICMP) were easier to detect by an automated monitoring solution or human analyst since their behaviour pattern is familiar and well understood.

Experiment scenario Testing environment and experiments were set up according to the following scenario. The cyberattack target was a small- to medium-sized research organisation that had a network of up to 100 nodes. This organisation assumed that it was running an IPv4-only core network, however, all their network hosts were dual-stack capable by default. With that assumption, the network administrators had implemented organisational network security policies only for the IPv4 protocol. The following common services and egress ports were allowed through the perimeter firewall: SSH (tcp/22), DNS (udp/53, tcp/53), HTTP (tcp/80), HTTPS (tcp/443), and ICMP (type 8—echo). When connecting to such ports, internal network nodes were able to establish a connection to the Internet without the use of proxies or any other connection handlers. The network administrators had unfortunately neglected that their ISP had just recently started to roll out IPv6 connectivity to customers.

Moreover, this organisation had been recently contracted by the government to conduct advanced technological research which involved storing and processing sensitive information on the organisation's workstations and servers. To verify the fulfilment correct security measures at the organisation, penetration testers (the red team) was tasked to assume the role of a reasonably sophisticated attacker with persistent foothold in the research organization's network. Their mission was to exfiltrate sensitive information from the target network without being detected. For the sake of the experiment, the red team had a wide selection of tools available at their disposal for establishing covert exfiltration channels.

Technical setup The testing environment consisted of six VMs. The network map of the experiment environment is presented in Figure 5. The organisation's network consisted of the compromised internal host and a perimeter router (*Router1*) connecting to the simulated ISP network. The attacker's C&C server resided in another network segment behind *Router2*. Two monitoring VMs provided detection capability. The first machine was connected with a virtual tap to the ISP network link between the routers, so all packets traversing this link were copied to its network capture interface and stored in a corresponding PCAP file. PCAP analysis and result collection was performed separately on a second monitoring machine.

Before setting up the detection environment, we worked together with the red team counterpart in this research to establish all potential exfiltration methods that we had to detect. The sensitive data that was to be exfiltrated in every attempt comprised the */etc/shadow* file and the root user's private SSH cryptographic key. Both files could potentially be used for gaining unauthorised access organisational assets.

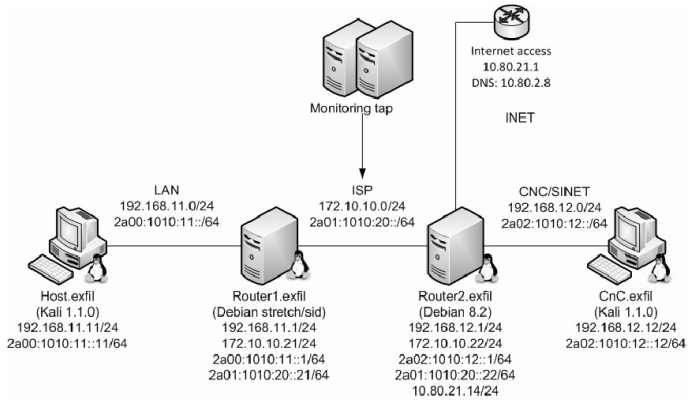

Figure 5: Network map of the data exfiltration experiment environment. [19]

We had to ensure fair and reproducible testing conditions for all tools, so each unique exfiltration attempt was captured as a PCAP file that was later analysed by every NSM tool. One idea was to mix standard port numbers for various protocols (e.g., establishing an SSH connection on port 80 or setting up a DNS tunnel on port 443) to see whether it would evade or hinder detection. Furthermore, the *nc64* and *tun64* tools employed several distinct operating modes (e.g., transmitting plain-text or base64 encoded payloads). With all combinations of exfiltration tools, operating modes, destination port numbers, transport layer protocols, and IP versions we ended up with 126 unique PCAP files. Each exfiltration attempt and PCAP file creation was automated by a set of scripts [80] which enabled to repeat the experiment multiple times while also avoiding potential human errors when handling such a diverse combination of exfiltration attempts.

The testing involved four open-source NSM tools that are often used in organisational environments—signature-based network IDS solutions Snort and Suricata, as well as network traffic analysers Bro[20] [160] and Moloch (now Arkime). Suricata was using the Emerging Threats Open ruleset (ET Open), while for Snort we compared the results with both SourceFire (SF) and ET Open rulesets.

Discussion of results The results of this experiment are summarised in Table 3. More detailed results are presented in an extensive table that can be found in the Appendix 1.A of Publication IV. Since the table spans over more than three pages, therefore only a brief summary is represented here. Each row in that table describes a unique data exfiltration attempt, while the columns represent a detection tool that was used to undertake its detection. Each detection attempt had four potential outcomes:

1. **Positive match** for an attack that was correctly classified as a malicious activity with appropriate alerts;
2. **Partial or abnormal footprint** which resulted in an alert, but the alert did not correctly describe the attack;
3. **Visual anomaly** which can potentially be detected by an expert human analyst inspecting logs or traffic patterns;
4. **Failed detection** when no alerts or specific connection logs were generated.

[20] Bro was renamed to Zeek in 2018

Table 3: Summarised results of the detection experiment.

Match type	**Snort SF**	**Snort ET**	**Suricata**	**Bro**	**Moloch**	**Sum**	**%**
Positive matches	0	61	61	0	0	122	19.4%
Partial matches	0	7	7	78	0	92	14.6%
Visual matches	0	0	7	48	62	117	18.6%
Failed detection	126	58	51	0	64	299	47.4%
Total iterations	*126*	*126*	*126*	*126*	*126*	*630*	*100%*

Firstly, from an attacker's point of view, we noted that in order to avoid detection data exfiltration tools using a particular application layer protocol should not reuse standard port numbers of other application layer protocols. For instance, an HTTP tunnel on TCP port 22 raised an *Outbound SSH Scan* alert with the ET Open signatures, whereas the same HTTP tunnel on port 80 only generated HTTP connection logs which are more likely to go unnoticed—as such, we classified this particular exfiltration attempt as being only *potentially visible*. However, the *Outbound SSH Scan* alert for the HTTP tunnel on port 22 was considered only a *partial match* because the traffic was incorrectly classified as outbound SSH connections. Notably, the same ET Open rule was again responsible for a *partial match* against the *nc64* tool on port 22.

Moreover, an alarm was triggered if SSH connection headers were detected on port 443, or if port 443 was used to send plain-text HTTP traffic. Similarly, if non-DNS traffic was detected on UDP port 53, the ET Open ruleset raised various alerts (e.g., for having *non-compliant traffic to DNS protocol*), or those connections being *overly aggressive* (i.e., generating too many connections). Surprisingly, most of these signatures were evaded when TCP port 53 was used.

Within the current experiment the selection of allowed outbound port numbers was rather limited, however, this is not the case for many less restricted organisational networks. As most server-side applications can be bound to listen on any applicable port number (e.g., an SSH server on TCP port 2022 or an HTTPS console on TCP port 8443) it is possible to evade or obscure detection by NSM tools even further while still staying within the relatively normal use of the particular application layer protocol.

The difference between SF and ET rulesets was critical for Snort. The SF ruleset seemed to primarily focus on inbound attacks on the network perimeter, and therefore could not identify any malicious outbound traffic in our tests. Note, that the poor results of Snort coupled with the SF ruleset negatively impacted the overall experiment statistics (see *Failed detection* rates in Table 3). Other tool and ruleset combinations resulted in significantly better detection rates. Moreover, there were also slight differences between Snort and Suricata when the ET Open ruleset was used. Most importantly, Snort clearly detected Ptunnel as the tool used for ICMP tunnelling.

Bro (now Zeek) does not use signatures like Snort or Suricata. In addition to its internal processors, custom scripts can be executed to inspect network traffic. By default, Bro was able to create protocol specific logs for all identified network connections. As such, it was able to generate log records of all exfiltration attempts. Although Bro does not generate alerts, it does have a specific log file called *weird.log* which lists detected anomalous connections. During the experiment several *weird.log* records were observed for *protocol non-compliant traffic* on port 53. Interestingly, Bro's SSH connection parser malfunctioned while processing non-SSH traffic on port 22. As a result, highly abnormal SSH logs were generated in the detection system (e.g., parts of HTTP traffic headers were parsed as SSH client and server identification strings).

Moloch does not provide traditional alerting functionality, however, its *viewer* component is a powerful network traffic search and visualisation tool. At the time of the experiment, Moloch did not support IPv6 due to various limitations in indexing 128-bit IP addresses in Elasticsearch prior to version 5.0. Taking this limitation into account, the IPv6-only channels were left completely unnoticed. Some dual-stack attacks still left a trace. For instance, when *tun64* was used with the *t6to4* mode so that IPv6 packets were encapsulated as the payload in IPv4 packets, thus making it *visible* in Moloch.

Notable considerations It must be acknowledged, that any sophisticated data exfiltration method will be hard to detect by existing network IDSs in real-time. Further complexity is added in situations where the data in question is split into smaller chunks and transmitted across different network sessions or protocols (in our case IPv4 and IPv6). To detect such malicious activity the NSM solution would need to correlate the session information in near real-time across various flows with different types of IP addresses. Although theoretically possible, it would incur a significant performance penalty for any such application. Distinguishing flows which belong together on busy network nodes with many simultaneous connections can be increasingly difficult.

Regardless, the research highlighted that by employing IPv6 tunnelling and dual-stack transition mechanisms in a particular way it is possible to evade detection of NSM solutions. This is something security operators have to be aware of. Considering the relatively high amount of *potential visual* matches, the experiment emphasises the shortcomings of traditional security monitoring solutions and clearly demonstrates the need for highly skilled operators in charge of those systems. Moreover, it illustrates the need for anomaly detection algorithms that can identify organisational network nodes with unusual traffic patterns. One such algorithm is presented in section 6.2.

6.1.1 Comparison with related work

The recent paper by Mazurczyk et al. in [101] specified a similar goal of testing the possibility of using IPv6 for covert channel data exfiltration. Furthermore, the research group assessed the covert channel detection capability of Suricata and Bro/Zeek. The experiments were set up using the hosting services of two major cloud service providers (AWS and Digital Ocean) in multiple geographical locations. Most importantly, the authors concluded that the hiding capacity of real network traffic in enterprise environments is actually less than what is suggested by experiments described in recent scientific literature. For example, many proposed covert channels work well in isolated lab setups, but when implemented in the wild, it turned out that the underlying network configuration of cloud providers simply rendered some of the covert channels unusable. The research also revealed that some of the detection mechanisms used by NSM tools cannot be considered effective to detect advanced covert channel techniques, especially when there is native IPv6 connectivity and no transitional mechanisms are required. Although the detection results were quite poor, the experiment indicated that Suricata outperformed Bro/Zeek.

The paper by Wendzel and Zander [181] discusses the use of protocol switching (e.g., between HTTP and DNS) to exfiltrate data over covert channels. While the general idea is similar to our approach, their research considers only IPv4 networks. Our *nc64* provides several additional features by allowing the user to also switch between IP protocol versions which in addition to other network parameters results in different source and destination IP addresses. The authors also proposed some novel detection methods which achieved up to 99% detection accuracy. Unfortunately, in practise the 1% false positive rate would still cause problems when analysing large amounts of network traffic.

6.2 Network anomaly detection

With the increasing complexity organisational networks and the sophistication of adversarial activity, the use of machine learning for advancing network security has become an important research problem. As discussed in the previous section 6.1, traditional rule-based technologies such as firewalls and network IDS/IPS solutions are only able to detect previously known and sufficiently described attacks, while advanced attack techniques that do not match any existing signatures often remain unnoticed.

This section describes a novel NetFlow-based framework for detecting anomalous end user nodes and their network usage patterns in organisational networks. The framework is focusing on end user nodes, since they are often targeted by malware, APTs and other threats. The unsupervised framework uses three distinct anomaly detectors for calculating a combined anomaly score for each end user node in hourly time windows. The framework is executed once every 60 minutes to process the network flows from the past hour for each end user node. If a particular node has no flows in that timeframe, it is considered inactive and will be skipped. Once an anomalous node has been identified, LogCluster algorithm is used for finding network traffic patterns by analysing NetFlow data. Figure 6 provides a general overview of the framework and its components.

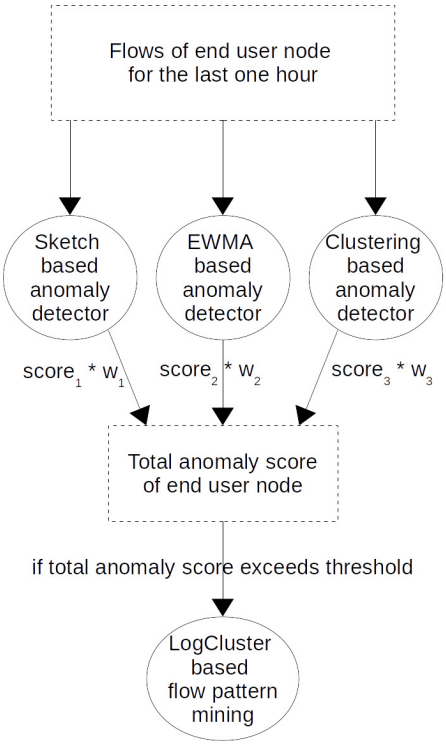

Figure 6: Overview of the anomaly detection framework. [176]

6.2.1 Ensemble of anomaly detectors

The framework comprises three anomaly detectors that calculate anomaly scores $score_1$, $score_2$ and $score_3$ within the range 0..1 for each node. Furthermore, the three anomaly detectors are assigned corresponding non-negative weights w_1, w_2 and w_3, whereas:

$$w_1 + w_2 + w_3 = 1$$

The total anomaly score of the end user node is calculated as follows:

$$score_1 * w_1 + score_2 * w_2 + score_3 * w_3$$

Evidently, the total anomaly score can range from 0..1. If the total anomaly score for a particular node exceeds a predefined threshold (we used a threshold of 0.5 in our experiments), an alert is generated and network flows related to that node from the past hour are processed using the LogCluster algorithm. Identified network traffic patterns are presented to the operator for further inspection. The following paragraphs provide a brief overview of each anomaly detection method that is employed in the framework.

EWMA-based anomaly detector The primary purpose of the EWMA-based (Exponentially Weighted Moving Average) anomaly detector is to detect unexpected increases for a number of monitored features. For each end user node *E*, the anomaly detector keeps track of the following features in 1-hour time increments:

- Peers: Nº of unique peers[21];
- RarePeers: Nº of new or rarely connected peers, *rare* meaning that node *E* has not communicated with a particular peer within the last *N* hours;
- PeerPorts: Nº of unique peer ports[22];
- LogFlows: $\log_{10} M$, where *M* denotes the Nº of flows;
- LogPackets: $\log_{10} M$, where *M* denotes the total Nº of packets exchanged;
- LogBytes: $\log_{10} M$, where *M* denotes the total Nº of bytes exchanged.

The first three features employ a simple unique count of peers and ports, however, the values of the other three network-specific characteristics (the number of flows, packets, and bytes) have been converted to logarithmic scale to reduce the number of false positives due to large variations of value *M*.

The feature *RarePeers* identifies unanticipated rise in the number of new or very rarely connected peers, even if the total number of peers remains within predicted limits. For our experiments, we set the value of *N* to 50 hours. A total of six features are tracked by the EWMA-based anomaly detector and each feature contributes to the reported anomaly score equally. For instance, if for node *E* alerts have been triggered for *PeerPorts* and *LogPackets* features, anomaly score $\frac{1}{3}$ will be reported for node *E*.

Clustering-based anomaly detector The main purpose of the clustering-based anomaly detector within the framework is to find collections of similarly behaving nodes and declare outliers as anomalous. The EWMA-based anomaly detector alone is not aware of the surrounding knowledge which may influence the level of criticality of detected anomalies. For instance, when a single node is observed downloading an unusually large amount of data it is considered anomalous, however, if many hosts suddenly exhibit similar behaviour it might be explained by scheduled updates or centralised patching of all devices.

[21] In this context, *peer* is defined as a 3-tuple containing *transport protocol ID*, *IP address of the remote node*, and *port number at the remote node*.

[22] *Peer port* is defined as a 2-tuple containing *transport protocol ID* and *port number at the remote node*.

We have employed the DBSCAN algorithm [41] for clustering the network nodes. For that purpose, each node *E* is represented by a 10-dimensional vector that comprises the hourly values of the following features:

1. Peers;
2. RarePeers;
3. PeerPorts;
4. total bytes;
5. total packets;
6. total flows;
7. outgoing bytes;
8. outgoing packets;
9. incoming bytes;
10. incoming packets.

The first three features (*Peers*, *RarePeers*, and *PeerPorts*) are defined the same as for the EWMA-based anomaly detector. The other seven features again describe a collection of network-specific characteristics of node. Note the different groups of incoming, outgoing, and total values.

Nodes that belong to a detected cluster have an anomaly score of 0. The anomaly score for *outlier* nodes depends on the number of detected outliers. Specifically, if *k* outliers are detected among a total of *n* nodes, an anomaly score of 1-($\frac{k}{n}$) is returned for each outlier node. Therefore, fewer outlier nodes will yield anomaly scores close to 1, while a large number of outliers (i.e., when anomalous behaviour among monitored nodes is more common) the anomaly scores will decrease.

Sketch-based anomaly detector The purpose of the sketch-based anomaly detector is to predict the future behaviour of monitored nodes. The sketch-based anomaly detector is our novel contribution which uses one-row sketches for compiling past communication patterns of individual end user nodes and employs those patterns for predicting future usage patterns. The sketch-based detector has three anomaly detection components:

- Similarity and entropy of peers;
- Similarity and entropy of peer ports;
- Entropy of node's local ports.

Again, each anomaly detection component contributes $\frac{1}{3}$ of the total anomaly score reported by the detector. The operation of the anomaly detector starts out with two sketches *S* and *V*, which are vectors of *n* counters. In our environment we set `n=100000` for peers, and `n=10000` for the values describing ports. The aim of vector *S* is to calculate recent average communication patterns of node *E*, while vector *V* captures this information for the last hour.

To provide an example how vector V is built for peers—for each flow, a peer is extracted and hashed into a value from range *1..n*, and if the value is *k*, the k^{th} counter in vector *V* is incremented (therefore, each counter in vector *V* reflects the number of flows for the

group of one or more peers). Vectors *V* for peer ports and local ports are built in a similar fashion.

Subsequently, each hourly calculated vector *V* is used for updating the average patterns in vector *S* in EWMA-based fashion. Since according to our findings end user nodes tend to follow recent communication patterns with peers and peer ports, vector *S* serves as a good predictor of the future behaviour of the monitored nodes. Thus, any significant difference between vector *S* and the newly calculated vector *V* can be considered an anomaly. Note, that our framework calculates the difference between *S* and *V* *before* *S* is updated with *V*.

During the development of the anomaly detector component several distance functions were tested, however, experiments revealed that cosine similarity produced the best results. Cosine similarity function (*cosim*) measures the angle between two vectors. Since the counter values of *S* and *V* are non-negative, the similarity value ranges from 0 to 1. Values closer to 1 indicate that the distribution of flows over peer (or port) groups during the last 1 hour is similar to the distribution of past measurements. Conversely, smaller values indicate larger deviations from prior observations. For instance, `cosim(S,V) = 1` if vectors *S* and *V* have the same direction (e.g., in case of vectors (2,4,23) and (4,8,46)), and `cosim(S,V) = 0`, if the vectors are orthogonal.

Our research indicated that organisational end user nodes mostly communicate with relatively small number of peers which means that counters of vector *S* rarely encounter large values, while many values often remain zero. That is because end user nodes tend to belong to a single user with clear network usage patterns. On the other hand, if the behaviour of nodes is changing frequently and has no clear baseline (e.g., in computer classrooms where machines are shared between many users with different usage habits), the similarity between vectors *S* and *V* remains low and becomes meaningless for detecting anomalous behaviour. To counter unwanted alerts in such situations, the anomaly detector performs additional checks for disabling similarity-based anomaly detection for these nodes.

Furthermore, the anomaly detector uses entropy-based anomaly detection for vector *S* to discover nodes with frequently changing behaviour. In previous research, entropy-based techniques have been successfully used for detecting such situations [182]. The entropy of *S* is calculated after each update of vector *S*. The anomaly detector raises an alarm if the calculated entropy is higher than the defined threshold of T_{ent}. In our experiment, we set T_{ent} to 0.5 and 0.3 for peers and peer ports respectively.

Finally, end user nodes typically use ephemeral (*short-lived*) local port numbers for establishing outgoing connections with remote services. The anomaly detector maintains a vector *S* for local ports of each node *E*. In case of ephemeral port numbers, vector *S* is expected to have a high entropy. As such, the anomaly detector raises an alarm if the entropy falls below a predefined threshold of T_{ent2}. In our experiments, we set $T_{ent2} = 0.2$.

6.2.2 Flow pattern mining with LogCluster

If a node's combined anomaly score from the three anomaly detectors for last hour exceeds a predefined threshold (set to 0.5 during our experiments), NetFlow records of the preceding 1 hour for this node are mined with the LogCluster algorithm. This procedure enables the detection of prominent traffic patterns that likely capture the nature of the reported anomaly. Since LogCluster is designed for mining textual events, each binary NetFlow record is transformed into a textual format with keyword-value pairs. This conversion can be performed using the advanced Perl-based input pre-processing functionality of LogCluster.

The resulting textual line-based flow description and two example output line patterns are outlined in Listing 13. Example #1 depicts DNS requests from node 10.3.7.22 to server 192.168.1.1 port 53/UDP. Furthermore, similarly to prior LogCluster usage example in Listing 7 we configured LogCluster to use advanced heuristics for joining flow patterns based on word weights. Example #2 demonstrates the case where two similar patterns for DNS queries from node 10.1.1.1 to two DNS servers 192.168.1.1 and 192.168.1.2 have been joined together. Note, that the *tcpflags <flagstring>* values are *NA* because both examples describe UDP traffic. In case of TCP traffic, the potential values for *flagstring* are *invalid-flags*, *with-ACK*, and *without-ACK*.

Moreover, as discussed previously in section 5, determining a good support threshold value depends highly on the type and nature of input data. Our experiments indicated that in case of NetFlow data, the support threshold value of $\sqrt{n}$ (where *n* denotes the number of flow records for node *E*) highlights the nature of anomaly adequately in most cases.

Listing 13: Input line format along with LogCluster output examples.

```
# Input line format:
# proto <transport protocol ID> srcip <source IP address> srcport <source port> dstip
# <destination IP address> dstport <destination port> tcpflags <flagstring>

# Example #1:
proto 17 srcip 10.3.7.22 srcport *{1,1} dstip 192.168.1.1 dstport 53 tcpflags NA

# Example #2:
proto 17 srcip 10.1.1.1 srcport *{1,1} dstip (192.168.1.1|192.168.1.2) dstport 53
tcpflags NA.
```

6.2.3 Evaluation

We conducted a five-month experiment to assess our framework's performance in the office network of an academic institution. The network consisted of over 200 workstations and laptops with various operating system platforms (e.g., Windows, Linux, MacOS). As described above, the framework was designed for detecting anomalous end user nodes, so server nodes were considered out of scope.

The anomaly detection framework employed softflowd NetFlow exporter [67] on a dedicated Linux server for monitoring the network traffic of the entire network segment without data sampling. The exported flow data was collected with nfdump NetFlow collector [50].

To reiterate some of the configuration items mentioned above—each anomaly detector was assigned an equal weight (i.e., $\frac{1}{3}$) and the anomaly score alarm threshold was set to 0.5. During the five months, the system raised 1026 alarms for 33 hosts. However, over 90% of those alarms were generated by four highly verbose hosts, and for 25 nodes only 1–3 alarms were raised. Listing 14 depicts some anomalous traffic patterns the framework discovered with LogCluster.

Detailed analysis of the alarms revealed that 638 (62.2%) alarms were generated for just five hosts which were using a BitTorrent client—an activity that is not allowed by organisational policies. Furthermore, 327 alarms (31.9%) were triggered by a special Internet Measurement probe which was erroneously connected to the office network segment. Despite its minor network footprint of up to 100Kbit/s, the anomaly detection framework was able to flag the probe as anomalous. Moreover, the framework generated a total of 46 false positive alarms (4.5%), with most of them being triggered by legitimate downloads of large files.

Listing 14: Discovered flow patterns for anomalous network traffic. [176]

```
# Patterns detected for an Internet Measurement probe 10.1.1.1. The first pattern
# reflects ICMP echo (type 8, code 0) packets sent to a large number of public hosts,
# while the second pattern reflects ICMP response packets from those hosts: ICMP
# echo reply (type 0, code 0), ICMP network unreachable (type 3, code 0), ICMP TTL
# expired in transit (type 11, code 0).

proto 1 srcip 10.1.1.1 srcport 0 dstip *{1,1} dstport 8.0 tcpflags NA

proto 1 srcip *{1,1} srcport 0 dstip 10.1.1.1 dstport (11.0|0.0|3.0) tcpflags NA


# Patterns detected for host 10.1.1.7 that ran a BitTorrent client at a known
# BitTorrent port 8999/udp, and exchanged data with other known remote BitTorrent
# ports 6881/udp and 51413/udp.

proto 17 srcip 10.1.1.7 srcport 8999 dstip *{1,1} dstport (51413|6881) tcpflags NA

proto 17 srcip *{1,1} srcport (6881|51413) dstip 10.1.1.7 dstport 8999 tcpflags NA


# Patterns detected for host 10.1.1.12 that executed a TCP SYN scan of the host
# 192.168.16.250 with nmap -sS
# Note that all TCP SYN packets were sent from a single port 44290/tcp.

proto 6 srcip 10.1.1.12 srcport 44290
            dstip 192.168.16.250 dstport *{1,1} tcpflags without-ACK

proto 6 srcip 192.168.16.250 srcport *{1,1}
            dstip 10.1.1.12 dstport 44290 tcpflags with-ACK
```

To evaluate the precision and recall of the anomaly detection framework, a controlled network scanning experiment with the *nmap* tool [95] was executed from one of the workstations against test targets for the duration of 15 hours. Malicious activity scenarios involved various types of scans against a single target and a full network scan involving 2048 hosts. Altogether, this workstation node was online for 548 hours during the entire five-month timeframe. The duration of the scanning experiment (15 hours) was correctly flagged as anomalous. No other malicious network activity was observed for this node during remaining 533 hours. Furthermore, as no false alarms were raised during the entire five-month timeframe, our framework demonstrated the precision and recall of 100%.

6.2.4 Comparison with related work

Similarities can be drawn with several publications dealing with network anomaly detection. During the last two decades, most research on network anomaly detection has not focused on end user nodes in organisational networks. This section will discuss recent methods which can be applied in the scope of organisational networks to specifically identify anomalous end user nodes.

Kind et al. have described a histogram-based network anomaly detection solution in their paper [74]. Their proposed solution employs a supervised method which involves generating detailed histograms of IP protocol header information. These histograms are clustered to build models of common network behaviour. Vectors that encode the network behaviour of nodes are compiled and for anomaly detection the distance of the vectors from the detected clusters is calculated. Unlike the work by Kind et al., our method is unsupervised and does not require retraining when the nature of the network traffic changes—creating training data sets for supervised algorithms is time consuming and requires input from human experts, and is thus an expensive process.

In a previous paper [169] Vaarandi proposed two distinct anomaly detection algorithms for analysing NetFlow data. The first algorithm maintains a network service usage profile for each node. The detection works near real time and raises an alarm if the node connects to unusual services. The second algorithm performs daily clustering of nodes based on behavioural similarities. The algorithm raises an alarm if a node's behaviour deviates from the rest of the group. According to the author, the algorithm is not just capable of detecting suspicious or unusual activity, but also inactivity compared to its peers. For instance, an alarm was raised if a node had stopped sending logs to the central log server or stopped synchronising its time with the NTP server. The main advantage of our method described in this thesis is its ability to detect situations when an end user node exchanges an abnormally large amount of data with its peers.

In [48] Grill et al. presented an anomaly detection method that uses a statistical approach of analysing NetFlow data to detect nodes compromised with malware types that employ a DGA for C&C communication. The algorithm calculates the mean and variance of the ratio between the number of DNS queries and the number of contacted IPs for every host in the monitored network. Deviations from the mean value are labelled as anomalous DNS queries that typically indicates an infection with DGA-based malware. In contrast, our method does not just focus on detecting a specific type of malware, but identifying anomalous behaviour in general, such as network scanning and prohibited file sharing activity.

In [182] Zhou et al. present a visualisation solution for addressing some of the shortcomings of entropy-based anomaly detection methods. Their ENTVis tool is designed to provide a more coherent visual analysis experience that makes analysing entropy-based NetFlow anomaly detection features more intuitive for users. This approach allows human analysts to spot alerted anomalies and understand the cause. Although our method also uses an entropy-based technique in one of its anomaly detectors, our method employs additional classifiers which allow for the detection of much wider range of anomalies (e.g., communications with unexpected peers and transfers of large data volumes over the network), making it more suitable as a generic anomaly detection algorithm.

7 Towards autonomous cyber defence

This chapter presents research that serves as a future outlook of potential developments in cyber defence. These developments focus on advancing cyber defences towards autonomy by proposing a concept and a reference architecture for intelligent cyber agents capable of performing self-determined defensive actions within a given scope (e.g., system or device). Our work refers to such agents as Autonomous Intelligent Cyber-defence Agents (AICAs). This research is based on the work of the NATO Science and Technology Organization's [27] Research Task Group (RTG) IST-152 [120], where the author of this thesis was an active and contributing member.

This chapter addresses research questions RQ3.1–RQ3.3. The chapter summarises Publications VII, VIII and IX. Thesis contribution 8 is described. Additionally, the final report [86] of the IST-152 group is the primary basis of these publications and includes the detailed description of the proposed reference architecture for AICAs.

Subsection 7.1 provides an overall description of AICA, the reference architecture, and motivation for developing such agents. Subsection 7.2 describes the particular contributions of the thesis author, i.e., specification of the *Sensing and World State Identification* (WSI) function and development of the Agentfly UAV use case to illustrate the role of AICA in a UAV.

7.1 Autonomous intelligent cyber-defence agents

This research has been driven by the evolution of advanced interconnected military systems, which are relying increasingly on new software and hardware technologies. To effectively defend such systems from cyberattacks an autonomous intelligent cyber defence system is required.

To clarify the most significant term, *autonomy*, we proposed the following definition:

> 'Autonomy is the capacity of systems to decide by themselves on their course of action in uncertain and challenging environments without the help of human operators.' [162]

Autonomy should not be confused with automation, i.e., performing a set tasks according to a collection of pre-defined rules in known environments where the level of uncertainty is low. While full and continuous autonomy is not necessarily the objective, AICAs will need to be able to cope with operational challenges such as making rapid decisions, analysing high volumes of data, handling intermittent communications, and remaining persistent in adverse situations.

AICAs are meant to autonomously and reliably handle cyberattacks affecting the system they defend. Furthermore, the agents can communicate with one another, a remote C&C system, or even a human operator when required and technically possible.

7.1.1 Rationale of AICA

Although military systems are generally resilient, the increasing number of high-tech features and interconnections that they hold make cyberattacks an attractive way for hindering their functions and the missions they are involved in. Even if AICA's IDS component reported any malicious activity, it might be too late to take manual action, thus these systems must be capable of taking autonomous defensive actions themselves.

Furthermore, on the battlefield there is always a scarcity of available cyber competencies. Deployed operational forces are often not specialists of cyber security nor cyber defence. Additionally, military systems are often deployed in remote areas with limited

connectivity and bandwidth, making any communication back to the C&C element even more difficult. For example, a well-planned automated attack could be carried out in a matter of seconds, and quite likely before any human operator can take action. Thus, the defence of ICT components in such complex systems should be relieved from any unqualified operators.

Although the IST-152 RTG focused primarily on military applications of AICAs, it was generally agreed within the group that AICAs could be deployed in civilian systems in a similar manner. For instance, complex interconnected systems or the wide-scale deployment of IoT devices in non-military domains are facing similar cyber defence concerns. Requirements to adapt with frequent reconfigurations and emerging circumstances may soon render classic centralised monitoring techniques and big SOCs infeasible for cyber defence. However, locally implemented distributed swarms of intelligent cyber defence agents can monitor and defend such fuzzy networks. Furthermore, AICA's capabilities will include learning from prior experience to improve the accuracy of its future decisions.

Potential cooperation between multiple agents is achieved through available communication channels. Agents must be as stealthy as possible to protect themselves from attacks by the adversary. Therefore, external communications must be kept as covert as possible given the agents' current security posture. However, this policy can limit the agent's capability to receive external orders, which may substantially diminish its usability.

As autonomous agents, AICAs should resolve most cyberattacks without interrupting end users and necessary system functionality. Naturally, agents should collaborate with human operators when they reach the limits of their capabilities, i.e., not understanding the current situation, needing to elaborate on defensive countermeasures, or when facing a critical error. However, such cases should remain minimal.

The balance of autonomy and human collaboration should be adapted based on the application and criticality of a particular system. For instance, an AICA responsible for defending a live weapon system might be more constrained in terms of autonomous decisions compared to AICAs defending a typical computer network. There is a vast difference between whether an AICAs false-positive decision would result in a weapon system to be fired or someone's workstation getting locked down.

Implementing AICAs in practice will not be a simple task. Constructing the required competencies, functions and technology will definitely be challenging. Some challenges, for example, might include working in resource-constrained environments, managing integration into host hardware, enabling the agent to learn and improve its own decision-making process, ensure agents' resilience, autonomously generating and executing plans of countermeasures in case of an attack, assuring its safety towards friendly systems, etc.

AICAs are implemented within or attached to one delimited system or device. Its primary objective is to ensure the availability and integrity of all relevant computerised functions in case of malicious attacks. Detecting abnormal behaviour of the physical platform (e.g., physical tampering) is not within the direct scope of the agent but may be detected by sensors in certain situations.

7.1.2 AICA Reference Architecture

The IST-152 RTG based the AICA reference architecture on the classical model proposed by Russel and Norvig in [146]. The architecture along with the functional components are illustrated in Figure 7. The functional components are divided into three primary classes based on their particular role—see Table 4 for the classification.

Based on the relevance of various components, the analysis proposed a selection of five high-level functions necessary for the cyber defence of military systems.

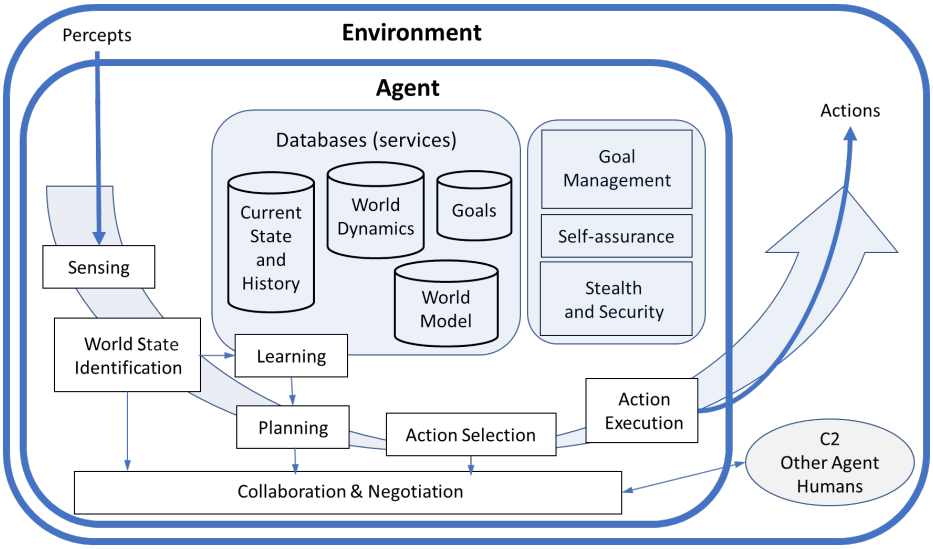

Figure 7: Proposed AICA's architecture and primary functions. [86]

Table 4: Classification of AICA components. [86]

Core components	Supporting functions	Data services
Sensing	Collaboration and negotiation	World model
WSI	Learning	Current state and history
Planning	Goals management	World dynamics
Action selection	Self-assurance	Goals
Action execution	Stealth and security	

1. **Sensing and WSI**: enables cyber-defence agents to gather data from the surrounding environment, systems in which it operates, and from itself. Any detected risks trigger the *Planning and Action Selection* function.
2. **Planning and Action Selection**: enables cyber-defence agents to specify one or more action proposals and propose them to the *action selection* subfunction, which decides the action or set of actions to execute.
3. **Action Execution**: enables cyber-defence agents to execute action(s) assigned by the previous step, as well as monitor the execution process and its effects.
4. **Collaboration and Negotiation**: enables cyber-defence agents to exchange information with other agents, a central C&C, or with a human operator.
5. **Learning**: enables cyber-defence agents to employ their prior experience to continuously improve their future decisions and efficiency.

7.2 Author's contributions

Chapter 7 summarises the results of a multi-year research effort by the NATO IST-152 RTG. Therefore, with four relevant publications, multiple authors, and the final report [86] of over 150 pages, it is essential to pinpoint the contributions made by the thesis author. The following subsections describe the two primary tasks the thesis author was working on.

7.2.1 Agent's sensing and World State Identification

To interact with the external environment, AICA has to perceive and understand itself and its surroundings. *Sensing* and *WSI* components are crucial for establishing agent's situation awareness. Achieving this first high-level functional requirement provides the necessary input for other functions down the line.

Sensing *Sensing* is AICAs first high-level function that is responsible for collecting data from both external (i.e., the defended system, external environment) and internal sources (i.e., the agent itself). Figure 8 depicts the primary elements of the *Sensing* component and how it relates to other AICA components.

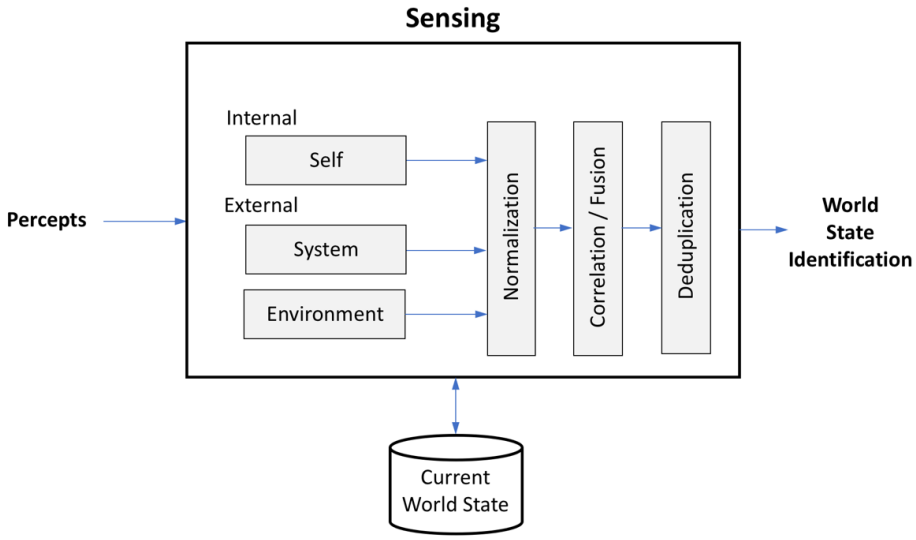

Figure 8: Overview of the AICA Sensing component. [86]

To ensure agent's own integrity and continuous operation, the *Sensing* function monitors itself by analysing runtime statistics and logs. The input modules of the *Sensing* function can largely be considered as typical system and network monitoring tools which collect logs and metrics from the agent, the defended system, and relevant functions running on the defended host. The *Sensing* component is also capable of capturing and analysing the protocol-specific network communications traversing the host network interfaces. To assure that the sensory data is readily usable by the rest of the system, it should be properly normalised, correlated, and deduplicated, so that only unique and relevant information is passed on. Moreover, any input data always goes through the input sanitation process. This also applies to data sent by other AICAs and C&C, because attackers may attempt to inject malicious code or garbage data into the agent.

World State Identification The *WSI* function processes and interprets the data from the *Sensing* component. The *WSI* function determines the current state of the world (i.e., the surrounding environment) and of the agent itself. This state is represented by an abstract model of reality that establishes a semantic meaning to the data perceived by the agents. *WSI* function detects any changes in this state and logs observations of any adversarial or suspicious events. Finally, the collected data is routinely analysed for anomalies to detect attack scenarios taking place over a longer period of time.

The WSI comprises four consecutive processes:

1. **Environment identification**: Determine the properties of the current operating environment based on both the *Sensing* information and the expected state knowledge. Any irrational input or unexpected changes (e.g., detection of virtualisation, adversarial debugging, or reverse engineering) can trigger a change to a heightened security posture.
2. **Friend-or-foe identification**: Tagging of processes, files and any network connections on the system.
3. **Anomaly identification**: Anomaly detection in the Sensing function data. The detection can be a selection or combination of rule-based, pattern-based, or behaviour-based detection. The anomaly detection baseline is stored in the current world state.
4. **WSI update**: Transforms the data from the previous steps (i.e., environment, friend-or-foe, and anomaly identification) into an update of the World Model and World Dynamics databases.

The processes along with supporting data sources are illustrated in Figure 9. The *World Model*, the *Current State* and the *World Dynamics* databases are used to establish and store a baseline of incoming data feeds. Additionally, these databases also include a limited set of historic data to detect unexpected changes and anomalous behaviour.

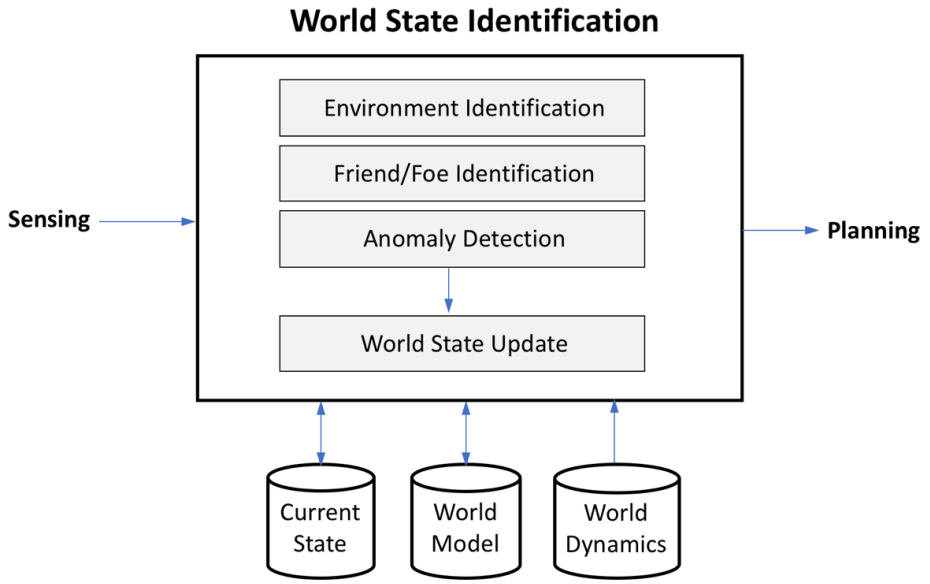

Figure 9: Overview of the AICA World State Identification component. [86]

The exact components of the *world models* are highly dependent on the particular agent's design and implementation. However, given the required operational parameters of AICA, it is expected that the model should minimally contain the components listed in Table 5. The components and data requirements strongly overlap with some of the previously described systems (e.g., Frankenstack). The *World Dynamics* knowledge updates must be computed during run-time based on the *World Model* and the *Current State* data.

Table 5: Components of the World and Current State models. ([86])

Component	Relevant model	Description
Flow database	Current State	Network flow records; full traffic capture (when feasible)
Log stash	Current State	Collection of all available logs; indexed for quick searching and analysis
System metrics	Current State	Performance and operational metrics of the agent and host system
Behaviour baselines	World Model	Predetermined policies and baselines of normal agent and host behaviour
Entity description	Current State & World Model	Technical properties of assets in AICA's proximity

7.2.2 Use case of AICA

It is evident that the type of the defended host system will have different operational impacts on AICAs designed to protect them. An AICA responsible for defending the computerised components of a static system might be significantly easier to engineer compared to an AICA that has to be mounted to a military vehicle's (e.g., ground, aerial, surface, subsurface) computational unit. Additional distinction to consider is based on type of the control system that the system has, e.g., manned, optionally manned, or unmanned systems. This distinction is important because in some cases (e.g., if the agent loses control or fails), it may be possible to fall back to manual control by the on-site human operator—something which is not applicable to unmanned systems.

The AICA reference architecture aims to describe the common characteristics applicable to every system. Therefore, the IST-152 RTG attempted to generalise the reference architecture as much as possible. However, to better explain how an AICA might perform in practice the thesis author developed a hypothetical military UAV use case, which describes the possible deployment of AICAs to the AgentFly project developed by the AgentFly Technologies [2].

Figure 10 illustrates the relevant components within the UAV example. Highlighted elements in the figure are within the scope of AICA's defence responsibilities. *Computing power* means the computational units that are required to operate the UAV. *Actuators* are devices used controlling the physical elements (e.g., wheels, wings, flaps) of the UAV. *Actuators*, *sensors* and *communication module* are assumed to include computer processing and can thus be targets of cyberattacks.

Note, that the following paragraphs until the end of this subsection are a close representation of the use case published in Publication VIII. Minor changes have been applied to synchronise terminology used throughout this thesis and improve readability.

> The AgentFly project facilitates the simulation of multi agent Unmanned Aerial Vehicles (UAV). Its features include flight path planning, decentralised collision avoidance and models of UAVs, physical capabilities, and environmental conditions [152]. In addition to simulations, AgentFly was implemented on a real fixed-wing Procerus UAV [133].
>
> This use case is based on the set of missions described within the AgentFly project. They are extended to include an adversarial cyberattack activity against the AgentFly UAV to disrupt its mission. The use case is that a swarm of AgentFly UAVs perform a routine tactical aerial surveillance mission

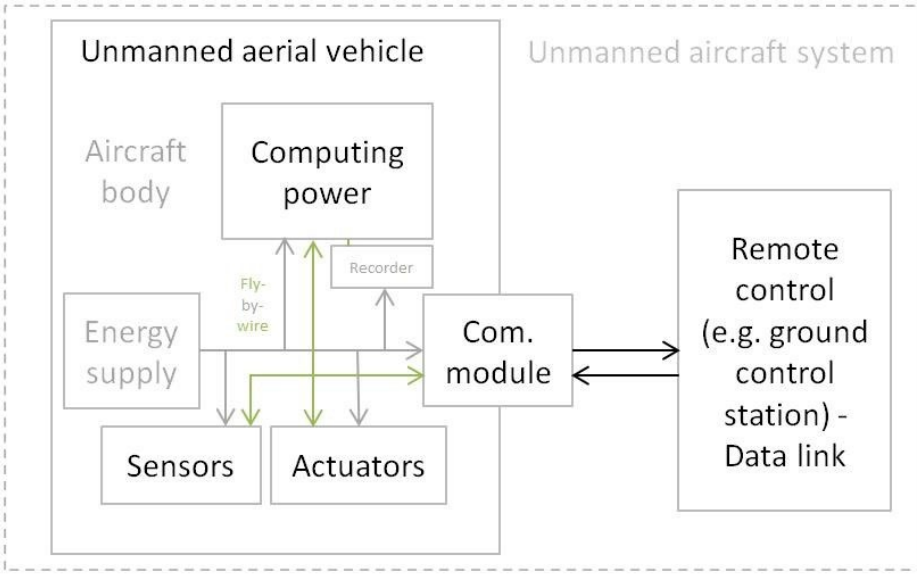

Figure 10: Potential scope of an AICA illustrated in the context of a hypothetical UAV. [86]

in an urban area. Typical collaboration between AgentFly UAVs aims at collision avoidance, trajectory planning, automatic distributed load-balancing, and mission assurance. The following use case follows the perspective of a single AICA agent, but this basic use case could be expanded to multiple AICA agents deployed across the swarm of AgentFly UAVs.

The AgentFly UAVs use case is built around the following assumptions:

- AgentFly UAVs self-assess and share information with neighbouring UAVs.
- When setting up a communication channel, AgentFly UAVs have to determine whether they trust their correspondent.
- Network-wide collaboration and negotiation is affected by reachability, range, and timing issues.
- The AgentFly UAV lacks modern cyber defence capabilities and is thus vulnerable to potential cyberattacks.
- Due to environmental conditions, AgentFly UAVs might be offline for some time and later re-join the swarm when connectivity allows.
- A single AICA agent is implemented within each AgentFly UAV.
- The AICA connects with the modules of the UAV and can supervise the activity and signals in and between various UAV modules (e.g., sensors, navigation, and actuators).
- The AICA can function in isolation from other AICA agents present in the AgentFly UAV swarm.

Attackers have acquired a technology similar to that used in AgentFly UAVs' COMMS module. They have discovered a zero-day vulnerability that can be exploited remotely over the radio link from the ground, and they plan to use the vulnerability to gain control over the swarm of UAVs and cut them off

from the theatre's C&C system. The UAVs are using the COMMS module to collaborate among themselves and report to the C&C when needed.

The vulnerability lies in the functionality responsible for dynamically registering new UAV agents in the swarm upon due request. The COMMS module is interconnected with other intrinsic modules of the AgentFly UAV via a central control unit. The adversary has set up a ground station in the area of the surveillance mission. When AgentFly UAVs enter the area, the cyberattack is launched. AICA detects a connection to the COMMS module and allows the incoming connection for the dynamic registration of a new UAV agent into the swarm. Due to the nature of zero-day attacks, an IDS module would not have any corresponding signature or IoC to detect the compromised payload.

AICA's *Sensing* function monitors the entire set of modules of the AgentFly UAV. The AICA's *WSI* component flags the connection from the newly connected UAV agent as anomalous since it does not follow the baseline pattern that has been established based on previous connections with legitimate UAVs. It also detects a change in the UAV's system configuration and deems it anomalous because no new configurations have been received from the C&C. Using its *Sensing* function, AICA launches a full system integrity check. A compromise within the UAV's COMMS module is detected.

AICA decides (within its *Planning* and *Action Selection* modules) to isolate (using the *Action Execution* module) the COMMS module from other UAV modules in order to prevent further propagation of the attack. Alerting the C&C is not possible because of the compromised COMMS module.

In order to reduce the attack surface (decided in the *Planning* and *Action Selection* functions), the AICA requests (*Action Execution* module) that the UAV's central control unit resets the COMMS module, raises the security level and disables auxiliary functions (i.a., the dynamic inclusion of new UAVs into the swarm).

After the reset of the COMMS module, AICA attempts to recover communications functionality (*Sensing* function). AICA performs another integrity check to verify that no other compromise exists. It keeps its *Sensing* and *WSI* components on a high-level of vigilance in relation to integrity monitoring. AICA adds the signature of the payload that caused the anomaly into its IoC knowledge base (*WSI* function). Furthermore, it sends out an alert along with malware signature updates to other agents as well as to the C&C.

This concludes the use case description. While the use case might be slightly simplistic, it nevertheless provides a generic concept of how AICA should perform in relation to supervising and defending an existing interconnected unmanned system.

7.3 Conclusion

Foreseeable evolution of military systems suggests that AICA agents will be required to effectively defend complex systems. Furthermore, it is likely that civil systems, e.g., the wide-scale deployment of the IoT technology, will soon generate similar demands. The proposed AICA reference architecture is a response to these requirements, aiming to assure a common structure, defence efficacy and of the agents.

Since the development of the AICA reference architecture has been an arduous endeavour, the number of people involved with the project is substantial. The current chapter first outlined the relevance of autonomy in cyber defence and then described the

generic AICA architecture. Secondly, the chapter summarised specific contributions, i.e., the *Sensing* and *World State Identification* functions, as well as the example use case to describe the functions of an AICA within a UAV.

The AICA research turned out to be a prominent activity which gathered substantial interest from both industry and academia. The official NATO IST-152 RTG activities concluded with the publication of the final report [86], however, the group decided to pursue a continuation. This endeavour has already resulted in a two-day workshop [12] which was organised in cooperation with NATO NCIA and a new AICA-themed academic conference [13] held in spring 2021.

8 Thesis conclusions and future work discussion

8.1 Summary and conclusions

The thesis describes the research and methods for automating defences against adversarial cyber operations in computer networks. Additionally, this thesis aims to bridge the gap between theoretical research and practical guidance usable for cyber security practitioners. The difficulty of applying theoretical research outcomes in practice has been previously criticised in related publications by several authors. To alleviate this issue, the thesis and the accompanying publications provide plenty of valuable practical guidance and implementation examples.

The related work discussion in chapter 2 follows the overall structure of the thesis. Furthermore, most thesis chapters also include a dedicated subsection that describes some of the key similarities and differences compared to related work from other authors, including the works that have been published after the author's corresponding publications in this thesis.

Chapter 3 discusses problems in the areas of developing well-performing situation awareness (SA) systems and establishing relevant metrics for security monitoring. The chapter addresses research question RQ1.1 by providing numerous collection and practical usage examples of relevant security metrics based on a production security monitoring framework. The details of this framework are presented in Publication I. In response to RQ1.2, the chapter proposes to verify the suitability and performance of various SA systems in realistic yet supervised environments, e.g., during cyber security exercises (CSXs). This proposal (presented in Publication V) is introduced in chapter 3 and the subsequent CSX-specific discussion carries over to chapter 4.

Chapter 4 describes the use of CSXs to improve and verify complex SA systems and enhance cyber defence operator readiness. The chapter addresses research questions RQ1.3 and RQ1.4 by first describing the background of two CSXs (Crossed Swords and Locked Shields) that needed improved SA systems and naturally aimed to improve the quality of participants' learning experience during the exercise. The response for RQ1.3 details the technical design and implementation of the Frankenstack framework (presented in Publications V and XI), and the CSX Availability Scoring system (presented in Publication VI). Additionally, chapter 4 (collectively with Publications V, VI, and XI) describes efforts of enhancing operator learning experience by providing them instant technical feedback about their in-game progress and actions. Although the qualitative aspects of learning and training quality are not the primary focus points of this thesis, they cannot be neglected because providing continuous improvement for exercise participants is the driving force behind every consecutive exercise.

Chapter 5 presents the novel LogCluster log mining algorithm for discovering both frequent line patterns as well as outlier events from textual event logs. The chapter describes the development and implementation of LogCluster and presents a performance comparison of several competing log analysis algorithms which have been released over the past few years. In response to RQ2.1, the chapter provides numerous practical examples of how LogCluster can be used to perform log data exploration and extract valuable information from *syslog* events as well as CSX-specific SA system logs. The LogCluster algorithm and its various use cases have been presented in Publications II and III.

Chapter 6 describes the detection of covert network communication channels and anomalous network traffic. The chapter addresses RQ2.2 by providing detailed guidance on data exfiltration detection and by presenting a novel anomaly detection framework which employs LogCluster and three distinct anomaly detectors working in ensemble to

discover anomalous flows in NetFlow data. The data exfiltration detection research along with an extensive experiment has been presented in Publication IV. The detailed description and verification results of the novel NetFlow-based anomaly detection framework has been presented in Publication X.

Chapter 7 provides the reasoning behind the need to improve cyber defences by designing Autonomous Intelligent Cyber-defence Agents (AICAs). In response to RQ3.1 and RQ3.2 the first half of the chapter provides a brief overview of the AICA reference architecture and describes the generic purpose, rationale and capabilities of proposed agents. The development of the AICA Reference Architecture was an extensive international project with a number of different authors. The second half of the chapter answers RQ3.3 and focuses on the particular AICA functions (i.e., *Sensing* and *World state identification*) which the thesis author directly contributed to. The chapter is based on three Publications VII, VIII, and IX.

8.2 Future work

When considering some of the thesis contributions, it is safe to conclude that the research is far from complete—there is always more to do and improve.

For instance, the exercise-specific SA systems are in need of improved visualisations which would benefit the participants of the exercises even more. This research would first need to gather the timely feedback from actual participants to assess their information needs at various stages of the exercise more appropriately. This is not an easy task, because exercise participants rarely attend the same exercise more than once, so getting a follow-up feedback assessment from the same person is difficult. Moreover, it is important to bear in mind that people's ideas of an expected feature or functionality can often contradict one another.

While some novel tools (e.g., LogCluster) presented in this thesis are stable and mature, there are some (e.g., Frankenstack and the exercise Availability Scoring system) which still require additional work so that they would be easier to use and deploy in the future. Furthermore, having proper deployment and end-user documentation is something that is often forgot.

Finally, the AICA research is still fairly new and conceptual. The main objective of this research is to advance monitoring and cyber defence techniques forward to a degree where it would be possible to consider a fully functional autonomous cyber defence system. Perhaps eventually leaving the human operator out of the immediate decision-making loop.

List of Figures

List of Tables

References

[1] abuse.ch. abuse.ch - A Swiss non-profit cybersecurity organisation. Available: https://abuse.ch/, 2020.

[2] AgentFly Technologies s.r.o. Tactical Operations - Military & Security. Available: https://www.agentfly.com/tactical-operations-military-security, 2021.

[3] angr. angr. Available: https://angr.io, 2021.

[4] R. Anthony. Detecting security incidents using windows workstation event logs. *SANS Institute Reading Room*, 2013.

[5] Apache Software Foundation. Apache Kafka. Available: https://kafka.apache.org/, 2021.

[6] AppArmor. AppArmor Linux kernel security module. Available: https://apparmor.net/, 2021.

[7] D. L. Arendt, D. Best, R. Burtner, and C. L. Paul. CyberPetri at CDX 2016: Real-time network situation awareness. In *2016 IEEE Symposium on Visualization for Cyber Security (VizSec)*, pages 1–4, October 2016.

[8] D. L. Arendt, R. Burtner, D. M. Best, N. D. Bos, J. R. Gersh, C. D. Piatko, and C. L. Paul. Ocelot: user-centered design of a decision support visualization for network quarantine. In *2015 IEEE Symposium on Visualization for Cyber Security (VizSec)*, pages 1–8, October 2015.

[9] Arizona State University. Will we control artificial intelligence or will it control us? Available: https://asunow.asu.edu/20170223-will-we-control-artificial-intelligence-or-will-it-control-us, 2017.

[10] Arkime. Arkime. Available: https://arkime.com/, 2021.

[11] A. Atlasis. Security Impacts of Abusing IPv6 Extension Headers. Technical report, Centre for Strategic Cyberspace + Security Science, 2012.

[12] Autonomous Intelligent Cyber Defence Agents International Work Group. NCIA – AICA IWG Virtual Technical Workshop. Available: https://www.aica2021.org/ncia-aica-iwg-virtual-technical-workshop/, 2020.

[13] Autonomous Intelligent Cyber Defence Agents International Work Group. 1st International Conference on Autonomous Intelligent Cyber-defence Agents. Available: https://www.aica2021.org/, 2021.

[14] T. Avgerinos, D. Brumley, J. Davis, R. Goulden, T. Nighswander, A. Rebert, and N. Williamson. The mayhem cyber reasoning system. *IEEE Security & Privacy*, 16(2):52–60, 2018.

[15] H. Bahşi, V. Dieves, T. Kangilaski, P. Laud, L. Mõtus, J. Murumets, I. Ploom, J. Priisalu, M. Seeba, E. Täks, K. Tammel, P. Tammpuu, K. Taveter, A. Trumm, T. Truusa, and T. Vihalemm. Mapping the information flows for the architecture of a nationwide situation awareness system : (poster). In *2019 IEEE Conference on Cognitive and Computational Aspects of Situation Management (CogSIMA)*, pages 152–157, 2019.

[16] P. Black, K. Scarfone, and M. Souppaya. *Cyber Security Metrics and Measures*. John Wiley and Sons, 2009.

[17] Bloomberg. AI Scientists Gather to Plot Doomsday Scenarios (and Solutions). Available: https://www.bloomberg.com/news/articles/2017-03-02/ai-scientists-gather-to-plot-doomsday-scenarios-and-solutions, 2017.

[18] B. Blumbergs. *Specialized Cyber Red Team Responsive Computer Network Operations*. PhD thesis, Tallinn University of Technology, 2019.

[19] B. Blumbergs, M. Pihelgas, M. Kont, O. Maennel, and R. Vaarandi. Creating and Detecting IPv6 Transition Mechanism-Based Information Exfiltration Covert Channels. In *Secure IT Systems: 21st Nordic Conference*, *NordSec 2016*, *Oulu*, *Finland*, *November 2-4*, *2016. Proceedings*, pages 85–100. Springer International Publishing, 2016.

[20] Boston Dynamics. Boston Dynamics' Robots. Available: `https://www.bostondynamics.com/`, 2020.

[21] D. Brauckhoff, X. Dimitropoulos, A. Wagner, and K. Salamatian. Anomaly extraction in backbone networks using association rules. *IEEE/ACM Transactions on Networking*, 20(6):1788–1799, 2012.

[22] W. K. Brotby and G. Hinson. *PRAGMATIC Security Metrics: Applying Metametrics to Information Security*. Auerbach Publications, 2013.

[23] Center for Internet Security. CIS Controls Measurement Companion Guide version 6. Available: https://www.cisecurity.org/white-papers/a-measurement-companion-to-the-cis-critical-controls/, October 2015.

[24] Center for Internet Security. CIS Controls V7 Measures & Metrics. Available: https://www.cisecurity.org/white-papers/cis-controls-v7-measures-metrics/, March 2018.

[25] Center for Internet Security. CIS Controls V7.1 Implementation Groups. Available: https://www.cisecurity.org/white-papers/cis-controls-v7-1-implementation-groups/, March 2020.

[26] M. Chmelař. Utilizing MITRE ATT&CK to Create Adversary Reports of Live-Fire Cybersecurity Exercises for Feedback Purposes. Technical report, Tallinn University of Technology, 2020.

[27] Collaboration Support Office. NATO Science and Technology Organization. Available: `https://www.sto.nato.int/`, 2020.

[28] CSIRT-MU. Cyber Czech Security Exercise. Available: `https://csirt.muni.cz/projects/cyber-czech`, 2021.

[29] Cymmetria. Cyber deception & NATO red teams. Available: `https://cymmetria.com/white-paper/nato-crossed-swords/`, 2018.

[30] F. De Gaspari, S. Jajodia, L. V. Mancini, and A. Panico. Ahead: A new architecture for active defense. In *Proceedings of the 2016 ACM Workshop on Automated Decision Making for Active Cyber Defense*, SafeConfig '16, page 11–16, New York, NY, USA, 2016. Association for Computing Machinery.

[31] Defense Advanced Research Projects Agency (DARPA). Cyber Grand Challenge (CGC). Available: https://www.darpa.mil/program/cyber-grand-challenge, 2016.

[32] Defense Advanced Research Projects Agency (DARPA). DARPA Celebrates Cyber Grand Challenge Winners. Available: `https://www.darpa.mil/news-events/2016-08-05a`, 2016.

[33] Docker, Inc. Docker. Available: `https://www.docker.com/`, 2021.

[34] M. Du and F. Li. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 859–864, 2016.

[35] P. Duszyński. Portspoof. Available: `https://github.com/drk1wi/portspoof`, 2017.

[36] Elasticsearch B.V. Elasticsearch. Available: `https://www.elastic.co/elasticsearch`, 2021.

[37] Elasticsearch B.V. Kibana. Available: `https://www.elastic.co/kibana`, 2021.

[38] Elasticsearch B.V. Logstash. Available: `https://www.elastic.co/logstash`, 2021.

[39] M. Endsley. Toward a Theory of Situation Awareness in Dynamic Systems. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 37:32–64, March 1995.

[40] M. Ernits, K. Maennel, S. Mäses, T. Lepik, and O. Maennel. From Simple Scoring Towards a Meaningful Interpretation of Learning in Cybersecurity Exercises. In *15th International Conference on Cyber Warfare and Security (ICCWS 2020)*, March 2020.

[41] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[42] J. N. Frye, C. K. Veitch, M. E. Mateski, J. T. Michalski, J. M. Harris, C. M. Trevino, and S. Maruoka. Cyber threat metrics, March 2012.

[43] Q. Fu, J. Lou, Y. Wang, and J. Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 Ninth IEEE International Conference on Data Mining*, pages 149–158, 2009.

[44] D. Geer, K. S. Hoo, and A. Jaquith. Information security: why the future belongs to the quants. *IEEE Security & Privacy*, 1(4):24–32, 2003.

[45] R. Graf, F. Skopik, and K. Whitebloom. A decision support model for situational awareness in national cyber operations centers. In *2016 International Conference On Cyber Situational Awareness, Data Analytics And Assessment (CyberSA)*, pages 1–6, 2016.

[46] GrafanaLabs. Grafana. Available: `https://grafana.com/`, 2021.

[47] Greycortex. Greycortex supports Crossed Shields for second year. Available: https://www.greycortex.com/blog/greycortex-supports-crossed-shields-second-year, 2019.

[48] M. Grill, I. Nikolaev, V. Valeros, and M. Rehak. Detecting dga malware using netflow. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 1304–1309, 2015.

[49] A. Guarino. Autonomous intelligent agents in cyber offence. In *2013 5th International Conference on Cyber Conflict (CYCON 2013)*, pages 1–12, 2013.

[50] P. Haag. nfdump. Available: `https://github.com/phaag/nfdump`, 2021.

[51] M. J. Hall, D. David Hansen, and K. Jones. Cross-domain situational awareness and collaborative working for cyber security. In *2015 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*, pages 1–8, 2015.

[52] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen. Logmine: Fast pattern recognition for log analytics. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, page 1573–1582, New York, NY, USA, 2016. Association for Computing Machinery.

[53] O. Hartong. sysmon-modular. Available: `https://github.com/olafhartong/sysmon-modular`, 2021.

[54] W. H. Hawkins, J. D. Hiser, M. Co, A. Nguyen-Tuong, and J. W. Davidson. Zipr: Efficient static binary rewriting for security. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 559–566, 2017.

[55] P. He, J. Zhu, Z. Zheng, and M. R. Lyu. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 33–40, 2017.

[56] C. H. Heinl. Artificial (intelligent) agents and active cyber defence: Policy implications. In *2014 6th International Conference On Cyber Conflict (CyCon 2014)*, pages 53–66, 2014.

[57] D. S. Henshel, G. M. Deckard, B. Lufkin, N. Buchler, B. Hoffman, P. Rajivan, and S. Collman. Predicting proficiency in cyber defense team exercises. In *MILCOM 2016 - 2016 IEEE Military Communications Conference*, pages 776–781, November 2016.

[58] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Communications Surveys Tutorials*, 16(4):2037–2064, 2014.

[59] R. Hofstede, V. Bartoš, A. Sperotto, and A. Pras. Towards real-time intrusion detection for netflow and ipfix. In *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, pages 227–234, 2013.

[60] J. Hou, P. Fu, Z. Cao, and A. Xu. Machine learning based ddos detection through netflow analysis. In *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, pages 1–6, 2018.

[61] IBM. Drain3. Available: `https://github.com/IBM/Drain3`, 2021.

[62] IEEE. Conference on Artificial Intelligence for Applications. Available: `https://ieeexplore.ieee.org/xpl/conhome/1000050/all-proceedings`, 1995.

[63] IEEE. Proceedings of International Symposium on Autonomous Decentralized Systems (ISADS). Available: https://ieeexplore.ieee.org/xpl/conhome/1000067/all-proceedings, 2017.

[64] IEEE. International Conference on Tools for Artificial Intelligence (ICTAI). Available: https://ieeexplore.ieee.org/xpl/conhome/1000763/all-proceedings, 2019.

[65] InfluxData. Telegraf. Available: https://www.influxdata.com/time-series-platform/telegraf/, 2021.

[66] InfluxData. TICK stack. Available: https://www.influxdata.com/time-series-platform/, 2021.

[67] H. Irino. softflowd – A software NetFlow probe. Available: https://code.google.com/archive/p/softflowd/, 2021.

[68] S. Jajodia, G. Cybenko, V. Subrahmanian, V. Swarup, C. Wang, and M. Wellman. *Adaptive Autonomous Secure Cyber Systems*. Springer International Publishing, 2020.

[69] A. Jaquith. Security Metrics: Replacing Fear, Uncertainty, and Doubt, January 2007.

[70] B. S. Jese. Snoopy Logger. Available: https://github.com/a2o/snoopy, 2021.

[71] N. Känzig, R. Meier, L. Gambazzi, V. Lenders, and L. Vanbever. Machine Learning-based Detection of C&C Channels with a Focus on the Locked Shields Cyber Defense Exercise. In *2019 11th International Conference on Cyber Conflict (CyCon)*, volume 900, pages 1–19, 2019.

[72] J. Kim, K. Kim, and M. Jang. Cyber-physical battlefield platform for large-scale cybersecurity exercises. In *2019 11th International Conference on Cyber Conflict (CyCon)*, volume 900, pages 1–19, 2019.

[73] J. Kim, Y. Maeng, and M. Jang. Becoming invisible hands of national live-fire attack-defense cyber exercise. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, pages 77–84, 2019.

[74] A. Kind, M. P. Stoecklin, and X. Dimitropoulos. Histogram-based traffic anomaly detection. *IEEE Transactions on Network and Service Management*, 6(2):110–121, 2009.

[75] J. Klein, S. Bhulai, M. Hoogendoorn, R. Van Der Mei, and R. Hinfelaar. Detecting Network Intrusion beyond 1999: Applying Machine Learning Techniques to a Partially Labeled Cybersecurity Dataset. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 784–787, 2018.

[76] M. Klemettinen. A knowledge discovery methodology for telecommunication network alarm databases, May 1999.

[77] M. Klemettinen, H. Mannila, P. Moen, H. Toivonen, and A. Verkamo. Finding interesting rules from large sets of discovered association rules. *Proceedings of the Third International Conference on Information and Knowledge Management*, February 1995.

[78] J. Kohlrausch and E. A. Brin. Arima supplemented security metrics for quality assurance and situational awareness. *Digital Threats: Research and Practice*, 1(1), March 2020.

[79] M. Kont. Sigma rule engine. Available: `https://github.com/markuskont/go-sigma-rule-engine`, 2020.

[80] M. Kont and M. Pihelgas. Automated virtual testing environment. Available: `https://github.com/markuskont/exfil-testbench`, 2015.

[81] M. Kont and M. Pihelgas. *IDS for logs: Towards implementing a streaming Sigma rule engine*. NATO CCD COE Publications, 2020.

[82] M. Kont, M. Pihelgas, K. Maennel, B. Blumbergs, and T. Lepik. Frankenstack: Toward real-time Red Team feedback. In *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, pages 400–405, October 2017.

[83] M. Kont, M. Pihelgas, J. Wojtkowiak, L. Trinberg, and A.-M. Osula. *Insider Threat Detection Study*. NATO CCD COE Publications, 2015.

[84] I. Kotenko and E. Doynikova. Dynamical calculation of security metrics for countermeasure selection in computer networks. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 558–565, 2016.

[85] A. Kott, L. V. Mancini, P. Théron, M. Drasar, E. Dushku, H. Günther, M. Kont, B. Leblanc, A. Panico, M. Pihelgas, and K. Rzadca. Initial Reference Architecture of an Intelligent Autonomous Agent for Cyber Defense. *CoRR*, abs/1803.10664, 2018.

[86] A. Kott, P. Théron, M. Drašar, E. Dushku, B. LeBlanc, P. Losiewicz, A. Guarino, L. Mancini, A. Panico, M. Pihelgas, and K. Rzadca. Autonomous Intelligent Cyber-defense Agent (AICA) Reference Architecture. Release 2.0, 2019.

[87] A. Kott and P. Theron. Doers, not watchers: Intelligent autonomous agents are a path to cyber resilience. *IEEE Security & Privacy*, 18(3):62–66, 2020.

[88] A. Kott, P. Théron, L. V. Mancini, E. Dushku, A. Panico, M. Drašar, B. LeBlanc, P. Losiewicz, A. Guarino, M. Pihelgas, and K. Rzadca. An introductory preview of Autonomous Intelligent Cyber-defense Agent reference architecture, release 2.0. *The Journal of Defense Modeling and Simulation*, 17(1):51–54, 2020.

[89] E. Leblond. Finding the Bad Guys, Yes Really. Available: `https://www.youtube.com/watch?v=Scntdv1Vp_0`, 2017. Hack.lu 2017 Presentation.

[90] E. Leblond. Finding the Bad Guys, Yes Really. SuriCon 2017, 2017. Presentation.

[91] E. Leblond and P. Manev. Suricata and XDP, Performance with a S like Security. Available: https://suricon.net/wp-content/uploads/2019/11/SURICON2019_XDP-New-Features-and-Testing-Methodologies.pdf, 2019.

[92] R. Liivoja, M. Naagel, and A. Väljataga. *Autonomous Cyber Capabilities under International Law*. NATO Cooperative Cyber Defence Centre of Excellence, 2018.

[93] LogPAI. Loghub. Available: `https://github.com/logpai/loghub`, 2020.

[94] LogPAI. Logparser. Available: https://github.com/logpai/logparser, 2021.

[95] G. Lyon. Nmap – Network Mapper. Available: https://nmap.org/, 2021.

[96] K. Maennel, J. Kim, and S. Sütterlin. From Text Mining to Evidence Team Learning in Cybersecurity Exercises. In *Companion Proceedings 10th International Conference on Learning Analytics and Knowledge (LAK20)*, March 2020.

[97] K. Maennel, R. Ottis, and O. Maennel. Improving and Measuring Learning Effectiveness at Cyber Defense Exercises. In *Secure IT Systems: 22nd Nordic Conference, NordSec 2017, Tartu, Estonia, November 8-10, 2017. Proceedings*, pages 123–138, November 2017.

[98] A. Makanju. Exploring event log analysis with minimum apriori information, 2012.

[99] A. Makanju, S. Brooks, A. N. Zincir-Heywood, and E. E. Milios. Logview: Visualizing event log clusters. In *2008 Sixth Annual Conference on Privacy, Security and Trust*, pages 99–108, 2008.

[100] A. Makanju, A. Nur Zincir-Heywood, and E. Milios. A Lightweight Algorithm for Message Type Extraction in System Application Logs. In *IEEE Transactions on Knowledge and Data Engineering*, pages 1921 – 1936, September 2012.

[101] W. Mazurczyk, K. Powójski, and L. Caviglione. Ipv6 covert channels in the wild. In *Proceedings of the Third Central European Cybersecurity Conference*, CECC 2019, New York, NY, USA, 2019. Association for Computing Machinery.

[102] F. J. R. Melón, T. Väisänen, and M. Pihelgas. EVE and ADAM: Situation Awareness Tools for NATO CCDCOE Cyber Exercises. In *STO-MP-SCI-300 Cyber Physical Security of Defense Systems*, pages 10–1–10–15, 2018.

[103] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, and R. Sasnauskas. A search-based approach for accurate identification of log message formats. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, pages 167–16710, 2018.

[104] Microsoft. Windows Sysinternals - Sysmon. Available: https://technet.microsoft.com/en-us/sysinternals/sysmon, 2021.

[105] Milrem AS. Milrem Robotics. Available: https://milremrobotics.com/, 2020.

[106] D. W. Morgan. Interview: Ethan Galstad - The Nagios future. Available: http://www.h-online.com/open/features/Interview-Ethan-Galstad-The-Nagios-future-958826.html, 2010.

[107] L. Motus, M. Teichmann, T. Kangilaski, J. Priisalu, and J. Kaugerand. Some issues in modelling comprehensive situation awareness. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 540–545, 2019.

[108] R. P. Murphy. IPv6 / ICMPv6 Covert Channels. DEF CON'14, 2014. Presentation.

[109] Nagios. Nagios Core 4.x Version History. Available: https://www.nagios.org/projects/nagios-core/history/4x/, 2021.

[110] Nagios Enterprises. Nagios Core. Available: https://www.nagios.org/projects/nagioscore, 2021.

[111] NATO CCD COE. EVE - Event Visualization Environment. Available: https://github.com/ccdcoe/EVE, 2017.

[112] NATO CCD COE. Otta. Available: https://github.com/ccdcoe/otta, 2017.

[113] NATO CCD COE. frankenSEC. Available: https://github.com/ccdcoe/frankenSEC, 2019.

[114] NATO CCD COE. Locked Shields 2019. Available: https://www.ccdcoe.org/news/2019/france-wins-cyber-defence-exercise-locked-shields-2019/, 2019.

[115] NATO CCD COE. Locked Shields Exercise. Available: https://ccdcoe.org/locked-shields.html, 2019.

[116] NATO CCD COE. Exercise Crossed Swords 2020 Reached New Levels of Multinational and Interdisciplinary Cooperation. Available: https://ccdcoe.org/news/2020/exercise-crossed-swords-2020-reached-new-levels-of-multinational-and-interdisciplinary-cooperation/, 2020.

[117] NATO CCD COE. Frankenstack. Available: https://github.com/ccdcoe/frankenstack, 2020.

[118] NATO CCD COE. Peek. Available: https://github.com/ccdcoe/go-peek, 2020.

[119] NATO CCD COE. Crossed Swords Exercise. Available: https://ccdcoe.org/exercises/crossed-swords/, 2021.

[120] NATO Science and Technology Organization. IST-152: Intelligent, Autonomous and Trusted Agents for Cyber Defense and Resilience. Available: https://www.sto.nato.int/Lists/test1/activitydetails.aspx?ID=16533, 2019.

[121] NfSen. NfSen. Available: http://nfsen.sourceforge.net/, 2011.

[122] A. Nguyen-Tuong, D. Melski, J. W. Davidson, M. Co, W. Hawkins, J. D. Hiser, D. Morris, D. Nguyen, and E. Rizzi. Xandra: An autonomous cyber battle system for the cyber grand challenge. *IEEE Security & Privacy*, 16(2):42–51, 2018.

[123] O.-P. Niemi, A. Levomäki, and J. Manner. Dismantling intrusion prevention systems. *SIGCOMM Comput. Commun. Rev.*, 42(4):285–286, August 2012.

[124] NXLog Ltd. NXLog. Available: https://nxlog.co/, 2021.

[125] R. Ošlejšek, V. Rusnak, K. Burská, V. Švábenský, J. Vykopal, and J. Cegan. Conceptual model of visual analytics for hands-on cybersecurity training. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, 2020.

[126] R. Ošlejšek, J. Vykopal, K. Burská, and V. Rusňák. Evaluation of cyber defense exercises using visual analytics process. In *2018 IEEE Frontiers in Education Conference (FIE)*, pages 1–9, 2018.

[127] One Identity LLC. syslog-ng. Available: https://www.syslog-ng.com/, 2021.

[128] Open Information Security Foundation. Suricata. Available: https://suricata-ids.org/, 2021.

[129] OpenJS Foundation. Node.js. Available: `https://nodejs.org/`, 2021.

[130] E. Orlandi. Computer security: a consequence of information technology quality. In *IEEE International Carnahan Conference on Security Technology, Crime Countermeasures*, pages 109–112, 1990.

[131] R. Ošlejšek, V. Rusňák, K. Burská, V. Švábenský, and J. Vykopal. Visual feedback for players of multi-level capture the flag games: Field usability study. In *2019 IEEE Symposium on Visualization for Cyber Security (VizSec)*, 2019.

[132] S. C. Payne. A Guide to Security Metrics, June 2006.

[133] M. Pěchouček, M. Jakob, and P. Novák. Towards simulation-aided design of multi-agent systems. In *Programming Multi-Agent Systems*, pages 3–21, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[134] M. Pendleton, R. Garcia-Lebron, J.-H. Cho, and S. Xu. A survey on systems security metrics. *ACM Comput. Surv.*, 49(4), December 2016.

[135] M. Pihelgas. Design and Implementation of an Availability Scoring System for Cyber Defence Exercises. In *14th International Conference on Cyber Warfare and Security (ICCWS 2019)*, page 329–337, 2019.

[136] M. Pihelgas and M. Kont. Frankenstack: Real-time Cyberattack Detection and Feedback System for Technical Cyber Exercises. In *2021 IEEE CSR Workshop on Cyber Ranges and Security Training (CRST)*. IEEE, July 2021. (Accepted paper).

[137] J. Preden, J. Kaugerand, E. Suurjaak, S. Astapov, L. Motus, and R. Pahtma. Data to decision: pushing situational information needs to the edge of the network. In *2015 IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision*, pages 158–164, 2015.

[138] Proofpoint. Emerging Threats rules. Available: `https://rules.emergingthreats.net/`, 2020.

[139] Python Software Foundation. Python Package Index (PyPI). Available: `https://pypi.org/`, 2021.

[140] A. Ramos, M. Lazar, R. H. Filho, and J. J. P. C. Rodrigues. Model-based quantitative network security metrics: A survey. *IEEE Communications Surveys Tutorials*, 19(4):2704–2734, 2017.

[141] Red Hat. Security-Enhanced Linux (SELinux). Available: `https://www.redhat.com/en/topics/linux/what-is-selinux`, 2021.

[142] RedisLabs. Redis. Available: `https://redis.io/`, 2021.

[143] T. Reidemeister. *Fault Diagnosis in Enterprise Software Systems Using Discrete Monitoring Data*. PhD thesis, University of Waterloo, 2012.

[144] T. Reidemeister, M. Jiang, and P. A. S. Ward. Mining unstructured log files for recurrent fault diagnosis. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, pages 377–384, May 2011.

[145] F. Roth. Sigma. Available: `https://github.com/Neo23x0/sigma`, 2021.

[146] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.

[147] P. Sanders. The Effect of Packet Loss on an IDS Deployment. Available: https://www.napatech.com/the-effect-of-packet-loss-on-an-ids-deployment/, 2019.

[148] SANS Institute. CIS Critical Security Controls. Available: `https://www.sans.org/critical-security-controls`, 2020.

[149] N. Satterly. Alerta. Available: `http://alerta.io/`, 2021.

[150] Y. Shoshitaishvili, A. Bianchi, K. Borgolte, A. Cama, J. Corbetta, F. Disperati, A. Dutcher, J. Grosen, P. Grosen, A. Machiry, C. Salls, N. Stephens, R. Wang, and G. Vigna. Mechanical Phish: Resilient Autonomous Hacking. *IEEE Security & Privacy*, 16(2):12–22, 2018.

[151] Y. Shoshitaishvili, R. Wang, C. Salls, N. Stephens, M. Polino, A. Dutcher, J. Grosen, S. Feng, C. Hauser, C. Kruegel, and G. Vigna. SOK: (State of) The Art of War: Offensive Techniques in Binary Analysis. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 138–157, 2016.

[152] D. Sislak, P. Volf, S. Kopriva, and M. Pěchouček. *AgentFly: Scalable, High-Fidelity Framework for Simulation, Planning and Collision Avoidance of Multiple UAVs*, chapter 9, pages 233–264. John Wiley & Sons, Ltd, 2012.

[153] Software Freedom Conservancy. Selenium. Available: `https://www.selenium.dev//`, 2021.

[154] Stamus Networks. Stamus Networks at XS20. Available: https://twitter.com/StamusN/status/1339968510924120066, 2021.

[155] Starship Technologies OÜ. Starship. Available: `https://www.starship.xyz/`, 2020.

[156] SwiftOnSecurity. sysmon-config. Available: https://github.com/SwiftOnSecurity/sysmon-config, 2021.

[157] Tesla Motors. Autopilot. Available: `https://www.tesla.com/autopilot`, 2020.

[158] Thales Group. Secure, sustainable smart cities and the IoT. Available: https://www.thalesgroup.com/en/markets/digital-identity-and-security/iot/inspired/smart-cities, 2020.

[159] The Graphite Project. Graphite. Available: `https://graphiteapp.org/`, 2021.

[160] The Zeek Project. The Zeek Network Security Monitor. Available: `https://zeek.org/`, 2021.

[161] P. Théron, A. Kott, M. Drašar, K. Rzadca, B. LeBlanc, M. Pihelgas, L. Mancini, and A. Panico. Towards an active, autonomous and intelligent cyber defense of military systems: The NATO AICA reference architecture. In *2018 International Conference on Military Communications and Information Systems (ICMCIS)*, pages 1–9, May 2018.

[162] P. Théron, A. Kott, M. Drašar, K. Rzadca, B. LeBlanc, M. Pihelgas, L. Mancini, and F. de Gaspari. Reference Architecture of an Autonomous Agent for Cyber Defense of Complex Military Systems. In *Adaptive Autonomous Secure Cyber Systems*, pages 1–21, Cham, 2020. Springer International Publishing.

[163] D. Tovarňák, S. Špaček, and J. Vykopal. *Dataset: Traffic and Log Data Captured During a Cyber Defense Exercise*. Zenodo, April 2020.

[164] D. Tovarňák, S. Špaček, and J. Vykopal. Traffic and log data captured during a cyber defense exercise. *Data in Brief*, 31:105784, 2020.

[165] E. Tyugu. Command and control of cyber weapons. In *2012 4th International Conference on Cyber Conflict (CYCON 2012)*, pages 1–11, 2012.

[166] USENIX. The Computer Failure Data Repository (CFDR). Available: `https://www.usenix.org/cfdr-data`.

[167] R. Vaarandi. A data clustering algorithm for mining patterns from event logs. In *IP Operations Management, 2003. (IPOM 2003). 3rd IEEE Workshop on*, pages 119–126, October 2003.

[168] R. Vaarandi. Mining event logs with SLCT and LogHound. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, pages 1071–1074, April 2008.

[169] R. Vaarandi. Detecting anomalous network traffic in organizational private networks. In *Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2013 IEEE International Multi-Disciplinary Conference on*, pages 285–292, February 2013.

[170] R. Vaarandi. LogCluster. Available: `http://ristov.github.io/logcluster`, 2019.

[171] R. Vaarandi. SEC - simple event correlator. Available: `https://simple-evcorr.github.io`, 2021.

[172] R. Vaarandi, B. Blumbergs, and E. Çalişkan. Simple event correlator - Best practices for creating scalable configurations. In *Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2015 IEEE International Conference on*, pages 96–100, March 2015.

[173] R. Vaarandi, M. Kont, and M. Pihelgas. Event log analysis with the LogCluster tool. In *MILCOM 2016 - 2016 IEEE Military Communications Conference*, pages 982–987, November 2016.

[174] R. Vaarandi and M. Pihelgas. Using Security Logs for Collecting and Reporting Technical Security Metrics. In *Military Communications Conference (MILCOM), 2014 IEEE*, pages 294–299, October 2014.

[175] R. Vaarandi and M. Pihelgas. LogCluster - A data clustering and pattern mining algorithm for event logs. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 1–7, November 2015.

[176] R. Vaarandi and M. Pihelgas. NetFlow Based Framework for Identifying Anomalous End User Nodes. In *15th International Conference on Cyber Warfare and Security (ICCWS 2020)*, page 448–456, 2020.

[177] R. Vaarandi and K. Podiņš. Network IDS alert classification with frequent itemset mining and data clustering. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 451–456, October 2010.

[178] J. Vykopal, R. Ošlejšek, K. Burská, and K. Zákopčanová. Timely feedback in unstructured cybersecurity exercises. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, page 173–178, New York, NY, USA, 2018. Association for Computing Machinery.

[179] J. Vykopal, M. Vizvary, R. Oslejsek, P. Celeda, and D. Tovarnak. Lessons learned from complex hands-on defence exercises in a cyber range. In *2017 IEEE Frontiers in Education Conference (FIE)*, pages 1–8, 2017.

[180] D. Wallace. Cyber weapon reviews under international humanitarian law: A critical analysis. *Tallinn Papers*, 11, 2018.

[181] S. Wendzel and S. Zander. Detecting protocol switching covert channels. In *37th Annual IEEE Conference on Local Computer Networks*, pages 280–283, 2012.

[182] F. Zhou, W. Huang, Y. Zhao, Y. Shi, X. Liang, and X. Fan. Entvis: A visual analytic tool for entropy-based network traffic anomaly detection. *IEEE Computer Graphics and Applications*, 35(6):42–50, 2015.

[183] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu. Tools and benchmarks for automated log parsing. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '19, page 121–130. IEEE Press, 2019.

[184] C. Zhuge and R. Vaarandi. Efficient Event Log Mining with LogClusterC. In *Proceedings of the 2017 IEEE International Conference on Big Data Security on Cloud*, pages 261–266, May 2017.

Acknowledgements

The **cruelties of rare diseases** and the **effectuation of unlikely odds within my family** has rendered this undertaking significantly more difficult than many of the readers can probably imagine.

I am deeply grateful to my beloved family for their unwavering support throughout the years. Without their kind understanding and encouragement it would not have been possible for me to complete this endeavour.

Furthermore, I thank my supervisors, Risto and Olaf, for their valuable advice, continuous support, and patience. I would also like to thank Raimo for supporting the *research pillar*, which has enabled me to carry out my research. I would also like to thank my current and past colleagues at the NATO CCD COE for their cooperation. Special thanks go to Markus, a good friend and an invaluable co-researcher in many sophisticated projects.

This research has been supported by the Estonian IT Academy (StudyITin.ee) and the NATO Cooperative Cyber Defence Centre of Excellence.

Abstract

Automating Defences against Cyber Operations in Computer Networks

This thesis is based on a collection of eleven publications. The thesis explores the improvement of organisational security monitoring capability and readiness to advance towards intelligent autonomous cyber defence systems.

Additionally, the thesis aims to reduce the gap between suggestions derived from academic research and practical guidelines that are useful for cyber defenders. The feasibility of utilising theoretical research outcomes in practice has been criticised in related publications by several different authors. To relieve this issue, this thesis and the bundled collection of publications provide numerous actionable recommendations and practical examples.

This thesis addresses problems in the areas of establishing which metrics are relevant for security monitoring, how to build both general-purpose and cyber-exercise-specific situation awareness systems, how to raise SA qualifications and readiness of cyber defenders, how to implement and verify novel log mining algorithms and network security frameworks for cyber defence, and how to improve cyber defences by designing autonomous intelligent cyber-defence agents.

The thesis provides recommendations for metrics and log data collection, transformation, and analysis methods alongside relevant data representation techniques. Furthermore, a novel data clustering and log mining algorithm LogCluster is proposed, compared thoroughly with several other log analysis tools, and later used to provide practical examples of clustering logs from two different cyber security exercises (Locked Shields and Crossed Swords). Furthermore, the thesis describes two novel cyber-exercise-specific situation awareness systems—Frankenstack and the Availability Scoring system: comprising an overview of the development process, technical architecture, and validation during the aforementioned cyber security exercises. In the area of network security, the thesis describes the research on data exfiltration detection with open-source tools and details a novel NetFlow-based anomaly detection framework. Finally, the concept and reference architecture for autonomous intelligent cyber-defence agents is described and proposed as the basis for future military and civil cyber defence systems.

Kokkuvõte
Arvutivõrkude kaitse automatiseerimine küberoperatsioonide vastu

Käesolev ingliskeelne doktoritöö põhineb autori üheteistkümnel publikatsioonil ja uurib võimalusi arvutivõrkude küberkaitse automatiseerimiseks küberoperatsioonide vastu. Töö peamine eesmärk on luua eeldused ning tõsta üldist valmisolekut autonoomsete küberkaitse süsteemide arendamiseks ja juurutamiseks lähitulevikus.

Töö teine eesmärk on teoreetilise teadustöö ning praktikas rakendatavate juhiste tihedam sidumine. Akadeemilistes publikatsioonides jagatud soovituste rakendamine on tihti liiga keeruline—taoliste soovituste ebapraktilisust on kritiseeritud mitmes doktoritöös viidatud allikas. Selle probleemi leevendamiseks pakub käesolev doktoritöö ja sellega kaasnevad publikatsioonid arvukalt praktilisi soovitusi ning näiteid tehniliste lahenduste juurutamiseks.

Doktoritöös otsitakse vastuseid järgnevatele küsimustele ja probleemidele: milliseid tehnilisi meetrikaid on oluline jälgida küberturbe seires; kuidas rajada nii tavakasutuse kui ka küberharjutuste jaoks mõeldud situatsiooniteadlikkuse süsteeme; mil viisil oleks võimalik tõsta küberkaitsjate üldist kvalifikatsiooni ja treenida nende oskusi situatsiooniteadlikkuse valdkonnas; kuidas oleks võimalik testida uudsete andmekaevandamisalgoritmide ja võrguturbesüsteemide efektiivsust ja töökindlust; ning kuidas tõsta küberkaitse võimekust iseõppivate autonoomsete küberkaitse agentidega.

Doktoritöö kätkeb soovitusi meetrikate ja logiandmete kogumise, töötlemise ning esitlemise parendamiseks. Töö kirjeldab logide kaevandamise algoritmi LogCluster, võrdleb LogClusterit mitme konkureeriva logianalüüsi tööriistaga ning toob mitmeid praktilisi näiteid LogClusteri kasutamisest küberharjutuste andmekogude analüüsimiseks. Eraldi käsitletakse küberharjutuste tarbeks loodud kahte vabatarkvaralist monitooringusüsteemi: töö sisaldab ülevaadet nende süsteemide väljatöötamisest, komponentide tehnilisest ülesehitusest ning katsetamisest kahe erineva küberharjutuse, Locked Shields ja Crossed Swords, raames. Võrguturbe valdkonnas uurib töö andmelekete avastamist vabatarkvaraliste vahenditega ning kirjeldab hiljuti publitseeritud NetFlow-põhist võrguanomaaliate tuvastamise seireraamistikku. Viimaks kirjeldatakse kontseptuaalset intelligentsete küberagentide etalonarhitektuuri, mida saaks potentsiaalselt rakendada autonoomsete küberkaitse agentide arendamisel ja juurutamisel.

Appendix 1

Publication I

R. Vaarandi and M. Pihelgas. Using Security Logs for Collecting and Reporting Technical Security Metrics. In *Military Communications Conference (MILCOM), 2014 IEEE*, pages 294–299, October 2014

The paper is included in the *Proceedings of the 2014 IEEE Military Communications Conference (MILCOM 2014)*.
DOI: 10.1109/MILCOM.2014.53.

Using Security Logs for Collecting and Reporting Technical Security Metrics

Risto Vaarandi and Mauno Pihelgas
NATO Cooperative Cyber Defence Centre of Excellence
Tallinn, Estonia
firstname.lastname@ccdcoe.org

***Abstract*—During recent years, establishing proper metrics for measuring system security has received increasing attention. Security logs contain vast amounts of information which are essential for creating many security metrics. Unfortunately, security logs are known to be very large, making their analysis a difficult task. Furthermore, recent security metrics research has focused on generic concepts, and the issue of collecting security metrics with log analysis methods has not been well studied. In this paper, we will first focus on using log analysis techniques for collecting technical security metrics from security logs of common types (e.g., IDS alarm logs, workstation logs, and Netflow data sets). We will also describe a production framework for collecting and reporting technical security metrics which is based on novel open-source technologies for big data.**

Keywords—security metrics; security log analysis

I. INTRODUCTION

During recent years, the question "how to measure system security?" has received increasing attention, and has been addressed in a number of academic papers [1–6], governmental research reports [7–9], standards [10], books [11–12], and various other documents like recommendations and domain overviews [13–15]. For easing the process of measuring the system security, the notion of *security metric* is employed by most researchers and practitioners. Although this notion is defined in a slightly different way in different documents, all sources agree that security metric refers to a standard of measurement. For example, one can define the metric *number of port scanners per day* which involves collecting relevant data from a firewall after the end of each day. In several sources, the following common properties of a good security metric have been identified [4, 9, 11–13]:

- It is unambiguous and meaningful for predefined purposes, making sense to the human analyst
- Taking measurements for the metric should not involve significant cost impact
- Measurements should be taken consistently using the same methodology, with appropriate time frames between measurements, and preferably through an automated data collection procedure

It is often hard to find a metric which is equally meaningful for every possible audience. For example, while the metric *number of port scanners per day* is useful for security administrators, it has little value to a higher level executive who is interested in business level metrics. In a recent study [14], security metrics are classified by their expected audience into *management*, *operational*, and *technical* metrics. Technical metrics provide details for security experts, but also a foundation for other two metric classes which are primarily meant for different levels of management [14].

Even if the metric is meaningful for a particular audience, the knowledge of a wider context around the metric is often useful for increasing its clarity [13]. For instance, the metric *number of port scanners per day* does not make much sense if one is looking only at a single measurement taken for the last 24 hours, since it is not known what are the usual values for this metric in a given environment. Furthermore, during the metrics collection process the knowledge about the surrounding environment should be used. For example, if known false positive alarms are excluded when the metric *number of network attacks per day* is collected, the value of this metric will greatly increase.

Although security metrics related issues have been studied in a number of sources, they often lack detailed recommendations for implementing security metrics collection and reporting system. Furthermore, since many metrics can only be obtained from security logs which are often very large in size, metrics collection requires a log management solution for big data with efficient searching and reporting capabilities. However, many traditional log management solutions are not able to cope with big data which creates a serious obstacle for metrics collection. Moreover, high-end commercial solutions are not affordable for many smaller institutions.

Also, in existing literature metrics reporting is often seen as the generation of static reports to end users. One notable exception is a hierarchical visualization architecture proposed by Savola and Heinonen [3] which supports interactive navigation from generic metrics to underlying more specific metrics. We take a step further and argue that the security metrics reporting system should be able to access raw security data sets (such as security logs) and have efficient drill-down functionality – the generation of more specific reports on user-defined queries, and the identification of individual entities in raw security data (such as log messages or Netflow records). This allows the human analyst to study details behind the

This work was supported by the SEB financial group. This paper is a product of the authors; it does not represent the opinions or official policies of NATO CCDCOE or NATO and is designed to provide an independent position.

metric and increase its meaningfulness [3], and also helps to find root causes for anomalies which have been spotted in reported metric values.

During the last few years, open-source technologies for storing, searching, analyzing, and visualizing very large log data sets have rapidly emerged (e.g., Elasticsearch, Kibana, and Graylog2). These technologies can be used for creating a cost-efficient security metrics collection and reporting system with dynamic reporting and drill-down capabilities for security logs.

Unfortunately, these developments have received little attention in recent academic and industrial papers, and previous works have not focused on using security logs for metrics collection. This paper addresses this research gap, and discusses log analysis methods and open-source solutions for collecting and reporting technical security metrics. The remainder of the paper is organized as follows – section II provides an overview of related work, section III discusses log analysis methods for extracting metrics from security logs of common types, section IV describes an open-source based production framework for security metrics collection and reporting, and section V concludes the paper.

II. Related Work

One of the earliest works which suggested the use of security metrics was a book by Jaquith [11]. The book describes the properties of a good metric and provides a detailed discussion on how to report metrics. During the past few years, the use of security metrics has been proposed for a wide variety of domains, including SCADA and other control systems [5, 7, 8], cloud computing [6], application security [2], software design [1], and assessment of cyber threats [9].

Recently, the Center for Internet Security has published a report [14] on standard metric and data definitions that can be used across different organizations, in order to collect and analyze data on security processes and outcomes. The paper offers universal guidelines on implementing security metrics program in an organization. The paper proposes 28 metric definitions that have all been categorized in two ways, either by relevant business function or by purpose and target audience. These metrics are meant to serve as a starting point for organizations which are beginning to implement their metrics program. Nevertheless, the paper does not offer any detailed recommendations for implementing a production system for metrics collection and reporting.

A recent paper by the Council on CyberSecurity [15] describes 20 critical controls for achieving effective cyber defense. The paper considers security log collection and analysis as one of the critical controls, and also emphasizes the importance of IDS and Netflow based network monitoring. Although the main focus of the paper is not on security metrics, it proposes a number of specific metrics for measuring the efficiency of suggested cyber defense controls.

Security metrics have also been discussed in the ISO/IEC 27004:2009 standard [10] which aims to measure, report on, and systematically improve the effectiveness of Information Security Management Systems that have been specified in ISO/IEC 27001. However, the current standard has been criticized by some security practitioners for being too generic and lacking practical guidance on which particular metrics to collect [12].

In addition to aforementioned sources, a recent book by Brotby and Hinson [12] offers practical recommendations and examples on implementing security metrics program. The authors of the book propose the novel PRAGMATIC methodology for defining, scoring, and ranking security metrics, in order to identify the most beneficial metrics for different audiences (e.g., security professionals, managers, and other stakeholders). Furthermore, the book describes over 150 example metrics, in order to help the reader to build his/her own metrics program.

Apart from generic studies, security metrics have also been suggested for measuring specific aspects of cyber security. For example, a recent study conducted in Sandia National Labs [9] discusses possible metrics for cyber threats (malicious organizations and individuals), and proposes the use of the threat matrix model for assessing cyber threats.

III. Extracting Technical Security Metrics from Security Logs

As discussed in the previous section, existing works often focus on generic security metric concepts, and lack recommendations for implementing production systems for metrics collection and reporting. In this section, we will discuss the use of log analysis methods and tools for several common security log types, in order to collect technical security metrics.

A. Extracting Security Metrics from IDS Alarm Logs

Today, IDSs are used by vast majority of institutions which are processing data of critical importance. Therefore, IDS alarm based security metrics are a popular choice for measuring the system security and the threat level against the local network. In production systems, it is a common practice to measure the number of IDS alarms per hour, day, or some other time frame, and report this metric as time-series data to the human analyst. Based on IDS alarm attributes, a number of additional metrics can be defined (such as the number of botnet related alarms per time frame or the number of distinct malicious hosts per time frame). Also, it is often worthwhile to use event correlation for detecting alarm patterns that correspond to specific attacks, for example, the appearance of 7 different alarms within 60 seconds that indicate the use of a certain attack toolkit. This allows for creating metrics for these specific attacks (section IIIc provides a more detailed example on how to employ event correlation for extracting metrics from log data).

Although IDS alarm based metrics are commonly used, they are sensitive to false positives, especially if a larger volume of false positive alarms appears and the reported metric becomes seriously distorted. Furthermore, IDS signatures which detect frequent bad traffic of low importance (such as probes from well-known Internet worms) can routinely trigger many alarms over longer periods of time [16]. Such alarms form the background noise which might again distort reported metrics. Although filters for known false positives and threats of low importance can be created manually, new types of false

positives and noise might be easily introduced with signature updates and changes in the environment. In order to alleviate this problem, we have proposed a real-time IDS alarm classification algorithm during our past research which is able to distinguish false positives and noise from interesting alarms [16]. The proposed algorithm applies various data mining techniques to past IDS alarm logs, in order to learn patterns which describe noise and false positive alarms, and uses detected patterns for real-time IDS alarm classification. The learning step of the algorithm is periodically repeated (e.g., once in 24 hours), in order update the classification knowledge and adjust to changes in the surrounding environment. While our previous paper described preliminary results of using this algorithm [16], we have employed this method for several years in production. One of the purposes for introducing this method was to filter out irrelevant IDS alarms and calculate more meaningful security metrics from important alarms only.

B. Extracting Security Metrics from Netflow Data

Netflow is a protocol for collecting network traffic statistics which was developed by Cisco Systems in the 1990s. If a network device has Netflow statistics collection enabled, it will extract data from the header of each observed packet, and store these data in Netflow records. For each network flow a separate record is maintained, where the network flow is identified by the transport protocol ID, source and destination IP addresses, source and destination port numbers, and couple of other fields (such as Type of Service). Apart from transport protocol, source and destination transport addresses, each Netflow record contains a number of additional fields, including the total number of packets and bytes for the network flow, the union of all TCP flags seen in the packet headers of the flow, and the start and end times of the flow. Since collecting Netflow data in Internet backbone networks requires a lot of resources, the collection is often accomplished with sampling in such environments (e.g., only 0.01% of the packets are processed). However, in institutional networks it is often feasible to collect Netflow data without sampling which provides a detailed picture of all communications in the monitored network without storing full packet payloads.

If Netflow data are collected, they can be used for calculating a number of security metrics. Firstly, it is often useful to set up blacklist-based security metrics, in order to monitor and collect trend information on data exchange with known malicious, compromised, or suspicious peers in the Internet. Some security institutions such as EmergingThreats are maintaining publicly available blacklists of known bad hosts, including botnet C&C nodes, compromised nodes, Russian Business Network nodes, and Tor nodes (e.g., see [17]). When these blacklists are frequently downloaded and used for querying collected Netflow data sets, it is straightforward to create metrics that describe communications with malicious peers. For example, Figures 1 and 2 depict metrics which reflect daily traffic exchanged with known compromised nodes and Tor network nodes during the last 2 months (depicted metrics are collected on the outer network perimeter of a large institution).

Collected Netflow data can also be used for creating metrics for abnormal and potentially malicious network activity which supplement similar IDS alarm based metrics. For example, since a Netflow record contains a field which holds the union of all observed TCP flags for the given flow, it is straightforward to write a filtering condition for detecting flows with illegal flag combinations (e.g., TCP FIN flag never appears without TCP ACK flag in normal network traffic). Based on detected flows, metrics can be set up which describe various aspects of illegal traffic (such as the number of distinct Internet hosts per hour which are sources of abnormal traffic).

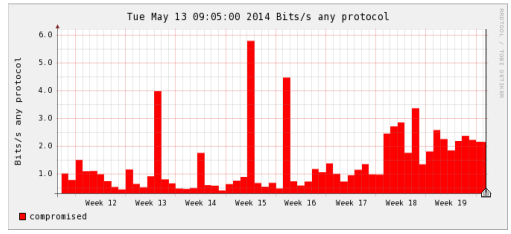

Fig. 1. Daily traffic exchanged with known compromised hosts (reflects probing activity from infected Internet hosts, but also suspicious or unwanted communications from local network to malicious hosts)

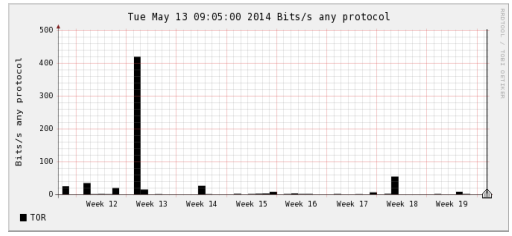

Fig. 2. Daily traffic exchanged with known Tor network hosts (reflects Tor client traffic to institutional web site and other public services, but can also reveal Tor clients in the institutional network)

Netflow statistics collection and analysis can also be employed in private networks, in order to discover illegal devices and services, malicious insiders, and infected computers, since they often manifest themselves through anomalous network traffic which differs from regular network usage patterns. Netflow based network monitoring offers some unique advantages. Firstly, it does not involve inspecting network packet payloads and consumes much less computing resources than IDS. Also, traditionally illegal devices and services are detected by scanning the entire network with dedicated tools. However, this is an expensive and time consuming procedure which might also alert the owner of illegal device or service. In contrast, Netflow based detection is stealthy and does not consume network bandwidth.

However, service detection from Netflow data involves several caveats. Due to commonly found design flaws in Netflow implementations, Netflow records might have imprecise timestamps which confuses the service detection algorithm [18, 19].

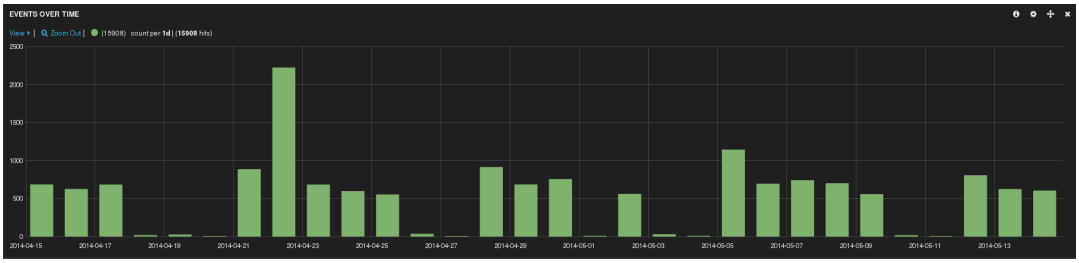

Fig. 3. Daily login failures for all institutional Windows workstations (note that unexpected spike in April 22 reflects an account probing activity by malware which infected one of the workstations, but was promptly detected and removed)

For example, if the Netflow record for server-to-client traffic is incorrectly tagged with an earlier timestamp than the record for client-to-server traffic, the server can be mistakenly taken for the client. Also, some router-based Netflow implementations might leave the "union-of-flags" field unset in some cases [18], and this exacerbates service detection further. For addressing these issues, various heuristic techniques have been suggested by us and other researchers [19, 20]. After services and hosts have been identified from Netflow data and compared with the lists of legitimate hosts and services, it is straightforward to identify illegal devices and services, and create corresponding security metrics (e.g., the number of illegal devices by organizational unit as recommended in [15]).

In order to detect anomalous network traffic in private networks which might indicate malware infection, illegal data access, or malicious insider activity, various methods can be applied to Netflow data sets. For example, if workstations in the private network are using a small set of well-known services, a simple filtering condition might be written which reports workstation traffic not related to known services (this would easily allow to find a number of network misuse cases, such as malware propagation from an infected workstation to other workstations). For more complex networks, automated methods might be used which learn normal network usage patterns from past Netflow data sets, and use detected knowledge for finding deviations from normal behavior. For example, during our past research, we have developed a method which learns and updates service usage profiles for each individual client node and the entire network, and uses these profiles for real-time detection of anomalous TCP and UDP network flows [20]. Once anomalous network flows have been detected, it is straightforward to create metrics from them (e.g., the number of anomalous TCP flows per 24 hours).

C. Extracting Security Metrics from Workstation Logs

Workstations in institutional networks are major targets for malware and targeted attacks, and therefore their monitoring and the creation of security metrics from monitoring information plays an important role. Significant amount of workstation security status information can be obtained from workstation logs, such as login failures into the workstation from console or other hosts, antivirus alerts, installation of new software and new services, alerts about modification of protected system files, etc. Unfortunately, the collection and analysis of workstation logs are often neglected – largely because workstations create large volumes of log data which makes the log collection and analysis an expensive process. A recent SANS paper by Anthony [21] proposes several strategies for setting up a centralized log collection and analysis framework for Windows workstations, and identifies a number of event types which should be collected, monitored, and correlated. In order to minimize the cost of log collection and analysis, the author of the paper proposes to send only events of few relevant types to the central collection point where they are analyzed with SEC (an event correlation tool created by one of the authors of this paper [22]). In another recent paper [23], a number of detailed recommendations are provided for monitoring Windows event logs, in order to detect adversarial activities.

One group of well-known security event types in the workstation log reflects login attempts into the local workstation. Note that these event types are not Windows-specific, but can also be easily identified for UNIX-like workstation platforms (e.g., login failures for SSH or FTP services). As discussed in [15], user account monitoring and control is one of the critical cyber defense controls. Also, the monitoring of unusual login attempts helps to detect malware propagation [21] and malicious insiders [15]. For these reasons, it makes sense to set up metrics which reflect different types of successful and failed login attempts into workstations (e.g., the number of successful logins from unexpected remote hosts per 1 hour, or the number of login failures for administrative accounts per 15 minutes). For example, Figure 3 depicts a metric which reflects daily numbers of login failures for all institutional workstations during 1 month time frame (this example metric is collected in a large institutional network from the logs of thousands of workstations). In addition to the above scenario, several other metrics could be collected from workstation logs, for example, the number of distinct workstations or accounts with login failures in a given timeframe (sudden increase in the number of hosts and accounts might indicate massive account probing over the entire network, in order to get unauthorized access to data).

Also, event correlation techniques are often useful for creating more meaningful metrics from workstation log events. For example, instead of including each accidental login failure in a relevant metric, the login failure might only be taken into account if it is not followed by a successful login within a

reasonable amount of time (e.g., 60 seconds). Figure 4 displays an example SEC event correlation rule for Linux platform which processes SSH login failure events, and sends collected metric values to Graphite reporting and visualization system.

```
# if the login failure is not followed by a successful login
# within 60 seconds, include the failure in the metric

type=PairWithWindow
ptype=RegExp
pattern=sshd\[\d+\]: Failed .+ for (?:invalid user )?(\S+) \
from ([\d.]+) port \d+ ssh2
desc=SSH login failed for user $1 from IP $2
action=lcall %count %count -> ( sub { ++$_[0] } )
ptype2=RegExp
pattern2=sshd\[\d+\]: Accepted .+ for $1 from $2 port \d+ ssh2
desc2=SSH login successful for user %1 from IP %2
action2=none
window=60

# send the current metric value to the Graphite server
# in every 5 minutes and reset the metric counter

type=Calendar
time=*/5 * * * *
desc=report SSH login failure metric once in 5 minutes
action=if %count () else ( assign %count 0 ); eval %n "\n"; \
  tcpsock graphite:2003 login.failures.ssh.total5m %count %u%n; \
  free %count
```

Fig. 4. Sample SEC ruleset for collecting the metric *number of SSH login failures for all workstations per 5 minutes*, and sending it to Graphite reporting and visualization platform

Apart from event types mentioned above, workstation logs contain a wide variety of other events which can be harnessed for creating useful security metrics. For example, Figure 5 presents two metrics which indicate daily numbers of update and patching failures for Windows operating system and Internet Explorer (depicted metrics are collected from the logs of thousands of Windows workstations of a large institution, and the metrics are used for measuring the quality of the patching process). Finally, it should be noted that the relevance of a particular event type for the metric collection process depends on the nature of the environment (e.g., in many regular networks USB insertion events are unimportant, while in classified networks they often deserve closer inspection).

D. Extracting Security Metrics from Other Logs

Security metrics can be extracted from a number of other logs, including server and router logs, firewall logs, service logs, etc. Establishing proper metrics is especially important for public services that can be accessed from the Internet. Apart from creating metrics from events which describe known security issues, one can also process both regular and unusual events. For example, while normally most HTTP client requests are for existing web pages, occasionally clients might request non-existing or forbidden pages which produce HTTP log entries with 4xx response codes. However, unexpectedly large volumes of 4xx log messages or normal 200 messages can indicate a reconnaissance scan or the start of a DDoS attack. Deriving metrics from such messages will help to assess the threat level for the service. Also, applying event correlation techniques for a service log or cross-correlating messages from different logs (e.g., service log and IDS log) is often useful for detecting advanced threats, and creating metrics for these threats (see our previous papers [22, 24] for examples on how to employ SEC for various event correlation tasks).

IV. Security Metrics Collection and Reporting Framework for Security Logs

In this section, we will describe a production framework for collecting and reporting security metrics that harnesses log analysis techniques outlined in the previous section. The framework has been set up in a large institution which is an important part of the national critical information infrastructure, and has a complex organizational network consisting of many thousands of workstations, servers, network devices, IDSs, firewalls, and other nodes.

As discussed in section I, the collection of security metrics should not involve significant cost impact, and it should be preferably done with an automated collection system. In order to address these requirements, our framework is centralized, since analyzing security logs locally at workstations, servers, and other nodes would impose additional load on them, and interfere with normal system activities. Also, decentralized log analysis would considerably increase the complexity of the metrics collection system. For reducing the implementation costs, our framework is based on open-source solutions.

Our centralized framework is receiving security log data from all relevant nodes in the organizational network over the *syslog* and Netflow protocols. For events which are not natively in *syslog* format, appropriate format conversion gateways are used (e.g., Windows EventLog messages are converted to *syslog* format with nxlog [25]). Incoming *syslog* and Netflow data are received by several central log collection servers which are running SEC for correlating incoming *syslog* events.

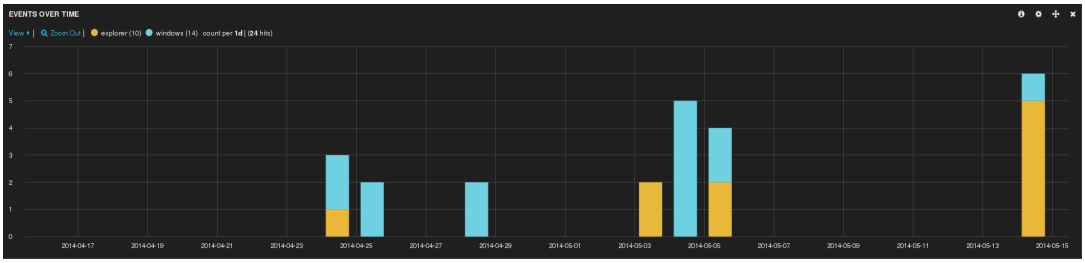

Fig. 5. Daily updating and patching failures of Windows operating system and Internet Explorer for all institutional Windows workstations

During the event correlation, a number of security metrics are extracted and sent to Graphite reporting and visualization system. Graphite [26] has been specifically designed for performing computations on time-series data, and generating wide variety of graphs and reports from computation results. Also, Netflow data are sent to NfSen [27] which is a flexible visualization tool for Netflow with drill-down capabilities (see Figures 1-2 for example metric reports generated with NfSen).

From log collection servers, all *syslog* events and Netflow data are forwarded to a central Elasticsearch [28] database cluster which the end users are accessing through Kibana visualization interface [29]. Currently, almost 100 million security log records are stored in Elasticsearch on a daily basis. In order to receive, parse, and store these volumes of data, we are using rsyslog [30] and logstash [31] (rsyslog is one of the most efficient *syslog* servers with Elasticsearch support, while logstash supports flexible parsing of *syslog* and Netflow data [32]). In Kibana, more than 20 dashboards have been set up for displaying various security metrics (Figures 3 and 5 display two metric report examples). All reports generated with Kibana are interactive and allow for drilling down to more specific reports and individual log records. Therefore, after spotting an anomaly in a metric report, the root cause events for this anomaly can be quickly identified.

V. Conclusion and Future Work

In this paper, we have discussed the use of security logs for collecting and reporting of security metrics, and have reviewed a number of metrics collection scenarios for common security log types. Also, we have described a production framework for metrics collection and reporting which is based on open-source log management technologies. For the future work, we plan to research log analysis methods for insider threat detection, and to implement relevant algorithms within our framework.

Acknowledgment

The authors express their gratitude to Mr. Kaido Raiend and Mr. Ain Rasva from SEB Estonia for supporting this work.

References

[1] B. Alshammari, C. Fidge, and D. Corney, “Security Metrics for Object-Oriented Class Designs,” in *Proceedings of 2009 International Conference on Quality Software*, pp. 11-20

[2] T. Heyman, R. Scandariato, C. Huygens, and W. Joosen, “Using security patterns to combine security metrics,” in *Proceedings of 2008 International Conference on Availability, Reliability and Security*, pp. 1156-1163

[3] R. M. Savola and P. Heinonen, “A Visualization and Modeling Tool for Security Metrics and Measurements Management,” in *Proceedings of 2011 Information Security for South Africa Conference*, pp. 1-8

[4] R. Barabanov, S. Kowalski, and L. Yngström, “Information Security Metrics: Research Directions,” University of Stockholm, Technical Report, 2011

[5] W. Boyer and M. McQueen, “Ideal Based Cyber Security Technical Metrics for Control Systems,” in *Proceedings of 2007 International Conference on Critical Information Infrastructures Security*, pp. 246-260

[6] C. A. da Silva, A. S. Ferreira, and P. L. de Geus, “A Methodology for Management of Cloud Computing using Security Criteria,” in *Proceedings of 2012 IEEE Latin American Conference on Cloud Computing*, pp. 49-54

[7] R. A. Kisner, W. W. Manges, L. P. MacIntyre, J. J. Nutaro, J. K. Munro, P. D. Ewing, M. Howlander, P. T. Kuruganti, R. M. Wallace, and M. M. Olama, “Cybersecurity through Real-Time Distributed Control Systems,” Oak Ridge National Laboratory, Technical Report ORNL/TM-2010/30, February 2010

[8] A. McIntyre, B. Becker, and R. Halbgewachs, “Security Metrics for Process Control Systems,” Sandia National Laboratories, Sandia Report SAND2007-2070P, September 2007

[9] M. Mateski, C. M. Trevino, C. K. Veitch, J. Michalski, J. M. Harris, S. Maruoka, and J. Frye, “Cyber Threat Metrics,” Sandia National Laboratories, Sandia Report SAND2012-2427, March 2012

[10] ISO/IEC 27004:2009 standard “Information technology -- Security techniques -- Information security management -- Measurement”, 2009

[11] A. Jaquith, *Security Metrics: Replacing Fear, Uncertainty, and Doubt*. Addison-Wesley, 2007

[12] W. K. Brotby and G. Hinson, *PRAGMATIC Security Metrics: Applying Metametrics to Information Security*. Auerbach Publications, 2013

[13] P. E. Black, K. Scarfone, and M. Souppaya, “Cyber Security Metrics and Measures,” in *Wiley Handbook of Science and Technology for Homeland Security*, John Wiley and Sons, 2009

[14] “The CIS Security Metrics,” The Center for Internet Security, Technical Report, version 1.1.0, November 1 2010

[15] “The Critical Controls for Effective Cyber Defense,” Council on CyberSecurity, Technical Report, version 5.0, 2014

[16] R. Vaarandi and K. Podiņš, “Network IDS Alert Classification with Frequent Itemset Mining and Data Clustering,” in *Proceedings of the 2010 IEEE Conference on Network and Service Management*, pp. 451-456

[17] http://rules.emergingthreats.net/fwrules/emerging-Block-IPs.txt

[18] R. Hofstede, I. Drago, A. Sperotto, R. Sadre, and A. Pras, “Measurement Artifacts in NetFlow Data,” in *Proceedings of the 2013 Passive and Active Measurement Conference*, pp. 1-10

[19] B. Trammell, B. Tellenbach, D. Schatzmann, and M. Burkhardt, “Peeling Away Timing Error in NetFlow Data,” in *Proceedings of the 2011 Passive and Active Measurement Conference*, pp. 194-203

[20] R. Vaarandi, “Detecting Anomalous Network Traffic in Organizational Private Networks,” in *Proceedings of the 2013 IEEE CogSIMA Conference*, pp. 285-292

[21] R. Anthony, “Detecting Security Incidents Using Windows Workstation Event Logs,” SANS Institute, InfoSec Reading Room Paper, June 19 2013

[22] R. Vaarandi, “Simple Event Correlator for real-time security log monitoring,” Hakin9 Magazine, vol. 1/2006 (6), pp. 28-39, 2006

[23] “Spotting the Adversary with Windows Event Log Monitoring,” National Security Agency/Central Security Service, Information Assurance Directorate, Technical Report, Revision 2, December 16 2013

[24] R. Vaarandi and M. R. Grimaila, “Security Event Processing with Simple Event Correlator,” Information Systems Security Association Journal, vol. 10(8), pp. 30-37, 2012

[25] http://nxlog.org

[26] http://graphite.readthedocs.org

[27] http://nfsen.sourceforge.net

[28] http://www.elasticsearch.org

[29] http://www.elasticsearch.org/overview/kibana/

[30] http://www.rsyslog.com

[31] http://logstash.net

[32] R. Vaarandi and P. Niziński, "Comparative Analysis of Open-Source Log Management Solutions for Security Monitoring and Network Forensics," in *Proceedings of the 2013 European Conference on Information Warfare and Security*, pp. 278-287

Appendix 2

Publication II

R. Vaarandi and M. Pihelgas. LogCluster - A data clustering and pattern mining algorithm for event logs. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 1–7, November 2015

The paper is included in the *Proceedings of the 11th International Conference on Network and Service Management (CNSM 2015)*, ISBN: 978-3-901882-77-7.
DOI: 10.1109/CNSM.2015.7367331.

LogCluster - A Data Clustering and Pattern Mining Algorithm for Event Logs

Risto Vaarandi and Mauno Pihelgas
TUT Centre for Digital Forensics and Cyber Security
Tallinn University of Technology
Tallinn, Estonia
firstname.lastname@ttu.ee

***Abstract*—Modern IT systems often produce large volumes of event logs, and event pattern discovery is an important log management task. For this purpose, data mining methods have been suggested in many previous works. In this paper, we present the LogCluster algorithm which implements data clustering and line pattern mining for textual event logs. The paper also describes an open source implementation of LogCluster.**

Keywords—event log analysis; mining patterns from event logs; event log clustering; data clustering; data mining

I. INTRODUCTION

During the last decade, data centers and computer networks have grown significantly in processing power, size, and complexity. As a result, organizations commonly have to handle many gigabytes of log data on a daily basis. For example, in our recent paper we have described a security log management system which receives nearly 100 million events each day [1]. In order to ease the management of log data, many research papers have suggested the use of data mining methods for discovering event patterns from event logs [2–20]. This knowledge can be employed for many different purposes like the development of event correlation rules [12–16], detection of system faults and network anomalies [6–9, 19], visualization of relevant event patterns [17, 18], identification and reporting of network traffic patterns [4, 20], and automated building of IDS alarm classifiers [5].

In order to analyze large amounts of textual log data without well-defined structure, several data mining methods have been proposed in the past which focus on the detection of line patterns from textual event logs. Suggested algorithms have been mostly based on data clustering approaches [2, 6, 7, 8, 10, 11]. The algorithms assume that each event is described by a single line in the event log, and each line pattern represents a group of similar events.

In this paper, we propose a novel data clustering algorithm called LogCluster which discovers both frequently occurring line patterns and outlier events from textual event logs. The remainder of this paper is organized as follows – section II provides an overview of related work, section III presents the LogCluster algorithm, section IV describes the LogCluster prototype implementation and experiments for evaluating its performance, and section V concludes the paper.

This work has been supported by Estonian IT Academy (StudyITin.ee) and SEB Estonia.

978-3-901882-77-7 © 2015 IFIP

II. RELATED WORK

One of the earliest event log clustering algorithms is SLCT that is designed for mining line patterns and outlier events from textual event logs [2]. During the clustering process, SLCT assigns event log lines that fit the same pattern (e.g., *Interface * down*) to the same cluster, and all detected clusters are reported to the user as line patterns. For finding clusters in log data, the user has to supply the support threshold value *s* to SLCT which defines the minimum number of lines in each cluster. SLCT begins the clustering with a pass over the input data set, in order to identify frequent words which occur at least in *s* lines (word delimiter is customizable and defaults to whitespace). Also, each word is considered with its position in the line. For example, if *s=2* and the data set contains the lines

Interface eth0 down

Interface eth1 down

Interface eth2 up

then words *(Interface,1)* and *(down,3)* occur in three and two lines, respectively, and are thus identified as frequent words. SLCT will then make another pass over the data set and create cluster candidates. When a line is processed during the data pass, all frequent words from the line are joined into a set which will act as a candidate for this line. After the data pass, candidates generated for at least *s* lines are reported as clusters together with their supports (occurrence times). Outliers are identified during an optional data pass and written to a user-specified file. For example, if *s=2* then two cluster candidates *{(Interface,1), (down,3)}* and *{(Interface,1)}* are detected with supports 2 and 1, respectively. Thus, *{(Interface,1), (down,3)}* is the only cluster and is reported to the user as a line pattern *Interface * down* (since there is no word associated with the second position, an asterisk is printed for denoting a wildcard). Reported cluster covers the first two lines, while the line *Interface eth2 up* is considered an outlier.

SLCT has several shortcomings which have been pointed out in some recent works. Firstly, it is not able to detect wildcards after the last word in a line pattern [11]. For instance, if *s=3* for three example lines above, the cluster *{(Interface,1)}* is reported to the user as a line pattern *Interface*, although most users would prefer the pattern *Interface * **. Secondly, since word positions are encoded into words, the algorithm is

sensitive to shifts in word positions and delimiter noise [8]. For instance, the line *Interface HQ Link down* would not be assigned to the cluster *Interface * down*, but would rather generate a separate cluster candidate. Finally, low support thresholds can lead to overfitting when larger clusters are split and resulting patterns are too specific [2].

Reidemeister, Jiang, Munawar and Ward [6, 7, 8] developed a methodology that addresses some of the above shortcomings. The methodology uses event log mining techniques for diagnosing recurrent faults in software systems. First, a modified version of SLCT is used for mining line patterns from labeled event logs. In order to handle clustering errors caused by shifts in word positions and delimiter noise, line patterns from SLCT are clustered with a single-linkage clustering algorithm which employs a variant of the Levenshtein distance function. After that, a common line pattern description is established for each cluster of line patterns. According to [8], single-linkage clustering and post-processing its results add minimal runtime overhead to the clustering by SLCT. The final results are converted into bit vectors and used for building decision-tree classifiers, in order to identify recurrent faults in future event logs.

Another clustering algorithm that mines line patterns from event logs is IPLoM by Makanju, Zincir-Heywood and Milios [10, 11]. Unlike SLCT, IPLoM is a hierarchical clustering algorithm which starts with the entire event log as a single partition, and splits partitions iteratively during three steps. Like SLCT, IPLoM considers words with their positions in event log lines, and is therefore sensitive to shifts in word positions. During the first step, the initial partition is split by assigning lines with the same number of words to the same partition. During the second step, each partition is divided further by identifying the word position with the least number of unique words, and splitting the partition by assigning lines with the same word to the same partition. During the third step, partitions are split based on associations between word pairs. At the final stage of the algorithm, a line pattern is derived for each partition. Due to its hierarchical nature, IPLoM does not need the support threshold, but takes several other parameters (such as partition support threshold and cluster goodness threshold) which impose fine-grained control over splitting of partitions [11]. As argued in [11], one advantage of IPLoM over SLCT is its ability to detect line patterns with wildcard tails (e.g., *Interface * **), and the author has reported higher precision and recall for IPLoM.

III. The LogCluster Algorithm

The LogCluster algorithm is designed for addressing the shortcomings of existing event log clustering algorithms that were discussed in the previous section. Let $L = \{l_1,...,l_n\}$ be a textual event log which consists of n lines, where each line l_i $(1 \le i \le n)$ is a complete representation of some event and i is a unique line identifier. We assume that each line $l_i \in L$ is a sequence of k_i words: $l_i = (w_{i,1},...,w_{i,k_i})$. LogCluster takes the *support threshold* s $(1 \le s \le n)$ as a user given input parameter and divides event log lines into clusters $C_1,...,C_m$, so that there are at least s lines in each cluster C_j (i.e., $|C_j| \ge s$) and O is the cluster of outliers: $L = C_1 \cup ... \cup C_m \cup O$, $O \cap C_j = \varnothing$, $1 \le j \le m$. LogCluster views the log clustering problem as a pattern mining problem – each cluster C_j is uniquely identified by its line pattern p_j which matches all lines in the cluster, and in order to detect clusters, LogCluster mines line patterns p_j from the event log. The *support* of pattern p_j and cluster C_j is defined as the number of lines in C_j: $supp(p_j) = supp(C_j) = |C_j|$. Each pattern consists of words and wildcards, e.g., *Interface *{1,3} down* has words *Interface* and *down*, and wildcard *{1,3} that matches at least 1 and at most 3 words.

In order to find patterns that have the support s or higher, LogCluster relies on the following observation – all words of such patterns must occur at least in s event log lines. Therefore, LogCluster begins its work with the identification of such words. However, unlike SLCT and IPLoM, LogCluster considers each word without its position in the event log line. Formally, let I_w be the set of identifiers of lines that contain the word w: $I_w = \{i \mid l_i \in L,\ 1 \le i \le n,\ \exists j\ w_{i,j} = w,\ 1 \le j \le k_i\}$. The word w is *frequent* if $|I_w| \ge s$, and the set of all frequent words is denoted by F. According to [2, 3], large event logs often contain many millions of different words, while vast majority of them appear only few times in event logs. In order to take advantage of this property for reducing its memory footprint, LogCluster employs a sketch of h counters $c_0,...,c_{h-1}$. During a preliminary pass over event log L, each unique word of every event log line is hashed to an integer from 0 to h-1, and the corresponding sketch counter is incremented. Since the hashing function produces output values 0…h-1 with equal probabilities, each sketch counter reflects the sum of occurrence times of approximately d / h words, where d is the number of unique words in L. However, since most words appear in only few lines of L, most sketch counters will be smaller than support threshold s after the data pass. Thus, corresponding words cannot be frequent, and can be ignored during the following pass over L for finding frequent words.

After frequent words have been identified, LogCluster makes another pass over event log L and creates cluster candidates. For each line in the event log, LogCluster extracts all frequent words from the line and arranges the words as a tuple, retaining their original order in the line. The tuple will serve as an identifier of the cluster candidate, and the line is assigned to this candidate. If the given candidate does not exist, it is initialized with the support counter set to 1, and its line pattern is created from the line. If the candidate exists, its support counter is incremented and its line pattern is adjusted to cover the current line. Note that LogCluster does not memorize individual lines assigned to a cluster candidate.

For example, if the event log line is *Interface DMZ-link down at node router2*, and words *Interface*, *down*, *at*, and *node* are frequent, the line is assigned to the candidate identified by the tuple *(Interface, down, at, node)*. If this candidate does not exist, it will be initialized by setting its line pattern to *Interface *{1,1} down at node *{1,1}* and its support counter to 1 (wildcard *{1,1} matches any single word). If the next line which produces the same candidate identifier is *Interface HQ link down at node router2*, the candidate support counter is incremented to 2. Also, its line pattern is set to *Interface *{1,2} down at node *{1,1}*, making the pattern to match at least one but not more than two words between *Interface* and *down*. Fig. 1 describes the candidate generation procedure in full details.

```
Procedure: Generate_Candidates
Input: event log L = {l1,…,ln}
       set of frequent words F
Output: set of cluster candidates X

X := ∅
for (id = 1; id <= n; ++id) do
  tuple := ()
  vars := ()
  i := 0; v := 0
  for each w in (wid,1,…,wid,kid) do
    if (w ∈ F) then
      tuple[i] := w
      vars[i] := v
      ++i; v := 0
    else
      ++v
    fi
  done
  vars[i] := v
  k := # of elements in tuple
  if (k > 0) then
    if (∃Y ∈ X, Y.tuple == tuple) then
      ++Y.support
      for (i := 0; i < k+1; ++i) do
        if (Y.varmin[i] > vars[i]) then
          Y.varmin[i] := vars[i]
        fi
        if (Y.varmax[i] < vars[i]) then
          Y.varmax[i] := vars[i]
        fi
      done
    else
      initialize new candidate Y
      Y.tuple := tuple
      Y.support := 1
      for (i := 0; i < k+1; ++i) do
        Y.varmin[i] := vars[i]
        Y.varmax[i] := vars[i]
      done
      X := X ∪ { Y }
    fi
    Y.pattern = ()
    j: = 0
    for (i := 0; i < k; ++i) do
      if (Y.varmax[i] > 0) then
        min := Y.varmin[i]
        max := Y.varmax[i]
        Y.pattern[j] := "*{min,max}"
        ++j
      fi
      Y.pattern[j] := tuple[i]
      ++j
    done
    if (Y.varmax[k] > 0) then
      min := Y.varmin[k]
      max := Y.varmax[k]
      Y.pattern[j] := "*{min,max}"
    fi
  fi
done
return X
```

Fig. 1. Candidate generation procedure of LogCluster.

After the data pass for generating cluster candidates is complete, LogCluster drops all candidates with the support counter value smaller than support threshold *s*, and reports remaining candidates as clusters. For each cluster, its line pattern and support are reported, while outliers are identified during additional pass over event log *L*. Due to the nature of its frequent word detection and candidate generation procedures, LogCluster is not sensitive to shifts in word positions and is able to detect patterns with wildcard tails.

When pattern mining is conducted with lower support threshold values, LogCluster is (similarly to SLCT) prone to overfitting – larger clusters might be split into smaller clusters with too specific line patterns. For example, the cluster with a pattern *Interface *{1,1} down* could be split into clusters with patterns *Interface *{1,1} down*, *Interface eth1 down*, and *Interface eth2 down*. Furthermore, meaningful generic patterns (e.g., *Interface *{1,1} down*) might disappear during cluster splitting. In order to address the overfitting problem, LogCluster employs two optional heuristics for increasing the support of more generic cluster candidates and for joining clusters. The first heuristic is called *Aggregate_Supports* and is applied after the candidate generation procedure has been completed, immediately before clusters are selected. The heuristic involves finding candidates with more specific line patterns for each candidate, and adding supports of such candidates to the support of the given candidate. For instance, if candidates *User bob login from 10.1.1.1*, *User *{1,1} login from 10.1.1.1*, and *User *{1,1} login from *{1,1}* have supports 5, 10, and 100, respectively, the support of the candidate *User *{1,1} login from *{1,1}* will be increased to 115. In other words, this heuristic allows clusters to overlap.

The second heuristic is called *Join_Clusters* and is applied after clusters have been selected from candidates. For each frequent word $w \in F$, we define the set C_w as follows: $C_w = \{f \mid f \in F, I_w \cap I_f \neq \emptyset\}$ (i.e., C_w contains all frequent words that co-occur with w in event log lines). If $w' \in C_w$ (i.e., w' co-occurs with w), we define *dependency* from w to w' as $dep(w, w') = |I_w \cap I_{w'}| / |I_w|$. In other words, $dep(w, w')$ reflects how frequently w' occurs in lines which contain w. Also, note that $0 < dep(w, w') \leq 1$. If $w_1,\ldots,w_k$ are frequent words of a line pattern (i.e., the corresponding cluster is identified by the tuple $(w_1,\ldots,w_k)$), the *weight* of the word w_i in this pattern is calculated as follows: $weight(w_i) = \sum_{j=1}^{k} dep(w_j, w_i) / k$. Note that since $dep(w_i, w_i) = 1$, then $1/k \leq weight(w_i) \leq 1$. Intuitively, the weight of the word indicates how strongly correlated the word is with other words in the pattern. For example, suppose the line pattern is *Daemon testd killed*, and words *Daemon* and *killed* always appear together, while the word *testd* never occurs without *Daemon* and *killed*. Thus, *weight(Daemon)* and *weight(killed)* are both 1. Also, if only 2.5% of lines that contain both *Daemon* and *killed* also contain *testd*, then *weight(testd)* = (1 + 0.025 + 0.025) / 3 = 0.35. (We plan to implement more weight functions in the future versions of the LogCluster prototype.)

The *Join_Clusters* heuristic takes the user supplied word weight threshold t as its input parameter ($0 < t \leq 1$). For each cluster, a secondary identifier is created and initialized to the cluster's regular identifier tuple. Also, words with weights smaller than t are identified in the cluster's line pattern, and each such word is replaced with a special token in the secondary identifier. Finally, clusters with identical secondary identifiers are joined. When two or more clusters are joined, the support of the joint cluster is set to the sum of supports of original clusters, and the line pattern of the joint cluster is adjusted to represent the lines in all original clusters.

```
Procedure: Join_Clusters
Input: set of clusters C = {C1,…,Cp}
       word weight threshold t
       word weight function W()
Output: set of clusters C' = {C'1,…,C'm}, m ≤ p

C' := ∅
for (j = 1; j <= p; ++j) do
  tuple := Cj.tuple
  k := # of elements in tuple
  for (i := 0; i < k; ++i) do
    if (W(tuple, i) < t) then
      tuple[i] := TOKEN
    fi
  done
  if (∃Y ∈ C', Y.tuple == tuple) then
    Y.support := Y.support + Cj.support
    for (i := 0; i < k+1; ++i) do
      if (Y.varmin[i] > Cj.varmin[i]) then
        Y.varmin[i] := Cj.varmin[i]
      fi
      if (Y.varmax[i] < Cj.varmax[i]) then
        Y.varmax[i] := Cj.varmax[i]
      fi
    done
  else
    initialize new cluster Y
    Y.tuple := tuple
    Y.support := Cj.support
    for (i := 0; i < k+1; ++i) do
      Y.varmin[i] := Cj.varmin[i]
      Y.varmax[i] := Cj.varmax[i]
      if (i < k AND Y.tuple[i] == TOKEN) then
        Y.wordlist[i] := ∅
      fi
    done
    C' := C' ∪ { Y }
  fi
  Y.pattern := ()
  j: = 0
  for (i := 0; i < k; ++i) do
    if (Y.varmax[i] > 0) then
      min := Y.varmin[i]
      max := Y.varmax[i]
      Y.pattern[j] := "*{min,max}"
      ++j
    fi
    if (Y.tuple[i] == TOKEN) then
      if (Cj.tuple[i] ∉ Y.wordlist[i]) then
        Y.wordlist[i] :=
          Y.wordlist[i] ∪ { Cj.tuple[i] }
      fi
      Y.pattern[j] := "( elements of
          Y.wordlist[i] separated by | )"
    else
      Y.pattern[j] := Y.tuple[i]
    fi
    ++j
  done
  if (Y.varmax[k] > 0) then
    min := Y.varmin[k]
    max := Y.varmax[k]
    Y.pattern[j] := "*{min,max}"
  fi
done
return C'
```

Fig. 2. Cluster joining heuristic of LogCluster.

For example, if two clusters have patterns *Interface *{1,1} down at node router1* and *Interface *{2,3} down at node router2*, and words *router* and *router2* have insufficient weights, the clusters are joined into a new cluster with the line pattern *Interface *{1,3} down at node (router1|router2)*. Fig. 2 describes the details of the *Join_Clusters* heuristic. Since the line pattern of a joint cluster consists of strongly correlated words, it is less likely to suffer from overfitting. Also, words with insufficient weights are incorporated into the line pattern as lists of alternatives, representing the knowledge from original patterns in a compact way without data loss. Finally, joining clusters will reduce their number and will thus make cluster reviewing easier for the human expert.

Fig. 3 summarizes all techniques presented in this section and outlines the LogCluster algorithm. In the next section, we describe the LogCluster implementation and its performance.

IV. LogCluster Implementation and Performance

For assessing the performance of the LogCluster algorithm, we have created its publicly available GNU GPLv2 licensed prototype implementation in Perl. The implementation is a UNIX command line tool that can be downloaded from *http://ristov.github.io/logcluster*. Apart from its clustering capabilities, the LogCluster tool supports a number of data preprocessing options which are summarized below. In order to focus on specific lines during pattern mining, a regular expression filter can be defined with the *--lfilter* command line option. For instance, with *--lfilter='sshd\[\d+\]:'* patterns are detected for *sshd* syslog messages (e.g., *May 10 11:07:12 myhost sshd[4711]: Connection from 10.1.1.1 port 5662*).

```
Procedure: LogCluster
Input: event log L = {l1,…,ln}
       support threshold s
       word sketch size h  (optional)
       word weight threshold t  (optional)
       word weight function W()  (optional)
       boolean for invoking Aggregate_Supports
                 procedure A  (optional)
       file of outliers ofile  (optional)
Output: set of clusters C = {C1,…,Cm}
        the cluster of outliers O (optional)

1. if (defined(h)) then
   make a pass over L and build the word sketch
   of size h for filtering out infrequent words
   at step 2
2. make a pass over L and find the set of
   frequent words:  F := {w | |Iw| ≥ s}
3. if (defined(t)) then
   make a pass over L and find dependencies for
   frequent words: {dep(w, w') | w ∈ F, w' ∈ Cw}
4. make a pass over L and find the set of cluster
   candidates X:  X := Generate_Candidates(L, F)
5. if (defined(A) AND A == TRUE) then
   invoke Aggregate_Supports() procedure
6. find the set of clusters C
   C := {Y ∈ X | supp(Y) ≥ s}
7. if (defined(t)) then
   join clusters:  C := Join_Clusters(C, t, W)
8. report line patterns and their supports
   for clusters from set C
9. if (defined(ofile)) then
   make a pass over L and write outliers to ofile
```

Fig. 3. The LogCluster algorithm.

If a template string is given with the *--template* option, match variables set by the regular expression of the *--lfilter* option are substituted in the template string, and the resulting string replaces the original event log line during the mining. For example, with the use of *--lfilter='(sshd\[\d+\]: .+)'* and *--template='$1'* options, timestamps and hostnames are removed from *sshd* syslog messages before any other processing. If a regular expression is given with the *--separator* option, any sequence of characters that matches this expression is treated as a word delimiter (word delimiter defaults to whitespace).

Existing line pattern mining tools treat words as atoms during the mining process, and make no attempt to discover potential structure inside words (the only exception is SLCT which includes a simple post-processing option for detecting constant heads and tails for wildcards). In order to address this shortcoming, LogCluster implements several options for masking specific word parts and creating word classes. If a word matches the regular expression given with the *--wfilter* option, a word class is created for the word by searching it for substrings that match another regular expression provided with the *--wsearch* option. All matching substrings are then replaced with the string specified with the *--wreplace* option. For example, with the use of *--wfilter='=', --wsearch='=.+'*, and *--wreplace='=VALUE'* options, word classes are created for words which contain the equal sign (=) by replacing the characters after the equal sign with the string *VALUE*. Thus, for words *pid=12763* and *user=bob*, classes *pid=VALUE* and *user=VALUE* are created. If a word is infrequent but its word class is frequent, the word class replaces the word during the mining process and will be treated like a frequent word. Since classes can represent many infrequent words, their presence in line patterns provides valuable information about regularities in word structure that would not be detected otherwise.

For evaluating the performance of LogCluster and comparing it with other algorithms, we conducted a number of experiments with larger event logs. For the sake of fair comparison, we re-implemented the public C-based version of SLCT in Perl. Since the implementations of IPLoM and the algorithm by Reidemeister et al. are not publicly available, we were unable to study their source code for creating their exact prototypes. However, because the algorithm by Reidemeister et al. uses SLCT and has a similar time complexity (see section II), its runtimes are closely approximated by results for SLCT. During our experiments, we used 6 logs from a large institution of a national critical information infrastructure of an EU state. The logs cover 24 hour timespan (May 8, 2015), and originate from a wide range of sources, including database systems, web proxies, mail servers, firewalls, and network devices. We also used an availability monitoring system event log from the NATO CCD COE Locked Shields 2015 cyber defense exercise which covers the entire two-day exercise and contains Nagios events. During the experiments, we clustered each log file three times with support thresholds set to 1%, 0.5% and 0.1% of lines in the log. We also used the word sketch of 100,000 counters (parameter *h* in Fig. 3) for both LogCluster and SLCT, and did not employ *Aggregate_Supports* and *Join_Clusters* heuristics. Therefore, both LogCluster and SLCT were configured to make three passes over the data set, in order to build the word sketch during the first pass, detect frequent words during the second pass, and generate cluster candidates during the third pass. All experiments were conducted on a Linux virtual server with Intel Xeon E5-2680 CPU and 64GB of memory, and Table I outlines the results. Since LogCluster and SLCT implementations are both single-threaded and their CPU utilization was 100% according to Linux *time* utility during all 21 experiments, each runtime in Table I closely matches the consumed CPU time.

TABLE I. PERFORMANCE OF LOGCLUSTER AND SLCT

Row #	Event log type	Event log size in megabytes	Event log size in lines	Support threshold	Number of clusters found by LogCluster	LogCluster runtime in seconds	Number of clusters found by SLCT	SLCT runtime in seconds
1	Authorization messages	3800.1	7,757,440	7,757	49	3146.42	89	1969.04
2	Authorization messages	3800.1	7,757,440	38,787	32	3070.18	37	1892.41
3	Authorization messages	3800.1	7,757,440	77,574	9	3050.20	15	1911.93
4	UNIX daemon messages	740.2	5,778,847	5,778	150	692.08	158	479.90
5	UNIX daemon messages	740.2	5,778,847	28,894	40	682.95	44	462.85
6	UNIX daemon messages	740.2	5,778,847	57,788	12	667.82	16	470.48
7	Application messages	9363.0	34,516,290	34,516	109	5225.32	114	3674.47
8	Application messages	9363.0	34,516,290	172,581	16	4891.51	25	3559.36
9	Application messages	9363.0	34,516,290	345,162	5	4765.09	8	3517.67
10	Network device messages	4705.0	12,522,620	12,522	193	3181.97	195	2015.52
11	Network device messages	4705.0	12,522,620	62,613	31	3083.16	33	2000.98
12	Network device messages	4705.0	12,522,620	125,226	17	3080.66	19	1945.69
13	Web proxy messages	16681.5	49,376,464	49,376	105	8487.37	111	5409.23
14	Web proxy messages	16681.5	49,376,464	246,882	14	8128.34	14	5277.54
15	Web proxy messages	16681.5	49,376,464	493,764	5	8081.30	5	5244.96
16	Mail server messages	246.0	1,230,532	1,230	129	144.42	139	96.34
17	Mail server messages	246.0	1,230,532	6,152	40	141.83	40	96.85
18	Mail server messages	246.0	1,230,532	12,305	21	142.34	23	94.12
19	Nagios messages	391.9	3,400,185	3,400	45	435.76	46	316.77
20	Nagios messages	391.9	3,400,185	17,000	39	412.08	41	320.26
21	Nagios messages	391.9	3,400,185	34,001	19	409.87	22	318.25

```
May 8 *{1,1} myserver dhcpd: DHCPREQUEST for
*{1,2} from *{1,2} via *{1,4}

May 8 *{3,3} Note: no *{1,3} sensors

May 8 *{3,3} RT_IPSEC: %USER-3-RT_IPSEC_REPLAY:
Replay packet detected on IPSec tunnel on *{1,1}
with tunnel ID *{1,1} From *{1,1} to *{1,1} ESP,
SPI *{1,1} SEQ *{1,1}

May 8 *{1,1} myserver httpd: client *{1,1} request
GET *{1,1} HTTP/1.1 referer *{1,1} User-agent
Mozilla/5.0 *{3,4} rv:37.0) Gecko/20100101
Firefox/37.0 *{0,1}

May 8 *{1,1} myserver httpd: client *{1,1} request
GET *{1,1} HTTP/1.1 referer *{1,1} User-agent
Mozilla/5.0 (Windows NT *{1,3} AppleWebKit/537.36
(KHTML, like Gecko) Chrome/42.0.2311.135
Safari/537.36
```

Fig. 4. Sample clusters detected by LogCluster (for the reasons of privacy, sensitive data have been obfuscated).

As results indicate, SLCT was 1.28–1.62 times faster than LogCluster. This is due to the simpler candidate generation procedure of SLCT – when processing individual event log lines, SLCT does not have to check the line patterns of candidates and adjust them if needed. However, both algorithms require considerable amount of time for clustering very large log files. For example, for processing the largest event log of 16.3GB (rows 13-15 in Table I), SLCT needed about 1.5 hours, while for LogCluster the runtime exceeded 2 hours. In contrast, the C-based version of SLCT accomplishes the same three tasks in 18-19 minutes. Therefore, we expect a C implementation of LogCluster to be significantly faster.

According to Table I, LogCluster finds less clusters than SLCT during all experiments (some clusters are depicted in Fig. 4). The reviewing of detected clusters revealed that unlike SLCT, LogCluster was able to discover a single cluster for lines where frequent words were separated with a variable number of infrequent words. For example, the first cluster in Fig. 4 properly captures all DHCP request events. In contrast, SLCT discovered two clusters *May 8 * myserver dhcpd: DHCPREQUEST for * from * * via* and *May 8 * myserver dhcpd: DHCPREQUEST for * * from * * via* which still do not cover all possible event formats. Also, the last two clusters in Fig. 4 represent all HTTP requests originating from the latest stable versions of Firefox browser on all OS platforms and Chrome browser on all Windows platforms, respectively (all OS platform strings are matched by **{3,4}* for Firefox, while *Windows NT *{1,3}* matches all Windows platform strings for Chrome). Like in the previous case, SLCT was unable to discover equivalent two clusters that would concisely capture HTTP request events for these two browser types.

When evaluating the *Join_Clusters* heuristic, we found that word weight thresholds (parameter *t* in Fig. 3) between 0.5 and 0.8 produced the best joint clusters. Fig. 5 displays three sample joint clusters which were detected from the mail server and Nagios logs (rows 16-21 in Table I). Fig. 5 also illustrates data preprocessing capabilities of the LogCluster tool. For the mail server log, a word class is created for each word which contains punctuation marks, so that all sequences of non-punctuation characters which are not followed by the equal sign (=) or opening square bracket ([) are replaced with a single *X* character. For the Nagios log, word classes are employed for masking blue team numbers in host names, and also, trailing timestamps are removed from each event log line with *--lfilter* and *--template* options. The first two clusters in Fig. 5 are both created by joining three clusters, while the last cluster is the union of twelve clusters which represent Nagios SSH service check events for 192 servers.

```
logcluster.pl --support=12305 \
--input=mail.log --wfilter='[[:punct:]]' \
--wsearch='[^[:punct:]]++(?![[=])' \
--wreplace=X --wweight=0.75

May 8 X:X:X (myserver1|myserver2|myserver3)
sendmail[X]: STARTTLS=client,
(relay=relayserver1,|relay=relayserver2,
|relay=relayserver3,) version=TLSv1/SSLv3,
(verify=FAIL,|verify=OK,) (cipher=DHE-RSA-AES256-
SHA,|cipher=AES128-SHA,|cipher=RC4-SHA,)
(bits=256/256|bits=128/128)

May 8 X:X:X (myserver1|myserver2|myserver3)
sendmail[X]: X: from=<myrobot@mydomain>, size=X,
class=0, nrcpts=1, msgid=<X.X@X.X>,
bodytype=8BITMIME, proto=ESMTP, daemon=MTA,
(relay=relayserver1|relay=relayserver2)
([ipaddress1]|[ipaddress2])


logcluster.pl --support=3400 \
--input=ls15.log --separator='["|\s]+' \
--lfilter='^(.*)(?:\|"\d+"){2}' --template='$1' \
--wfilter='blue\d\d' --wsearch='blue\d\d' \
--wreplace='blueNN' --wweight=0.5

(ws4-01.lab.blueNN.ex|ws4-04.lab.blueNN.ex
|ws4-03.int.blueNN.ex|ws4-04.int.blueNN.ex
|ws4-02.int.blueNN.ex|ws4-05.lab.blueNN.ex
|ws4-05.int.blueNN.ex|dlna.lab.blueNN.ex
|ws4-01.int.blueNN.ex|ws4-02.lab.blueNN.ex
|ws4-03.lab.blueNN.ex|git.lab.blueNN.ex)
(ssh|ssh.ipv6) OK SSH OK -
(OpenSSH_6.6.1p1|OpenSSH_5.9p1|OpenSSH_6.6.1_hpn1
3v11) (Ubuntu-2ubuntu2|FreeBSD-20140420|Debian-
5ubuntu1|Debian-5ubuntu1.4) (protocol 2.0)
```

Fig. 5. Sample joint clusters detected by LogCluster (for the reasons of privacy, sensitive data have been obfuscated).

V. Conclusion

In this paper, we have described the LogCluster algorithm for mining patterns from event logs. For future work, we plan to explore hierarchical event log clustering techniques. We also plan to implement the LogCluster algorithm in C, and use LogCluster for automated building of user behavior profiles.

Acknowledgment

The authors thank NATO CCD COE for making Locked Shields 2015 event logs available for this research. The authors also thank Mr. Kaido Raiend, Mr. Ants Leitmäe, Mr. Andrus Tamm, Dr. Paul Leis and Mr. Ain Rasva for their support.

REFERENCES

[1] Risto Vaarandi and Mauno Pihelgas, "Using Security Logs for Collecting and Reporting Technical Security Metrics," in *Proceedings of the 2014 IEEE Military Communications Conference*, pp. 294-299.

[2] Risto Vaarandi, "A Data Clustering Algorithm for Mining Patterns From Event Logs," in *Proceedings of the 2003 IEEE Workshop on IP Operations and Management*, pp. 119-126.

[3] Risto Vaarandi, "A Breadth-First Algorithm for Mining Frequent Patterns from Event Logs," in *Proceedings of the 2004 IFIP International Conference on Intelligence in Communication Systems*, LNCS Vol. 3283, Springer, pp. 293-308.

[4] Risto Vaarandi, "Mining Event Logs with SLCT and LogHound," in *Proceedings of the 2008 IEEE/IFIP Network Operations and Management Symposium*, pp. 1071-1074.

[5] Risto Vaarandi and Kārlis Podiņš, "Network IDS Alert Classification with Frequent Itemset Mining and Data Clustering," in *Proceedings of the 2010 International Conference on Network and Service Management*, pp. 451-456.

[6] Thomas Reidemeister, Mohammad A. Munawar and Paul A.S. Ward, "Identifying Symptoms of Recurrent Faults in Log Files of Distributed Information Systems," in *Proceedings of the 2010 IEEE/IFIP Network Operations and Management Symposium*, pp. 187-194.

[7] Thomas Reidemeister, Miao Jiang and Paul A.S. Ward, "Mining Unstructured Log Files for Recurrent Fault Diagnosis," in *Proceedings of the 2011 IEEE/IFIP International Symposium on Integrated Network Management*, pp. 377-384.

[8] Thomas Reidemeister, "Fault Diagnosis in Enterprise Software Systems Using Discrete Monitoring Data," PhD Thesis, University of Waterloo, 2012.

[9] Wei Xu, Ling Huang, Armando Fox, David Patterson and Michael Jordan, "Mining Console Logs for Large-Scale System Problem Detection," in *Proceedings of the 3rd Workshop on Tackling Computer Systems Problems with Machine Learning Techniques*, 2008.

[10] Adetokunbo Makanju, A. Nur Zincir-Heywood and Evangelos E. Milios, "Clustering Event Logs using Iterative Partitioning," in *Proceedings of the 2009 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1255-1264.

[11] Adetokunbo Makanju, "Exploring Event Log Analysis With Minimum Apriori Information," PhD Thesis, University of Dalhousie, 2012.

[12] Mika Klemettinen, "A Knowledge Discovery Methodology for Telecommunication Network Alarm Databases," PhD thesis, University of Helsinki, 1999.

[13] Qingguo Zheng, Ke Xu, Weifeng Lv and Shilong Ma, "Intelligent Search of Correlated Alarms from Database Containing Noise Data," in *Proceedings of the 2002 IEEE/IFIP Network Operations and Management Symposium*, pp. 405-419.

[14] Sheng Ma and Joseph L. Hellerstein, "Mining Partially Periodic Event Patterns with Unknown Periods," in *Proceedings of the 17th International Conference on Data Engineering*, pp. 205-214, 2001.

[15] James J. Treinen and Ramakrishna Thurimella, "A Framework for the Application of Association Rule Mining in Large Intrusion Detection Infrastructures," in *Proceedings of the 2006 Symposium on Recent Advances in Intrusion Detection*, LNCS Vol. 4219, Springer, pp. 1-18.

[16] Chris Clifton and Gary Gengo, "Developing Custom Intrusion Detection Filters Using Data Mining," in *Proceedings of the 2000 IEEE Military Communications Conference*, pp. 440-443.

[17] Jon Stearley, "Towards Informatic Analysis of Syslogs," in *Proceedings of the 2004 IEEE International Conference on Cluster Computing*, pp. 309–318.

[18] Adetokunbo Makanju, Stephen Brooks, A. Nur Zincir-Heywood and Evangelos E. Milios, "LogView: Visualizing Event Log Clusters," in *Proceedings of the 6th Annual Conference on Privacy, Security and Trust*, pp. 99-108, 2008.

[19] Daniela Brauckhoff, Xenofontas Dimitropoulos, Arno Wagner and Kavè Salamatian, "Anomaly Extraction in Backbone Networks using Association Rules," in *Proceedings of the 2009 ACM SIGCOMM Internet Measurement Conference*, pp. 28-34.

[20] Eduard Glatz, Stelios Mavromatidis, Bernhard Ager and Xenofontas Dimitropoulos, "Visualizing big network traffic data using frequent pattern mining and hypergraphs," Computing Vol. 96(1), Springer, pp. 27-38, 2014.

Appendix 3

Publication III

R. Vaarandi, M. Kont, and M. Pihelgas. Event log analysis with the LogCluster tool. In *MILCOM 2016 - 2016 IEEE Military Communications Conference*, pages 982–987, November 2016

The paper is included in the *Proceedings of the 2016 IEEE Military Communications Conference (MILCOM 2016)*.
DOI: 10.1109/MILCOM.2016.7795458

Event Log Analysis with the LogCluster Tool

Risto Vaarandi
TUT Centre for Digital Forensics and Cyber Security
Tallinn University of Technology
Tallinn, Estonia
firstname.lastname@ttu.ee

Markus Kont and Mauno Pihelgas
Technology Branch
NATO CCD COE
Tallinn, Estonia
firstname.lastname@ccdcoe.org

***Abstract*—Today, event logging is a widely accepted concept with a number of event formatting standards and event collection protocols. Event logs contain valuable information not only about system faults and performance issues, but also about security incidents. Unfortunately, since modern data centers and computer networks are known to produce large volumes of log data, the manual review of collected data is beyond human capabilities. For automating this task, a number of data mining algorithms and tools have been suggested in recent research papers. In this paper, we will describe the application of the LogCluster tool for mining event patterns and anomalous events from security and system logs.**

Keywords—security log analysis; event log clustering; pattern mining from event logs; data mining

I. Introduction

Nowadays, event logging is supported by most applications, services, network devices, and other IT system components. Well-known standards exist for event logging (such as BSD syslog [1] and IETF syslog [2]) and widely used solutions have been developed for event log collection (such as rsyslog [3], syslog-ng [4], and Elastic Stack [5]). Event logs contain valuable information about security incidents, but since large volumes of log data are generated in modern data centers and computer networks [6], the manual review of event logs is infeasible. In order to aid the human analyst, a number of data mining algorithms and tools have been proposed [7–22]. Many suggested approaches are semi-automated, allowing for interactive discovery of event patterns from event logs. This knowledge can be used for various purposes like handling security incidents and developing event correlation rules [23]. During the last decade, data clustering algorithms have been often suggested for mining line patterns from textual event logs. Proposed algorithms assume that each line in the event log is a complete representation of some event. The algorithms divide the lines into clusters, so that lines from the same cluster are similar and matching the same line pattern. Instead of printing lines in each cluster, the algorithms output a line pattern for each cluster to the end user. Also, lines that do not fit into any of the detected clusters are arranged into a special cluster of outliers and reported individually. Due to their nature, clustering algorithms are able to identify not only event patterns that reflect regularities, but also unusual outlier events that deserve closer attention from security personnel.

In this paper, we describe the LogCluster tool for mining textual event logs and present example scenarios of detecting security incidents and anomalous events. Full details of the clustering algorithm implemented by the tool have been given in our recent paper [7]. The remainder of this paper is organized as follows – section II reviews related work, section III describes the LogCluster tool and focuses on its newly developed functionality along with several use cases, while section IV concludes the paper and provides the download and licensing information for the LogCluster tool.

This work has been supported by Estonian IT Academy (StudyITin.ee) and SEB Estonia.

II. Related Work

One of the earliest event log clustering algorithms is SLCT [8] which has been applied in various domains like IDS alarm log processing [9, 10], detection of recurrent fault conditions [11, 12], and visualization of event log data [19, 20]. SLCT takes *support threshold s* as a user-given input parameter, and starts the clustering process by identifying *frequent words* that appear in *s* or more event log lines. The words are considered with positional information, e.g., if the fifth word of the event log line is *kernel*, it is treated as a tuple *(kernel, 5)*. After identifying frequent words, another pass is made over input data for assigning lines to cluster candidates. For each line, all frequent words are extracted, and the candidate for this line is identified by the set of extracted words. After the data pass, *frequent candidates* that contain *s* or more lines are selected as clusters. The number of lines in a cluster (or a candidate) is called the *support* of the cluster (or the candidate). For example, consider the event log with four lines:

User bob login from 10.0.0.1

User alice login from 10.0.0.1

User jim login from 10.0.0.2

User Srv Admin login from 10.0.0.3

If *s=3*, the words *(User, 1)*, *(login, 3)*, and *(from, 4)* are detected as frequent. Also, two candidates are identified – the candidate *{(User, 1), (login, 3), (from, 4)}* with support 3 that contains first three lines, and the candidate *{(User, 1)}* with support 1 that contains the last line. The first candidate is selected as a cluster and is reported as the line pattern *User * login from* (since the cluster has no word associated with position 2, a wildcard is printed for this position). Finally, the last line is reported as an outlier.

Unfortunately, SLCT is known to suffer from some shortcomings [9, 12, 13]. Firstly, it does not detect wildcard suffixes for line patterns as illustrated by the previous example.

Secondly, SLCT is sensitive to word delimiter noise and shifts in word positions. For instance, in the above example the last event log line is not assigned to the cluster represented by the pattern *User * login from*. Finally, when mining is conducted with lower support thresholds, SLCT is prone to overfitting – clusters with meaningful line patterns could be needlessly split, so that resulting clusters have too specific line patterns. For example, if *s=2* for the above event log example, only the pattern *User * login from 10.0.0.1* is detected which does not represent the general case.

Recently, we have developed a clustering algorithm called LogCluster that addresses the shortcomings of SLCT [7]. Similarly to SLCT, the user must supply the support threshold *s* to LogCluster which is used for finding frequent words during the first pass over the event log. However, positional information is not encoded into words. In order to identify a cluster candidate for each event log line during the second data pass, LogCluster extracts all frequent words from the line and arranges them into a tuple. Also, summary information about infrequent words in all assigned lines is maintained with each candidate. Candidates containing *s* or more lines are selected as clusters and reported as line patterns, while lines without a cluster are regarded outliers and reported during an optional data pass. For instance, if *s=3* for the example event log above, all lines are assigned to the cluster identified by the tuple *(User, login, from)*, and the line pattern *User *{1,2} login from *{1,1}* is reported for this cluster together with its support of 4.

Reidemeister et al developed a methodology for diagnosing recurrent faults in software systems which employs a modified version of SLCT for software logs [11, 12]. In order to handle delimiter noise and shifts in word positions, results from SLCT are clustered further with a single-linkage clustering algorithm that uses a Levenshtein distance function. Detected knowledge is then harnessed for building decision tree classifiers.

Makanju developed a divisive clustering algorithm IPLoM that starts with the event log as a single cluster and splits it into partitions during three steps [13]. Splitting is based on various criteria, such as the number of words in event log lines and associations between word pairs. After splitting, a line pattern is derived for each partition. Unlike SLCT, IPLoM is able to identify wildcard suffixes for line patterns.

Apart from clustering algorithms, frequent itemset mining methods have been often employed for event log mining. LogHound is a generalization of the Apriori algorithm that can discover line patterns from textual event logs [9]. Other frequent itemset mining approaches have been mainly used for the detection of temporal associations between event types [14–18] and for mining NetFlow traffic patterns [21, 22].

III. The LogCluster Tool

The LogCluster tool is an open-source Perl-based UNIX command line utility. It is able to mine meaningful patterns from large event logs, and our recent study provides detailed performance data for the 0.01 version [7]. In this section, we will review the features of the latest version and discuss several use cases. Event logs presented in this section originate from several large and mid-size private and military organizations, with all sensitive data being obfuscated in Fig. 1–4.

A. Introduction and Basic Use

All parameters are supplied to the LogCluster tool with command line options. For example, the following command line

logcluster.pl --support=100 --input=/var/log/messages

mines line patterns from */var/log/messages* with support threshold 100. Default word delimiter is whitespace, but custom delimiter can be defined with the *--separator* command line option. In order to mine patterns from several log files, multiple *--input* options can be provided and wildcards can be used in file names (e.g., *--input=/var/log/*.log*). The above command line runs the basic variant of the LogCluster algorithm which involves two passes over */var/log/messages* for finding frequent words and cluster candidates respectively.

```
logcluster.pl --input=suricata.log --support=1000 \
              --wsize=10000 --csize=10000

Feb 27 *{1,1} myhost suricata[17447]: [1:2012708:2]
ET WEB_SERVER HTTP 414 Request URI Too Large
[Classification: Web Application Attack]
[Priority: 1] {TCP} 10.0.19.12:80 -> *{1,1}
Support: 44744

Feb 5 *{1,1} myhost suricata[2223]: [1:2006445:13]
ET WEB_SERVER Possible SQL Injection Attempt SELECT FROM
[Classification: Web Application Attack]
[Priority: 1] {TCP} *{1,1} -> 10.0.33.7:80
Support: 39692

Oct 18 *{1,1} myhost suricata[18941]: [1:2006446:11]
ET WEB_SERVER Possible SQL Injection Attempt UNION SELECT
[Classification: Web Application Attack]
[Priority: 1] {TCP} *{1,1} -> 10.0.3.5:80
Support: 7493

Mar 15 *{1,1} myhost suricata[25554]: [1:2016936:2]
ET WEB_SERVER SQL Injection Local File Access Attempt
Using LOAD_FILE [Classification: Web Application Attack]
[Priority: 1] {TCP} *{1,1} -> 10.0.6.1:80
Support: 3293

Jan 2 *{1,1} myhost suricata[30119]: [1:2101201:10]
GPL WEB_SERVER 403 Forbidden [Classification: Attempted
Information Leak] [Priority: 2] {TCP} 10.0.3.9:80 -> *{1,1}
Support: 2826
…
```

Fig. 1. Sample Suricata IDS alarm patterns.

When mining larger log files, the number of distinct words can be quite large, and with lower support thresholds many cluster candidates could be generated. Therefore, it is expensive to keep all words and cluster candidates in memory when their occurrence counts are established. In order to reduce the memory consumption by filtering out infrequent words, a sketch based technique can be employed which requires an extra data pass. During the data pass, the word sketch of *m* counters is created, where each counter reflects the occurrence counts of many words and acts as a filter (see [7] for full details). A similar method can be used for filtering out infrequent candidates. The number of counters in the word sketch is set with the *--wsize* command line option, while the size of the candidate sketch can be set with the *--csize* option.

Fig. 1 illustrates example line patterns detected by the LogCluster tool from the Suricata IDS log file. The log file covered the period of 6 months and contained 949,920 lines.

Since the support threshold was set to 1000, strong alarm patterns were identified which reflect the days when intensive attacks against specific hosts were conducted. During the mining process, the word and candidate sketches of 10,000 counters were employed. Both sketches involved an additional data pass, and the memory consumption of the LogCluster tool was reduced from 406.2 MB to 13.4 MB.

B. Preprocessing Input

While Fig. 1 provides an example of discovering relevant patterns from a raw log file, quite often the detection of meaningful patterns requires elaborate preprocessing of event logs (e.g., dropping irrelevant events, removal of timestamps, and rewriting specific parts of event log lines). In many cases, such tasks require dedicated scripts that store preprocessed log data to disk. For avoiding this overhead, the LogCluster tool provides native support for flexible input preprocessing. If a regular expression is supplied with the *--lfilter* option, only the lines matching this regular expression are clustered. Also, if the regular expression sets match variables, the variables can be used in the format string defined with the *--template* option, in order to convert matching event log lines in memory before further processing. Fig. 2 depicts a LogCluster application example for SSH daemon syslog events where timestamp, hostname, and program name were discarded during clustering (for instance, the event log line *Mar 27 12:01:33 server113 sshd[15437]: test message* was converted to *test message*). The *authpriv.log* file contained over 7 million lines from hundreds of UNIX servers, while 98,920 lines matched the *--lfilter* option and were converted. The mining was conducted with the relative support threshold of 1% (that means setting support threshold to 1% of the number of clustered lines, i.e., 989). Also, 2200 outlier event log lines were detected and written to the *outliers.log* file. Most outliers reflected SSH probing of non-existing user accounts by the organizational vulnerability scanning engine, but some outliers were also error messages that manifested system faults and configuration errors.

In some cases, regular expression based filtering and conversion might not be sufficient for complex preprocessing tasks. For addressing this issue, the LogCluster tool also supports the *--lcfunc* option which takes a definition of an anonymous Perl function for its value. The function is compiled when LogCluster starts, and the compiled code is invoked for filtering and converting each event log line. An event log line is passed to the function as its only input parameter, and in order to indicate the line should not be considered during clustering, Perl *undef* value must be returned from the function. If any other value is returned, it replaces the original event log line. For example, if the option

--lcfunc='sub { if ($_[0] =~ s/192\.168\.\d{1,3}\.\d{1,3}/ip-192.168/g) { return $_[0]; } return undef; }'

is employed, LogCluster only considers lines which contain IP addresses from the *192.168.0.0/16* network, and each such address is replaced with the string *ip-192.168*. Finally, since providing longer Perl functions in command line is not convenient, input preprocessing routines can be defined in a separate Perl module and used through the *--lcfunc* interface. For example, if the option

--lcfunc='require "/opt/logcluster/perlmod/Test.pm"; sub { Test::lineconvert($_[0]); }'

is provided, the function *lineconvert()* from the module */opt/logcluster/perlmod/Test.pm* is invoked for filtering and converting each event log line.

```
logcluster.pl --input=authpriv.log --rsupport=1 --aggrsup \
--lfilter='sshd\[\d+\]: (?<msgtext>.+)' --template='${msgtext}' \
--outliers=outliers.log

pam_unix(sshd:session): session opened for user *{1,1} by (uid=0)
Support: 26708

Accepted publickey for *{1,1} from *{1,1} port *{1,1} ssh2
Support: 24160

pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0
tty=ssh ruser= *{1,2}
Support: 1362
…

# examples of outlier events from outliers.log

Mar 18 04:43:43 server112 sshd[22902]: Failed password for
  invalid user emailswitch from 10.31.97.21 port 46668 ssh2
Mar 18 04:43:50 server112 sshd[22936]: Failed password for
  invalid user admin from 10.31.97.21 port 46686 ssh2
Mar 18 04:44:06 server112 sshd[23000]: Failed password for
  invalid user manage from 10.31.97.21 port 46726 ssh2
Mar 18 04:44:53 server112 sshd[23056]: Failed password for
  invalid user cisco from 10.31.97.21 port 46841 ssh2
Mar 18 06:31:38 server29 sshd[12133]: PAM unable to dlopen
  (/lib64/security/pam_oddjob_mkhomedir.so):
  /lib64/security/pam_oddjob_mkhomedir.so: cannot open shared
  object file: No such file or directory
```

Fig. 2. Sample SSH daemon event patterns and outlier events.

C. Defining Word Classes

In many cases, infrequent words share the same format that is not detected during clustering. For example, the program name of syslog messages is often followed by frequently changing process ID, creating many infrequent words for the same program (e.g., *sshd[18991]:* and *sshd[7655]:*). For discovering such regularities, LogCluster supports the creation of word classes according to user-defined rules, where each word class represents many infrequent words and captures their commonalities. If a regular expression is given with the *--wfilter* option, word classes are set up for all words that match this expression. Word class creation involves searching the word for all substrings that match the regular expression supplied with the *--wsearch* option, and replacing these substrings with the string provided with the *--wreplace* option. For example, with the following command line options

--wfilter='^\w+\[\d+\]:$' --wsearch='\[\d+\]' --wreplace='[PID]'

the word class *sshd[PID]:* is created for words *sshd[1321]:* and *sshd[9583]:*, while the word class *suricata[PID]:* is set up for words *suricata[2133]:* and *suricata[17743]:*. Word classes are treated like regular words by LogCluster. If a word class is frequent, it replaces all corresponding infrequent words during the clustering process.

The LogCluster tool also features a more powerful *--wcfunc* option which allows for the creation of word classes with a user-defined Perl function. Unlike with regular expression

based options, multiple word classes can be created for a word, and the classes are returned from the function as a Perl list. The order of word classes in the list defines their priority – if the word is infrequent, the first frequent word class from the list replaces the word. For example, if the

--wcfunc='sub { if ($_[0] =~ /^Chrome\/(\d+)/) { return ("Chrome/$1", "Chrome"); } }'

option is provided, word classes *Chrome/49* and *Chrome* are created for the word *Chrome/49.0.2623.87*, with *Chrome/49* having precedence over *Chrome*. If the word class *Chrome/49* and the word *Chrome/49.0.2623.87* are infrequent, then word class *Chrome* replaces *Chrome/49.0.2623.87* during the clustering process. As with the *--lcfunc* option described in the previous subsection, more complex word class creation routines can be separated into Perl modules (the LogCluster distribution includes one such example module).

D. Joining Clusters

As discussed in section II, clusters can be split needlessly with lower support thresholds. As a result, instead of generic meaningful line patterns too specific patterns are detected. For addressing the overfitting problem, LogCluster supports two heuristics for joining clusters that make resulting line patterns more comprehensible to the human analyst. The first heuristic is enabled with the *--aggrsup* option and is applied to cluster candidates before clusters are selected. The heuristic allows cluster overlaps, in order to increase the support of clusters with more generic patterns – for each cluster candidate, other candidates with more specific line patterns are identified, and their lines are also assigned to the given candidate. For instance, if there are two candidates with line patterns *Interface *{1,1} down* and *Interface eth0 down*, lines of the second candidate are also assigned to the first candidate, and the support of the first candidate is incremented by the support of the second candidate. The use of the *--aggrsup* option has been illustrated in Fig. 2.

The second heuristic is applied after clusters have been selected from candidates. The heuristic relies on word weight functions that are described below. Suppose that *a* and *b* are frequent words, *m* denotes the number of event log lines where *a* appears, and *n* denotes the number of event log lines where both *a* and *b* appear. Provided that $n > 0$ (i.e., there is at least one event log line where both *a* and *b* are present), dependency from *a* to *b* is defined as:

$$dep(a, b) = n / m$$

Note that $dep(a, a) = 1$ and $0 < dep(a, b) \leq 1$, with higher value of $dep(a, b)$ indicating higher likelihood of observing *b* when *a* is present. For measuring how strongly each word in the line pattern is correlated to other words in this pattern, LogCluster defines several word weight functions which return values from 0 to 1. If $w_1,\ldots,w_k$ are words in the line pattern, the word weight function f_1 is defined as:

$$f_1(w_i) = \sum_{j=1}^{k} dep(w_j, w_i) / k$$

The smaller the value of $f_1(w_i)$, the less likely it is to observe w_i with other words of the pattern. For identifying words with insufficient weights in line patterns, the *word weight threshold* is defined with the *--wweight* option. For example, if *--wweight=0.5* while $dep(Interface, eth0) = 0.07$ and $dep(down, eth0) = 0.07$, then the word *eth0* has insufficient weight in the pattern *Interface eth0 down*, since $f_1(eth0) = (0.07 + 0.07 + 1) / 3 = 0.38$. The cluster joining heuristic replaces such words with special tokens in cluster identifier tuples, and joins two clusters if their modified tuples are identical. Then a new line pattern is derived for the joint cluster by creating lists from words with insufficient weights and joining wildcards (see [7] for full details). For instance, suppose there are two clusters with identifier tuples *(Interface, eth0, down)* and *(Interface, eth1, down)*, and line patterns *Interface eth0 down* and *Interface eth1 down*. If words *eth0* and *eth1* have insufficient weights in their respective patterns, modified identifier tuples for both clusters are *(Interface, TOKEN, down)*, and two clusters are thus joined into a new cluster with the line pattern *Interface (eth0|eth1) down*. Since overfitting introduces words with lower weights into patterns, the cluster joining heuristic helps to reduce the number of such patterns by joining them into more meaningful patterns which are built around strongly associated words.

The f_1 word weight function has some drawbacks that have motivated the development of additional functions in recent versions of LogCluster. Firstly, if *k* is the number of words in the pattern, then $1/k \leq f_1(w_i) \leq 1$, since $dep(w_i, w_i) = 1$ (i.e., f_1 allows the word to contribute $1/k$ to its own weight). Therefore, if a pattern has few words, its words will be assigned higher weights than words of longer patterns. This bias becomes more noticeable if the same word appears several times in the pattern, for example, *interface *{1,1} down: interface *{1,2} fault*. The word weight function f_2 addresses this shortcoming by first identifying the set of unique words *U* for the pattern. For instance, for the previous example pattern $U = \{interface, down:, fault\}$. If *U* contains *p* words (i.e., $p = |U|$), f_2 is defined as follows:

$$f_2(w) = ((\sum_{v \in U} dep(v, w)) - 1) / (p - 1), \quad \text{if } p > 1$$

$$f_2(w) = 1, \quad \text{if } p = 1$$

For example, if $dep(Interface, eth0) = 0.07$ and $dep(down, eth0) = 0.07$, then $f_1(eth0) = 0.38$ for line pattern *Interface eth0 down*, while $f_2(eth0) = (0.07 + 0.07 + 1 - 1) / 2 = 0.07$. By not allowing the word to contribute to its own weight, f_2 calculates word weights in a fair way for shorter patterns.

One shortcoming of f_1 and f_2 weight functions is their inability to assign higher weights to groups of strongly associated words. For instance, suppose the line pattern is *kernel: interface *{1,1} down*, and the words *interface* and *down* always appear together (i.e., $dep(interface, down) = dep(down, interface) = 1$). Also, suppose the word *kernel:* is always present when words *interface* and *down* appear (i.e., $dep(interface, kernel:) = dep(down, kernel:) = 1$), while only 4% of event log lines that contain *kernel:* also contain the words *interface* and *down* (i.e., $dep(kernel:, interface) = dep(kernel:, down) = 0.04$). Thus, $f_1(kernel:) = 1$ and $f_1(interface) = f_1(down) = 0.68$, although word weights should be distributed more evenly in this pattern, considering that words *interface* and *down* form a very strong sub-pattern. For achieving this purpose, the mutual dependency between frequent words *a* and *b* is defined as:

$mdep(a, b) = (dep(a, b) + dep(b, a)) / 2$

Note that $mdep(a, b) = mdep(b, a)$ and $0 < mdep(a, b) \leq 1$. Also, high values of $mdep(a, b)$ indicate that words a and b usually appear together, while low values reflect the lack of such strong association. If $w_1,\ldots,w_k$ are words in the line pattern, the word weight function f_3 is defined as:

$f_3(w_i) = \sum_{j=1}^{k} mdep(w_j, w_i) / k$

Like the f_2 function, the word weight function f_4 employs the set of unique words U for the pattern, and is defined as follows (note that $p = |U|$):

$f_4(w) = ((\sum_{v \in U} mdep(v, w)) - 1) / (p - 1)$, if $p > 1$

$f_4(w) = 1$, if $p = 1$

For instance, in the case of the previous line pattern example $mdep(interface, down) = (1 + 1) / 2 = 1$, $mdep(kernel:, interface) = (0.04 + 1) / 2 = 0.52$, and $mdep(kernel:, down) = (0.04 + 1) / 2 = 0.52$. Therefore, $f_3(kernel:) = (1 + 0.52 + 0.52) / 3 = 0.68$, while $f_3(interface) = (0.52 + 1 + 1) / 3 = 0.84$ and $f_3(down) = (0.52 + 1 + 1) / 3 = 0.84$. In other words, compared to f_1, the f_3 function assigns more weight to words *interface* and *down* due to their strong association.

The LogCluster tool allows for choosing the weight function with the *--weightf* option, e.g., *--weightf=3* selects the f_3 function. Also, the *--color* option highlights words with insufficient weights in reported line patterns. Finally, detected cluster and word dependency information can be stored to the dump file given with the *--writedump* option. The data from previously created dump file can be loaded with the *--readdump* option during further runs of LogCluster. This is useful for quick evaluation of different word weight thresholds and functions without repeating the full clustering process that is computationally expensive. For example, the following command line

logcluster.pl --readdump=cluster.dump --wweight=0.75

loads the cluster and word dependency data from dump file *cluster.dump*, and joins clusters with the word weight threshold 0.75 and word weight function f_1 (the default function).

E. Case Studies

This subsection presents some log analysis scenarios that utilize previously described features of the LogCluster tool. Fig. 3 provides an example of employing LogCluster in a mid-size private organization, in order to create daily e-mail reports from syslog events of critical servers. Nearly 12 million events are collected each day with 150-300 detected line patterns. For producing more meaningful patterns, word classes are created for words which contain punctuation marks by replacing letters, digits, and other non-punctuation characters with character *X*. Replacement is not done if non-punctuation characters are followed by *[* or = character, in order to preserve keywords and program names which precede these characters (e.g., word class *to=<X@X.X>* is created for the word *to=<user@example.com>*). Also, the word weight threshold 0.5 and word weight function f_2 are used for joining clusters. The results of the clustering are written into the */tmp/logcluster-rotate.dmp* dump file, in order to facilitate quick additional analysis with different word weight thresholds and functions. Fig. 3 also depicts some example patterns detected with the LogCluster tool. The first pattern in Fig. 3 manifests an attempt to use the organizational DNS server for conducting DNS reflection and amplification attacks. The second pattern represents SSH account probing from several Internet hosts against a number of servers and routers of the organization. Remaining patterns reflect various attempts to distribute spam through the organizational mail server.

```
logcluster.pl --input=/var/log/all.log --rsupport=0.01 \
--wfilter='[[:punct:]]' --wsearch='[^[:punct:]]++(?![[=])' \
--wreplace=X --writedump=/tmp/logcluster-rotate.dmp \
--wweight=0.5 --weightf=2 --csize=100000 --wsize=100000

Mar 29 X:X:X nameserver2 named[10307]: security: info: client
(X.X.X.X#X:|10.0.137.69#25345:) view authoritative: query (cache)
('X.X.X.X.X.X/X/X'|'X.X.X.X/X/X'|'X.X.X.X.X/X/X'|'domain.nu/MX/IN'
|'isc.org/ANY/IN') denied
Support: 198152

Mar 29 X:X:X (backupserver|vps1|nameserver1|router1|vps2|router2
|mailserver|logserver|vps3|vps4) sshd[X]: pam_unix(sshd:auth):
authentication failure; logname= uid=0 euid=0 tty=ssh ruser=
(rhost=10.3.202.120|rhost=10.88.177.98) user=root
Support: 18112

Mar 26 X:X:X mailserver X/smtpd[X]: NOQUEUE: reject: RCPT from
exch001.example.com[10.52.134.35]: 454 4.7.1 <user@example.com>:
Relay access denied; from=<> to=<user@example.com> proto=ESMTP
helo=<webmail.example.com>
Support: 941

Mar 17 X:X:X mailserver X/smtpd[X]: warning: hostname
host94165.example.com does not resolve to address X.X.X.X
Support: 1217

Mar 9 X:X:X mailserver X/smtpd[X]: NOQUEUE: reject: RCPT from
unknown[X.X.X.X]: 554 5.7.1 Service unavailable; Client host
[X.X.X.X] blocked using cbl.abuseat.org;
Blocked - see X://X.X.X/X.X?ip=X.X.X.X; from=<X@X.X> to=<X@X.X>
proto=ESMTP *{1,1}
Support: 1219
…
```

Fig. 3. Sample attack patterns detected from syslog events.

In some cases, it might not be convenient to cluster the event log with one LogCluster run, since higher support threshold might yield too many outliers, while with lower support threshold a large number of clusters might be produced. This problem often appears for event logs which contain events from many servers and programs, and feature meaningful line patterns with a wide variety of supports. For addressing this problem, LogCluster can be used iteratively, clustering results from previous execution(s) at each step. Fig. 4 provides an example of iterative clustering of the *mail.log* file which contained syslog messages with *mail* facility from a number of mail servers. During the first iteration with relative support threshold 0.1%, each event log line was converted to a program name string, so that detected line patterns indicated programs that have produced most log messages in *mail.log*. A cluster for the *sendmail* daemon was discovered, and during the second iteration with relative support threshold 0.1% it was split into smaller clusters by analyzing the message text after the program name. The second iteration yielded 268 patterns that reflected normal system activity and 10,264 outliers. The outliers were clustered further with support threshold 50. As a result, 105 patterns and 2018 outliers were detected, with many

patterns and outliers representing error conditions and abnormal events such as connection attempts from spammers.

```
logcluster.pl --input=mail.log --rsupport=0.1 \
--lfilter=' ([\w\/.-]+)\[\d+\]: ' --template='$1[PID]:'

sendmail[PID]:
Support: 1007754
…

logcluster.pl --input=mail.log –rsupport=0.1 --aggrsup \
--lfilter='sendmail\[\d+\]: (.+)' --template='$1' \
--separator='(?:\s+|=)' --outliers=outliers.log

*{1,1} from *{1,5} size *{1,1} class 0, nrcpts 1, msgid *{1,5}
proto ESMTP, daemon MTA, relay *{1,5}
Support: 161976

STARTTLS client, relay *{1,1} version TLSv1/SSLv3, verify OK,
cipher AES128-SHA, bits 128/128
Support: 71062
…

logcluster.pl --input=outliers.log --support=50 --aggrsup \
--lfilter 'sendmail\[\d+\]: (.+)' --template '$1' \
--separator='(?:\s+|=)' --outliers=outliers2.log

*{1,1} ruleset check_rcpt, arg1 *{1,1} relay *{1,2} reject 550
5.1.1 *{1,1} User unknown
Support: 441

*{1,1} SYSERR(root): collect: I/O error on connection from *{1,1}
from *{1,2}
Support: 104
…

# examples of outlier events from outliers2.log

Mar 29 03:51:28 mailserver sendmail[30101]: ruleset=check_relay,
  arg1=box.example.com, arg2=127.0.0.1, reject=550 5.7.1
  Rejected: 10.193.172.92 listed at xbl.spamhaus.org
Mar 29 08:28:07 mailserver sendmail[6276]: XXX: mail.example.com
  [10.109.254.117]: Possible SMTP RCPT flood, throttling.
Mar 29 10:23:58 mailserver sendmail[22746]: XXX:
  ruleset=check_mail, arg1=<pzfsibdlkj@sjfqc.biz>,
  relay=[10.240.79.7], reject=553 5.1.8 <pzfsibdlkj@sjfqc.biz>...
  Domain of sender address pzfsibdlkj@sjfqc.biz does not exist
```

Fig. 4. Iterative analysis of mail server events.

IV. Conclusion

In this paper, we have presented the LogCluster tool for mining line patterns and outlier events from textual event logs. We have also described several scenarios of discovering security incidents and anomalous events with this tool. For a more detailed information on its performance and comparison with other log clustering algorithms, the reader is referred to our recent paper [7].

For the future work, we plan to harness the LogCluster tool for insider threat detection and to modify the LogCluster algorithm for stream mining purposes. The LogCluster tool has been released under the terms of GNU GPLv2 and is available from *http://ristov.github.io/logcluster*.

Acknowledgment

The authors thank Mr. Kaido Raiend, Mr. Ain Rasva, Mr. Raimo Peterson, Mrs. Katrin Kriiska, Prof. Olaf M. Maennel, and Dr. Rain Ottis for supporting this work.

References

[1] C. Lonvick, “The BSD syslog Protocol,” RFC3164, 2001.

[2] R. Gerhards, “The Syslog Protocol,” RFC5424, 2009.

[3] http://www.rsyslog.com

[4] https://www.balabit.com/network-security/syslog-ng

[5] https://www.elastic.co

[6] Risto Vaarandi and Mauno Pihelgas, “Using Security Logs for Collecting and Reporting Technical Security Metrics,” in Proceedings of the 2014 IEEE Military Communications Conference, pp. 294-299.

[7] Risto Vaarandi and Mauno Pihelgas, “LogCluster – A Data Clustering and Pattern Mining Algorithm for Event Logs,” in Proceedings of the 2015 International Conference on Network and Service Management, pp. 1-7.

[8] Risto Vaarandi, “A Data Clustering Algorithm for Mining Patterns From Event Logs,” in Proceedings of the 2003 IEEE Workshop on IP Operations and Management, pp. 119-126.

[9] Risto Vaarandi, “Mining Event Logs with SLCT and LogHound,” in Proceedings of the 2008 IEEE/IFIP Network Operations and Management Symposium, pp. 1071-1074.

[10] Risto Vaarandi and Kārlis Podiņš, “Network IDS Alert Classification with Frequent Itemset Mining and Data Clustering,” in Proceedings of the 2010 International Conference on Network and Service Management, pp. 451-456.

[11] Thomas Reidemeister, Miao Jiang and Paul A.S. Ward, “Mining Unstructured Log Files for Recurrent Fault Diagnosis,” in Proceedings of the 2011 IEEE/IFIP International Symposium on Integrated Network Management, pp. 377-384.

[12] Thomas Reidemeister, “Fault Diagnosis in Enterprise Software Systems Using Discrete Monitoring Data,” PhD Thesis, University of Waterloo, 2012.

[13] Adetokunbo Makanju, “Exploring Event Log Analysis With Minimum Apriori Information,” PhD Thesis, University of Dalhousie, 2012.

[14] Mika Klemettinen, “A Knowledge Discovery Methodology for Telecommunication Network Alarm Databases,” PhD thesis, University of Helsinki, 1999.

[15] Qingguo Zheng, Ke Xu, Weifeng Lv and Shilong Ma, “Intelligent Search of Correlated Alarms from Database Containing Noise Data,” in Proceedings of the 2002 IEEE/IFIP Network Operations and Management Symposium, pp. 405-419.

[16] Sheng Ma and Joseph L. Hellerstein, “Mining Partially Periodic Event Patterns with Unknown Periods,” in Proceedings of the 17th International Conference on Data Engineering, pp. 205-214, 2001.

[17] James J. Treinen and Ramakrishna Thurimella, “A Framework for the Application of Association Rule Mining in Large Intrusion Detection Infrastructures,” in Proceedings of the 2006 Symposium on Recent Advances in Intrusion Detection, LNCS Vol. 4219, Springer, pp. 1-18.

[18] Chris Clifton and Gary Gengo, “Developing Custom Intrusion Detection Filters Using Data Mining,” in Proceedings of the 2000 IEEE Military Communications Conference, pp. 440-443.

[19] Jon Stearley, “Towards Informatic Analysis of Syslogs,” in Proceedings of the 2004 IEEE International Conference on Cluster Computing, pp. 309–318.

[20] Adetokunbo Makanju, Stephen Brooks, A. Nur Zincir-Heywood and Evangelos E. Milios, “LogView: Visualizing Event Log Clusters,” in Proceedings of the 6th Annual Conference on Privacy, Security and Trust, pp. 99-108, 2008.

[21] Daniela Brauckhoff, Xenofontas Dimitropoulos, Arno Wagner and Kavè Salamatian, “Anomaly Extraction in Backbone Networks using Association Rules,” in Proceedings of the 2009 ACM SIGCOMM Internet Measurement Conference, pp. 28-34.

[22] Eduard Glatz, Stelios Mavromatidis, Bernhard Ager and Xenofontas Dimitropoulos, “Visualizing big network traffic data using frequent pattern mining and hypergraphs,” Computing Vol. 96(1), Springer, pp. 27-38, 2014.

[23] Risto Vaarandi, “Simple Event Correlator for real-time security log monitoring,” Hakin9 Magazine 1/2006 (6), pp. 28-39, 2006.

Appendix 4

Publication IV
B. Blumbergs, M. Pihelgas, M. Kont, O. Maennel, and R. Vaarandi. Creating and Detecting IPv6 Transition Mechanism-Based Information Exfiltration Covert Channels. In *Secure IT Systems: 21st Nordic Conference, NordSec 2016, Oulu, Finland, November 2-4, 2016. Proceedings*, pages 85–100. Springer International Publishing, 2016

The paper is included in the *Proceedings of the 2016 Secure IT Systems: 21st Nordic Conference (NordSec 2016)*.
DOI: 978-3-319-47560-8_6

Creating and Detecting IPv6 Transition Mechanism-Based Information Exfiltration Covert Channels

Bernhards Blumbergs[1], Mauno Pihelgas[1], Markus Kont[1], Olaf M. Maennel[2], and Risto Vaarandi[2]

[1] NATO Cooperative Cyber Defense Center of Excellence, Tallinn, Estonia
name.surname[a]ccdcoe.org
[2] Tallinn University of Technology, Tallinn, Estonia
name.surname[a]ttu.ee

Abstract. The Internet Protocol Version 6 (IPv6) transition opens a wide scope for potential attack vectors. Transition mechanisms could allow the set-up of covert egress communication channels over an IPv4-only or dual-stack network. Increased usage of IPv6 in attacks results in full compromise of a target network. Therefore effective tools are required for the execution of security operations for assessment of possible attack vectors related to IPv6 security.
In this paper, we review relevant transition technologies, describe and analyze two newly-developed IPv6 transition mechanism-based proof-of-concept tools for the establishment of covert information exfiltration channels. The analysis of the generated test cases confirms that IPv6 and various evasion techniques pose a difficult task for network security monitoring. While detection of various transition mechanisms is relatively straightforward, other evasion methods prove more challenging.

Keywords: IPv6 Security · IPv6 Transition · Covert Channels · Computer Network Operations · Red Teaming · Monitoring and Detection

1 Introduction

In this work we explore possible uses of IPv6 transition technologies for creation of covert channels over dual-stack and native IPv4 connectivity to exfiltrate information for red teaming [6] purposes. An analysis in Section 2 shows that this approach is novel and no implementations of such newly-developed tools have been identified previously.

The main contributions of this paper are:

1. two novel approaches for covert channel creation with IPv6 transition mechanisms;
2. fully self-developed proof-of-concept tools that implement the proposed methods (nc64 and tun64);
3. commonly-used protocol tunneling and developed proof-of-concept tool detection comparison table (Appendix 1.A); and

4. a reproducible virtual lab environment providing detection results using open-source network security monitoring tools.

The Internet is in a period of tremendous growth, currently evolving toward the Internet of Anything (IoA). The more widely-deployed IPv4 standard and IPv6 are incompatible, and they can communicate only via transition mechanisms and technologies [37] [43]. This introduces an additional layer of complexity and inherent security concerns for the transition and co-existence period [1]. The adoption of IPv6, and availability per the core backbone of the Internet infrastructure and edge networks, varies [10] [12]. IPv6 launch campaigns rapidly increased the number of autonomous systems (AS) announcing IPv6 prefix[3] [4]. Nevertheless, connecting to the IPv6 Internet while maintaining scalability and minimal overall complexity would require edge networks to depend on transition mechanisms [43], possibly meaning that local area networks (LANs) will continue to use primary IPv4 for an undefined period.

IPv6 protocol implementations and security solutions are relatively new, already supported by default by modern operating systems, and have not yet reached the level of acceptable quality and maturity [43] [15]. The lack of expertise and technological maturity result in IPv6 being considered in most cases as a "back-door" protocol, allowing evasion of security mechanisms [21] [23]. This is important particularly when an attack originates from inside the network, as network security devices are commonly configured and placed on the perimeter under the assumption that intruders will always come from outside [38].

In the age of advanced high-profile targeted attacks executed by sophisticated and resourceful adversaries, IPv6 is seen as an additional vector for persistent and covert attacks [40] [29]. The length of the transition period cannot be estimated, and it can be assumed that even once the entire Internet is native IPv6, there will still be systems running deprecated IPv6 functionality specifications, or heritage transition mechanisms.

Our research shows that current Network Intrusion Detection System (NIDS) solutions have serious drawbacks for handling IPv6 traffic. To address these shortcomings would require effort to redevelop the principles how NIDSs detect malicious behavior and correlate separate or encapsulated sessions. The described IPv6 transition-based covert channel creation approaches use both IP version implementations in the same protocol stack making it harder to correlate and attribute them to the same covert channel. The common protocol tunneling approaches (e.g. SSH, DNS) in comparison with the developed covert channel proof-of-concept tools (i.e. nc64 and tun64) would be easier to detect by an automated solution or human analyst since their behavior pattern is well known and understood.

In this paper, Section 2 reviews background and related work, evasion mechanisms, and covert channels; Section 3 describes common protocol tunneling

[3] IPv6 Enabled Networks, RIPE NCC. `http://v6asns.ripe.net/v/6` (Accessed 15/04/2016)

[4] IPv6 CIDR Report. `http://www.cidr-report.org/v6/as2.0/` (Accessed 15/04/2016)

approaches and newly-developed attack tool implementation and design; Section 4 describes the attack scenario, simulation environment, and generated test cases; Section 5 discusses experiment execution results (presented in Table 1), and additionally gives recommendations for such attack detection and mitigation possibilities; and Section 6 offers conclusions.

2 Background and related previous work

The aim for IPv6 was to evolve and eliminate the technical drawbacks and limitations of the IPv4 standard. However, IPv6 reintroduced almost the same security issues and, moreover, added new security concerns and vulnerabilities [11] [19]. Current IPv6 attack tools, such as the *THC-IPv6* [21], *SI6-IPv6*[5], *Topera*[6], and *Chiron*[7] toolkits, include the majority of techniques for abuse of IPv6 vulnerabilities, and can be utilized for network security assessment and IPv6 implementation verification.

The research paper by Ptacek and Newsham [33] proved that NIDS evasions are possible and pose a serious threat. A proof-of-concept tool, *v00d00N3t*, for establishment of covert channels over ICMPv6 [28] has demonstrated the potential for such approach, though it has not been released publicly. Techniques for evading NIDS based on mobile IPv6 implementations reveal that it is possible to trick NIDS using dynamically-changing communication channels [9]. Also, it could be viable to create a covert channel by hiding information within IPv6 and its extension headers [26]. Network intrusion detection system (NIDS) and firewall evasions based on IPv6 packet fragmentation and extension header chaining attacks, have been acknowledged [1] [2] [21]. Although current Requests for Comments (RFCs) have updated the processing of IPv6 atomic fragments [17], discarding overlapping fragments [24] and enforcing security requirements for extension headers [25] [20], these attacks will remain possible in the years ahead as vendors and developers sometimes fail to follow the RFC requirements or implement their own interpretation of them. General approaches for NIDS evasions have been described and analyzed [32] [3] [42] [7], with the basic principles behind evasions based on the entire TCP/IP protocol stack. Advanced evasion techniques (AETs) involve creating combinations of multiple atomic evasion techniques, potentially allowing evasion of detection by the majority of NIDS solutions [30]. Evasions are possible due to NIDS design, implementation and configuration specifics, and low network latency requirements [15].

Identified existing approaches and technologies consider native IPv6 network implementation and connectivity, and do not take into account possible methods for network security device evasions and covert channel establishment over IPv6 transition mechanisms, in order to reach the command and control (CnC) servers

[5] SI6 Networks' IPv6 Toolkit. `http://www.si6networks.com/tools/ipv6toolkit/` (Accessed 10/11/2015)

[6] Topera IPv6 analysis tool: the other side. `http://toperaproject.github.io/topera/` (Accessed 10/11/2015)

[7] Chiron. `http://www.secfu.net/tools-scripts/` (Accessed 10/11/2015)

over IPv4 only or dual-stack Internet connectivity. Moreover, to our knowledge no publicly available tools directly implement transition technology-based attacks. Here we address this gap and advance beyond previous work.

3 Covert channel implementations

3.1 Protocol tunneling

Protocol tunneling and IPv6 tunneling-based transition mechanisms pose a major security risk, as they allow bypassing of improperly-configured or IPv4-only network security devices [35] [11] [22] [23] [18]. IPv6 tunnel-based transition mechanisms, as well as general tunneling approaches (e.g. HTTP, SSH, DNS, ICMP, IPsec), can bypass network protection mechanisms. However, IPv6 tunnels simply add to the heap of possible tunneling mechanisms, leading to unmanaged and insecure IPv6 connections [35]. Moreover, dual-stack hosts and Internet browsers favor IPv6 over IPv4 [10]. Various protocol tunneling approaches can be used to set up a covert channel by encapsulating exfiltrated information in networking protocols. Covert channels based on DNS, HTTP(S), ICMP [5], and SSH [13] protocol tunneling implementations are acknowledged here as the most common approaches for eluding network detection mechanisms, due to both their frequent use and standard network policy, which allows outbound protocols and ports for user requirements and remote network administration needs. For the purposes of the NIDS test cases we consider already developed and publicly available tools herein.

3.2 Proof-of-concept nc64 tool

We have developed a proof-of-concept tool, nc64[8], for the creation of information exfiltration channel over dual-stack networks using sequential IPv4 and IPv6 sessions. The tool's source code is publicly available under MIT license.

Signature-based IDSs reassemble packets and data flows, in order to conduct inspection against a known signature database. We have observed that data reassembly by IDS is carried out on a per-session basis (e.g. a TCP session). If the data are sent in sequential sessions over different network or transport layer protocols interchangeably, the IDS tries to reassemble data based only on a specific protocol and session, and therefore cannot retrieve the full information to evaluate whether the traffic is malicious. In such scenario NIDS has to be context aware in order to be able to correlate and reconstruct the original stream from multiple sequential ones, which is very challenging to be achieved. While any set of networking protocols could be used for a sequential session creation, the security, transition, and immaturity of IPv6 makes it a preferred choice. When considering NIDS separate session correlation likely possibilities, IP protocol switching would make it harder to perform such correlation since destination IPv4 and IPv6 addresses are entirely different. In a dual-stack operating system,

[8] nc64 `https://github.com/lockout/nc64` (Accessed 12/03/2016)

IPv4 and IPv6 protocols are implemented side by side, thus adding a layer of separation between the two standards and making it more difficult for IDSs to carry out a data reassembly. Additionally, a single host can have multiple global IPv6 addresses, making the correlation to a single host even more harder.

To exfiltrate data from the source host to a destination CnC server over sequential IPv4 and IPv6 sessions, the data must be split into smaller chunks (i.e. up to IPv6 MTU of 1500B). Alternation between IPv4 and IPv6 per session has to be controlled to minimize the amount of information that is sent over a single IP protocol in successive sessions (e.g. not allowing three or more sequential IPv4 sessions). This control would avoid partial reassembly and deny successful payload inspection by NIDS.

A CnC server has both IPv4 and IPv6 addresses on which it listens for incoming connections. Once the connection is established, the listener service receives sessions and reassembles data in sequence of reception. This can be hard to accomplish if a stateless transport layer protocol is being used (i.e. UDP) or data chunk size exceeds the maximum path MTU (e.g. causing packet fragmentation).

Our proof-of-concept tool, nc64, is written in Python 3 using standard libraries. It implements the aforementioned principles, and additionally:

1. provides both the listener server and client part in one Python module;
2. accepts user-specified data from a standard input, which provides flexibility and freedom of usage;
3. requires both IPv4 and IPv6 addresses for the destination CnC listener, and can have a list of IPv6 addresses in case the CnC server has multiple IPv6 addresses configured;
4. supports UDP and TCP transport layer protocols, as these are the main ones used in computer networks;
5. enables the destination port to be freely selected to comply with firewall egress rules and match the most common outbound protocol ports (e.g. HTTP(S), DNS), and also allows for setting and randomizing of the source port for UDP-based communications;
6. provides payload Base64 encoding for binary data transmission, and to some degree can be treated as obfuscation if the IDS does not support encoding detection and decoding. It has to be noted that Base64-encoded traffic might reveal the exfiltrated data in the overall traffic since it would stand out, which would also apply when using payload encryption;
7. allows for the setting and randomizing of timing intervals between sequential sessions for an additional layer of covertness and to mitigate possible timing pattern prediction and detection by NIDS;
8. implements control over how many sequential sessions of the same protocol can be tolerated before forcing a switch to the other protocol, ensuring that small files are sent over both IP protocols; and
9. supports additional debugging features, exfiltrated data hash calculation, and transmission statistics.

3.3 Proof-of-concept tun64 tool

We have developed a second proof-of-concept tool, tun64[9], which exfiltrates information by abusing tunneling-based IPv6 transition mechanism capabilities over the IPv4-only computer network. The tool's source code is publicly available under MIT license.

Most tunneling-based IPv6 transition mechanisms rely on IPv4 as a link layer by using 6in4 encapsulation [31], whereby an IPv6 packet is encapsulated in IPv4 and the protocol number is set to decimal value 41 (the IANA-assigned payload type number for IPv6). Besides 6in4 encapsulation, we also acknowledge GRE (protocol-47) [14] as an applicable encapsulation mechanism for 6in4-in-GRE double encapsulation. When 6in4 (protocol-41) encapsulation is used, duplex connectivity might not be possible if the network relies on strict NAT. However, for the attack scenario considered in this paper, a one-way communication channel for information exfiltration to the CnC server is sufficient, making UDP the preferred transport layer protocol [34].

Most of the transition techniques cannot solve transition problems and hence are not appropriate for real-world implementation and widespread deployment [43]. Even though tunnel-based transition approaches are considered deprecated by the IETF, nevertheless some of these technologies continue to be supported by modern operating systems and ISPs. The 6over4 [8], ISATAP [39] [36], and 6to4 [27] [41] transition mechanisms were selected for implementation in our proof-of-concept tool for tunneling-based information exfiltration. Selection of these mechanisms was based upon the tunnel establishment from the target host or network, their support by either operating systems or local network infrastructure devices [36].

Our proof-of-concept tool, tun64, is written in Python 2 using the Scapy library[10]. It implements the aforementioned principles and additionally:

1. provides only the client part, thus relying on standard packet capture tools for reception and reassembly (e.g. tcpdump, Wireshark, tshark);
2. supports TCP, UDP, and SCTP as transport layer protocols;
3. emulates 6over4, 6to4, and ISATAP tunneling by assigning source and destination IPv6 addresses according to the transition protocol specification;
4. enables usage of 6to4 anycast relay routers if the tool is being tested in real Internet conditions, although in our simulated network, 6to4 relay routers or agents are not implemented;
5. allows additional GRE encapsulation to create a 6in4-in-GRE double encapsulated packet, which may allow obfuscation if the NIDS is not performing a full packet decapsulation and analysis;
6. gives an option to freely specify source and destination ports, in order to comply with firewall egress rules; and
7. supports sending a single message instead of files or standard input, a functionality designed with proof-of-concept approach in mind.

[9] tun64 `https://github.com/lockout/tun64` (Accessed 12/03/2016)
[10] Scapy project. `http://www.secdev.org/projects/scapy/` (Accessed 10/11/2015)

4 Testing environment and test description

4.1 Attack scenario

Our testing environment and experiments are designed according to the following scenario. The attack target is a small- to medium-sized research organization (up to 100 network nodes). Research organization assumes it is running an IPv4-only network, even though all the network hosts are dual-stack and their ISP just recently started to provide also IPv6 connectivity. Network administrators have implemented IPv4 security policies and only the following most common egress ports and services are allowed through the firewall: DNS (udp/53, tcp/53), HTTP (tcp/80), HTTPS (tcp/443), SSH (tcp/22), and ICMP (echo). All network hosts can establish a direct connection to the Internet without proxies or any other connection handlers. This organization was recently contracted by government to conduct advanced technological research and therefore has sensitive information processed and stored on the network hosts and servers. A red team, assuming the role of reasonably sophisticated attacker with persistent foothold in the research organization's network, is tasked to exfiltrate sensitive information from the target network. The red team has a selection of tools available at its disposal for the establishment of a covert information exfiltration channel, as described in Section 3.

4.2 Testing environment

To ensure reproducibility of the testbed, we created several *bash* scripts that leverage the Vagrant[11] environment automation tool. The scripts are publicly available in a GitHub repository[12]. A network map of the virtual testing environment is presented in Fig. 1.

The host and CnC devices were built on 32-bit Kali Linux 2.0, which comes bundled with several tunneling tools. Router1 served as the gateway for the target organization, and Router2 as an ISP node in the simulated Internet (SINET). Both routers were also built as authoritative DNS servers to facilitate usage of the Iodine tool, which was explicitly configured to query them during the tests. Two monitoring machines were built to provide detection capability. The first node was connected with a tap to the network link between the routers and all packets were copied to its monitoring interface. Second node was created to avoid conflicts between monitoring tools, and was therefore not used for capture.

In order to create identical testing conditions, we decided to store a packet capture (PCAP) file for each combination of the exfiltration tool, destination port number, transport layer protocol, and IP version. Additionally, several distinct operation modes were tested for the nc64 (e.g. both plain-text and base64 encoded payload) and tun64 (e.g. ISATAP, 6to4, and 6over4 tunneling mechanism emulation) tools, as these significantly impact the nature of the network

[11] Vagrant. `https://www.vagrantup.com/` (Accessed 07/12/2015)

[12] Automated virtual testing environment. `https://github.com/markuskont/exfil-testbench` (Accessed 07/12/2015)

traffic. Overall, 126 packet capture files were generated to be used as test cases. In the next phase we used the same monitoring nodes to run a selection of popular detection tools which would analyze these PCAP files, produce connection logs, and possibly generate alerts for suspicious activity.

We considered a number of open-source monitoring tools that are often used for network security analysis. These include the signature-based NIDSs Snort[13] and Suricata[14], as well as the network traffic analyzers Bro[15] and Moloch[16]. For Suricata, we used the Emerging Threats (ET) ruleset, while for Snort we experimented with rulesets from both SourceFire (SF) and ET signature providers. In our tests, the data exfiltrated from the host system comprise the highly sensitive */etc/shadow* file and the *root* user's private *SSH* cryptographic keys. Both of which could be used for gaining unauthorized access to potentially many other systems in the organization.

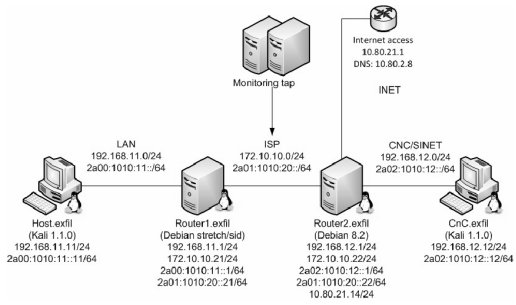

Fig. 1. Testing environment network map

5 Experiment execution and discussion of results

The results of the experiments are presented in an extensive table (see Table 1 in Appendix 1.A). Each row in the table describes a single attack, while the columns represent a detection tool that was used to attempt its detection. In our results, we distinguished four potential outcomes for a test:

1. a positive match (denoted by letter *Y* and a green cell in the table) was clearly identified as malicious activity with appropriate alerts;

[13] Snort v2.9.8.0. `http://manual.snort.org/` (Accessed 07/12/2015)
[14] Suricata v2.1beta4. `http://suricata-ids.org/docs/` (Accessed 07/12/2015)
[15] Bro v2.4.1 `https://www.bro.org/documentation/index.html` (Accessed 07/12/2015)
[16] Moloch v0.12.1. `https://github.com/aol/moloch` (Accessed 07/12/2015)

2. a partial or abnormal footprint (*P* and yellow cell) which raised an alert, but the alert did not describe the activity appropriately;
3. a potential visible match (*V* and orange cell) from connection logs which requires human analysis or sophisticated anomaly detection for a positive match; and
4. in the worst case, no visible alerts nor connection logs were generated (*N* and red cell).

Firstly, we observed that any exfiltration tool utilizing a specific application layer protocol should adhere to its standard port numbers if the malicious user aims to evade detection. For example, a HTTP tunnel on port 22 triggered an *outbound SSH Scan* alert with the ET ruleset, whereas when port 80 was used, only HTTP connection logs were generated such that we classified the attack as being only *visible*. Note that we marked the *outbound SSH Scan* alert for the HTTP tunnel on port 22 only as a *partial* match because it was incorrectly identified as an outbound SSH connection. Additionally, the same rule was responsible for a partial match against the nc64 technique on port 22. Furthermore, an alert was raised if a SSH header was detected on port 443, or if that port was used to send unencrypted HTTP traffic. Similarly, if abnormal (non-DNS) traffic was identified on UDP port 53, the ET ruleset triggered alerts for either *non-compliant traffic to DNS protocol*, or for being *overly aggressive* (i.e., having too many connections). These signatures were easily bypassed if TCP port 53 was used. However, it has to be noted that most server applications can be bound to any applicable port number (e.g. SSH on tcp/2022, HTTPS console over tcp/8443), and thus can potentially be used to avoid detection.

The difference between SF and ET rulesets in their default configurations is significant. The former seems to focus solely on perimeter intrusions, and hence could not detect any malicious outbound traffic in our tests. Furthermore, the ET ruleset produced slightly different results in Snort and Suricata. Most importantly, the former could clearly identify Ptunnel as the tool used for traffic exfiltration. Bro does not employ any traditional signatures like Snort or Suricata, but does create logs for all identified connections. As such, it was able to produce log records of all test cases. However, although Bro does not generate alerts, it does have an interesting log file named *weird.log* wherein a record of detected anomalous connections is kept. In fact, during our attacks, several *weird.log* records were generated for non-compliant traffic on port 53. Additionally, Bro's SSH connection parser malfunctioned while processing non-standard traffic, and abnormal logs could be observed in the detection system. Moloch provides no alerts, but is designed as a packet capture, indexing and visualization tool. In the most recent release, at the time of conducting the experiment, Moloch does not support IPv6 due to various limitations when indexing 128-bit IP addresses[17]. Therefore, IPv6-only iterations were unnoticed while IPv4 sessions generated by nc64 in dual-stack configuration were *visible*. The *t6to4* mode

[17] Moloch 0.14.0 2016/05/08 CHANGELOG specifies a notice that "[IPv6] support is experimental, and will change with ES 5.0." `https://github.com/aol/moloch/blob/master/CHANGELOG` (Accessed 16/08/2016)

in tun64 encapsulates the IPv6 packet as payload making it visible in IPv4 indexing system. This was observed only in cases of TCP connections without additional GRE encapsulation.

From the executed test results, detection of malicious activity by NIDS rules was based predominantly on the direction of network traffic, protocol, and destination port. This detection approach is generally favored because it uses resources (e.g. CPU, RAM) efficiently, with an expensive payload analysis attempted only after the preceding match conditions are achieved. In most cases, the nc64 tool avoided being detected, and Table 1 shows which protocol/port combinations can be used to minimize detection by selected NIDS solutions. In comparison with other exfiltration tools, nc64 performed very well on avoiding rule-based detection, and moreover could potentially elude payload inspection. In contrast, the tun64 tool was detected in the majority of cases, since protocol-41 and protocol-47 triggered the rules and generated warning messages by NIDSs. 6to4 tunneling emulation was detected when TCP or 6in4-in-GRE encapsulation was used, suggesting that double encapsulation is considered more suspicious. However, if an organization relies on IPv6 tunneling-based transition mechanisms utilizing 6in4 or GRE encapsulation, such warnings might be silenced or ignored by network-monitoring personnel. In contrast to other tunneling tools the approach taken by tun64 is feasible only if the network conditions comply with the specific operational requirements.

6 Conclusions

In this paper, the authors addressed a fundamental problem which could allow to bypass NIDSs by using the IPv6 tunneling-based and dual-stack transition mechanisms in a certain way. The proof-of-concept tools were prototyped to further verify under which circumstances the evasion of major open-source and commercial NIDS and monitoring solutions would be possible. Developed tools, tested along side with other well known protocol tunneling tools, proved to be able to evade detection and addressed certain shortcomings in the core principles of how modern NIDSs work.

It has to be noted, that any reasonably sophisticated method for exfiltrating data will be hard to detect in real-time by existing NIDSs, especially in situations where the data is split into smaller chunks and the resulting pieces use different connections or protocols (e.g. IPv4 and IPv6). Detecting such activity would require the capability to correlate the detection information in near real-time across different flows. This is theoretically possible, but would most likely incur a significant performance penalty and an increased number of false positives. There are several possibilities to attempt correlating flows using both IPv4 and IPv6 protocols. If the destination host (i.e. CnC) used in multi-protocol exfiltration has a DNS entry for both A and AAAA records, it would be possible to perform a reverse lookup to identify that the connections are going to the same domain name using IPv4 and IPv6 protocols simultaneously. This should not happen under normal circumstances, since IPv6 is usually the preferred proto-

col on dual-stack hosts. Another option would be to rely on source NIC MAC address for aggregating and correlating flows from both IPv4 and IPv6 which are originating from the the same network interface. Note, that this requires capturing the traffic from the network segment where the actual source node resides, otherwise source MAC address might get overwritten by network devices in transit. One caveat still remains — distinguishing the flows which are belonging together, especially on busy hosts with many connections. Finally, behavior based detection (e.g. unexpected traffic, malformed packets, specification non-compliance) would provide a way to detect such evasions, at the same time introducing a significant amount of false positives.

It has to be noted that any commercial product which uses an open-source tool for data acquisition is subjected to same limitations of the respective tool. Also, the lack of knowledge regarding IPv6 exploitation methods translate into low customer demand which leads to insufficient IPv6 support in final products. Furthermore, any reasonably sophisticated data exfiltration method which splits that data into smaller chunks and extracts the resulting pieces using different connections/flows (e.g. IPv4 and IPv6) will be very hard to detect in real-time by existing NIDS, which typically lack any capability to correlate across different connections/flows. Finally, commercial tools are often too expensive for small and medium sized organizations. Therefore, we did not consider these products in our final evaluation.

Authors believe, that the tendency of use of IPv6 in attack campaigns conducted by sophisticated malicious actors is going to increase; this is also recognized as an increasing trend by the security reports [16] [15] [4]. Since IPv6 security aspects are being addressed by protocol RFC updates and deprecation of obsolete transition mechanisms, it would be required to focus on these issues at the security solution developer (i.e. vendor) and implementer (i.e. consumer) levels. Adding IPv6 support to the security devices would not solve this problem, since fundamental changes would be required in the way how network traffic is interpreted and parsed, while being able to trace the context of various data streams and perform their correlation. Also, end-users should know how to properly configure, deploy and monitor security solutions in order to gain maximum awareness of the computer network flows under their direct supervision.

7 Acknowledgements

This research was conducted with the support of NATO Cooperative Cyber Defense Center of Excellence. The authors would like to acknowledge the valuable contribution of Leo Trukšāns, Walter Willinger, and Merike Kaeo.

Appendix 1.A

Table 1: Protocol tunneling and data exfiltration tool assessment

Iteration	IP Version	Protocol	Port	Snort SF	Snort ET	Suricata	Bro	Moloch
http-22	4	TCP	22	N	P	P	P	V
http-443	4	TCP	443	N	Y	Y	V	V
http-53	4	TCP	53	N	Y	Y	P	V
http-80	4	TCP	80	N	N	V	V	V
Iodine	4	UDP	53	N	N	Y	P	V
nc64-t-22-4-b64	4	TCP	22	N	P	P	V	V
nc64-t-22-4	4	TCP	22	N	P	P	V	V
nc64-t-22-64-b64	4+6	TCP	22	N	P	P	V	V
nc64-t-22-64	4+6	TCP	22	N	P	P	V	V
nc64-t-22-6-b64	6	TCP	22	N	P	P	V	N
nc64-t-22-6	6	TCP	22	N	P	P	V	N
nc64-t-443-4-b64	4	TCP	443	N	N	N	V	V
nc64-t-443-4	4	TCP	443	N	N	N	V	V
nc64-t-443-64-b64	4+6	TCP	443	N	N	N	V	V
nc64-t-443-64	4+6	TCP	443	N	N	N	V	V
nc64-t-443-6-b64	6	TCP	443	N	N	N	V	N
nc64-t-443-6	6	TCP	443	N	N	N	V	N
nc64-t-53-4-b64	4	TCP	53	N	N	N	P	V
nc64-t-53-4	4	TCP	53	N	N	N	P	V
nc64-t-53-64-b64	4+6	TCP	53	N	N	N	P	V
nc64-t-53-64	4+6	TCP	53	N	N	N	P	V
nc64-t-53-6-b64	6	TCP	53	N	N	N	P	N
nc64-t-53-6	6	TCP	53	N	N	N	P	N
nc64-t-80-4-b64	4	TCP	80	N	N	N	P	V
nc64-t-80-4	4	TCP	80	N	N	N	P	V
nc64-t-80-64-b64	4+6	TCP	80	N	N	N	P	V
nc64-t-80-64	4+6	TCP	80	N	N	N	P	V
nc64-t-80-6-b64	6	TCP	80	N	N	N	P	N
nc64-t-80-6	6	TCP	80	N	N	N	P	N
nc64-u-22-4-b64	4	UDP	22	N	N	N	V	V
nc64-u-22-4	4	UDP	22	N	N	N	V	V
nc64-u-22-64-b64	4+6	UDP	22	N	N	N	V	V
nc64-u-22-64	4+6	UDP	22	N	N	N	V	V
nc64-u-22-6-b64	6	UDP	22	N	N	N	V	N
nc64-u-22-6	6	UDP	22	N	N	N	V	N
nc64-u-443-4-b64	4	UDP	443	N	N	N	V	V
nc64-u-443-4	4	UDP	443	N	N	N	V	V
nc64-u-443-64-b64	4+6	UDP	443	N	N	N	V	V
nc64-u-443-64	4+6	UDP	443	N	N	N	V	V
nc64-u-443-6-b64	6	UDP	443	N	N	N	V	N
nc64-u-443-6	6	UDP	443	N	N	N	V	N

Table 1: Protocol tunneling and data exfiltration tool assessment

Iteration	IP Version	Protocol	Port	Snort SF	Snort ET	Suricata	Bro	Moloch
nc64-u-53-4-b64	4	UDP	53	N	Y	Y	P	V
nc64-u-53-4	4	UDP	53	N	Y	Y	P	V
nc64-u-53-64-b64	4+6	UDP	53	N	Y	Y	P	V
nc64-u-53-64	4+6	UDP	53	N	Y	Y	P	V
nc64-u-53-6-b64	6	UDP	53	N	Y	Y	P	N
nc64-u-53-6	6	UDP	53	N	Y	Y	P	N
nc64-u-80-4-b64	4	UDP	80	N	N	N	V	V
nc64-u-80-4	4	UDP	80	N	N	N	V	V
nc64-u-80-64-b64	4+6	UDP	80	N	N	N	V	V
nc64-u-80-64	4+6	UDP	80	N	N	N	V	V
nc64-u-80-6-b64	6	UDP	80	N	N	N	V	N
nc64-u-80-6	6	UDP	80	N	N	N	V	N
netcat-t-22-4	4	TCP	22	N	N	N	V	V
netcat-t-22-6	6	TCP	22	N	N	N	V	N
netcat-t-443-4	4	TCP	443	N	N	N	V	V
netcat-t-443-6	6	TCP	443	N	N	N	V	N
netcat-t-53-4	4	TCP	53	N	N	N	P	V
netcat-t-53-6	6	TCP	53	N	N	N	P	N
netcat-t-80-4	4	TCP	80	N	N	N	V	V
netcat-t-80-6	6	TCP	80	N	N	N	V	N
netcat-u-22-4	4	UDP	22	N	N	N	V	V
netcat-u-22-6	6	UDP	22	N	N	N	V	N
netcat-u-443-4	4	UDP	443	N	N	N	V	V
netcat-u-443-6	6	UDP	443	N	N	N	V	N
netcat-u-53-4	4	UDP	53	N	Y	Y	P	V
netcat-u-53-6	6	UDP	53	N	Y	Y	P	N
netcat-u-80-4	4	UDP	80	N	N	N	V	V
netcat-u-80-6	6	UDP	80	N	N	N	V	N
ptunnel	4	ICMP		N	Y	N	V	V
ssh-4-22	4	TCP	22	N	N	V	V	V
ssh-4-443	4	TCP	443	N	Y	Y	V	V
ssh-4-53	4	TCP	53	N	N	V	V	V
ssh-4-80	4	TCP	80	N	N	V	P	V
ssh-6-22	6	TCP	22	N	N	V	P	N
ssh-6-443	6	TCP	443	N	Y	Y	P	N
ssh-6-53	6	TCP	53	N	N	V	P	N
ssh-6-80	6	TCP	80	N	N	V	P	N
tun64-t-22-isatap	4	TCP	22	N	Y	Y	P	N
tun64-t-22-t6over4	4	TCP	22	N	Y	Y	P	N
tun64-t-22-t6to4	4	TCP	22	N	Y	Y	P	V
tun64-t-443-isatap	4	TCP	443	N	Y	Y	P	N

Table 1: Protocol tunneling and data exfiltration tool assessment

Iteration	IP Version	Protocol	Port	Snort SF	Snort ET	Suricata	Bro	Moloch
tun64-t-443-t6over4	4	TCP	443	N	Y	Y	P	N
tun64-t-443-t6to4	4	TCP	443	N	Y	Y	P	V
tun64-t-53-isatap	4	TCP	53	N	Y	Y	P	N
tun64-t-53-t6over4	4	TCP	53	N	Y	Y	P	N
tun64-t-53-t6to4	4	TCP	53	N	Y	Y	P	V
tun64-t-80-isatap	4	TCP	80	N	Y	Y	P	N
tun64-t-80-t6over4	4	TCP	80	N	Y	Y	P	N
tun64-t-80-t6to4	4	TCP	80	N	Y	Y	P	V
tun64-u-22-isatap	4	UDP	22	N	Y	Y	P	N
tun64-u-22-t6over4	4	UDP	22	N	Y	Y	P	N
tun64-u-22-t6to4	4	UDP	22	N	Y	Y	P	N
tun64-u-443-isatap	4	UDP	443	N	Y	Y	P	N
tun64-u-443-t6over4	4	UDP	443	N	Y	Y	P	N
tun64-u-443-t6to4	4	UDP	443	N	Y	Y	P	N
tun64-u-53-isatap	4	UDP	53	N	Y	Y	P	N
tun64-u-53-t6over4	4	UDP	53	N	Y	Y	P	N
tun64-u-53-t6to4	4	UDP	53	N	Y	Y	P	N
tun64-u-80-isatap	4	UDP	80	N	Y	Y	P	N
tun64-u-80-t6over4	4	UDP	80	N	Y	Y	P	N
tun64-u-80-t6to4	4	UDP	80	N	Y	Y	P	N
tun64-t-22-isatap-gre	4	TCP	22	N	Y	Y	P	N
tun64-t-22-t6over4-gre	4	TCP	22	N	Y	Y	P	N
tun64-t-22-t6to4-gre	4	TCP	22	N	Y	Y	P	V
tun64-t-443-isatap-gre	4	TCP	443	N	Y	Y	P	N
tun64-t-443-t6over4-gre	4	TCP	443	N	Y	Y	P	N
tun64-t-443-t6to4-gre	4	TCP	443	N	Y	Y	P	V
tun64-t-53-isatap-gre	4	TCP	53	N	Y	Y	P	N
tun64-t-53-t6over4-gre	4	TCP	53	N	Y	Y	P	N
tun64-t-53-t6to4-gre	4	TCP	53	N	Y	Y	P	V
tun64-t-80-isatap-gre	4	TCP	80	N	Y	Y	P	N
tun64-t-80-t6over4-gre	4	TCP	80	N	Y	Y	P	N
tun64-t-80-t6to4-gre	4	TCP	80	N	Y	Y	P	V
tun64-u-22-isatap-gre	4	UDP	22	N	Y	Y	P	N
tun64-u-22-t6over4-gre	4	UDP	22	N	Y	Y	P	N
tun64-u-22-t6to4-gre	4	UDP	22	N	Y	Y	P	V
tun64-u-443-isatap-gre	4	UDP	443	N	Y	Y	P	N
tun64-u-443-t6over4-gre	4	UDP	443	N	Y	Y	P	N
tun64-u-443-t6to4-gre	4	UDP	443	N	Y	Y	P	V
tun64-u-53-isatap-gre	4	UDP	53	N	Y	Y	P	N
tun64-u-53-t6over4-gre	4	UDP	53	N	Y	Y	P	N
tun64-u-53-t6to4-gre	4	UDP	53	N	Y	Y	P	V

Table 1: Protocol tunneling and data exfiltration tool assessment

Iteration	IP Version	Protocol	Port	Snort SF	Snort ET	Suricata	Bro	Moloch
tun64-u-80-isatap-gre	4	UDP	80	N	Y	Y	P	N
tun64-u-80-t6over4-gre	4	UDP	80	N	Y	Y	P	N
tun64-u-80-t6to4-gre	4	UDP	80	N	Y	Y	P	V

References

1. Atlasis, A.: Attacking IPv6 Implementation Using Fragmentation. Tech. rep., Centre for Strategic Cyberspace + Security Science (2011)
2. Atlasis, A.: Security Impacts of Abusing IPv6 Extension Headers. Tech. rep., Centre for Strategic Cyberspace + Security Science (2012)
3. Atlasis, A., Rey, E.: Evasion of High-End IPS Devices in the Age of IPv6. Tech. rep., secfu.net (2014)
4. Blumbergs, B.: Technical Analysis of Advanced Threat Tactics Targeting Critical Information Infrastructure. Cyber Security Review, Winter Edition pp. 25–36 (2014)
5. Blunden, B.: The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System, 2nd ed., chap. 14. Covert Channels. Jones and Bartlett Learning (2013)
6. Brangetto, P., Çalişkan, E., Rõigas, H.: Cyber Red Teaming - Organisational, technical and legal implications in a military context. Tech. rep., NATO CCD CoE (2015)
7. Bukač, V.: IDS System Evasion Techniques. Master's thesis, Masarykova Univerzita Fakulta Informatiky (2010)
8. Carpenter, B., Jung, C.: Transmission of IPv6 over IPv4 Domains without Explicit Tunnels. RFC 2529, IETF Secretariat (March 1999), standards Track
9. Colajanni, M., Zotto, L.D., Marchetti, M., Messori, M.: Defeating NIDS evasion in Mobile IPv6 networks. In: IEEE (2011)
10. Colitti, L., Gunderson, S.H., Kline, E., Refice, T.: Evaluating IPv6 Adoption in the Internet. In: PAM 2010. pp. 141–150. Springer-Verlag (2010)
11. Convery, S., Miller, D.: IPv6 and IPv4 Threat Comparison and Best-Practice Evaluation. White paper, Cisco Systems (March 2004)
12. Czyz, J., Allman, M., Zhang, J., Iekel-Johnson, S., Osterweil, E., Bailey, M.: Measuring IPv6 Adoption. In: ACM SIGCOMM14 (2014)
13. Ellens, W., Żuraniewski, P., Sperotto, A., Schotanus, H., Mandjes, M., Meeuwissen, E.: Covert Channels in IPv6. In: Doyen, G., Waldburger, M., Čeleda, P., Sperotto, A., Stiller, B. (eds.) Flow-Based Detection of DNS Tunnels, Lecture Notes in Computer Science, vol. 7943, pp. 124–135. Springer Berlin Heidelberg (2013)
14. Farinacci, D., Li, T., Hanks, S., Meyer, D., Traina, P.: Generic Routing Encapsulation (GRE). RFC 2784, IETF Secretariat (March 2000), standards Track. Supplemented with RFC2890.
15. Fortinet: Biting the Bullet: A Practical Guide for Beginning the Migration to IPv6. white paper, Fortinet Inc. (2011)
16. G Data SecurityLabs: Uroburos: Highly complex espionage software with Russian roots. Tech. rep., G Data Software AG (February 2014)
17. Gont, F.: Processing of IPv6 "Atomic" Fragments. RFC 6946, IETF Secretariat (May 2013), standards Track. Updates RFC 2460, 5722

18. Gont, F.: Security Implications of IPv6 on IPv4 Networks. RFC 7123, IETF Secretariat (February 2014), informational
19. Gont, F., Chown, T.: Network Reconnaissance in IPv6 Networks. Tech. rep., IETF Secretariat (February 2015), internet Draft. Obsoletes RFC 5157 (if approved)
20. Gont, F., Liu, W., Bonica, R.: Transmission and Processing of IPv6 Options. Tech. rep., IETF Secretariat (March 2015), best Current Practice. Updates RFC 2460 (if approved)
21. Gont, F., Heuse, M.: Security Assessments of IPv6 Networks and Firewalls. IPv6 Congress 2013 (2013), presentation
22. of the HKSAR, G.: IPV6 Security. Tech. rep., The Government of the Hong Kong Special Administrative Region (May 2011)
23. Hogg, S., Vyncke, E.: IPv6 Security. Cisco Press (2009)
24. Krishnan, S.: Handling of Overlapping IPv6 Fragments. RFC 5722, IETF Secretariat (December 2009), standards Track. Updates RFC 2460
25. Krishnan, S., Woodyatt, J., Kline, E., Hoagland, J., Bhatia, M.: A Uniform Format for IPv6 Extension Headers. RFC 6564, IETF Secretariat (April 2012), standards Track. Updates RFC 2460
26. Lucena, N.B., Lewandowski, G., Chapin, S.J.: Covert Channels in IPv6. In: Danezis, G., Martin, D. (eds.) PET 2005. pp. 147–166. Springer-Verlag (2006)
27. Moore, K.: Connection of IPv6 Domains via IPv4 Clouds. RFC 3056, IETF Secretariat (February 2001), standards Track
28. Murphy, R.: IPv6 / ICMPv6 Covert Channels. DEF CON'14 (2014), presentation
29. National Cybersecurity and Communications Integration Center: ICS-CERT Monitor. Tech. rep., US Dep. of Homeland Security (December 2013)
30. Niemi, O.P., Levomki, A., Manner, J.: Dismantling Intrusion Prevention Systems. In: ACM SIGCOMM12 (August 2012)
31. Nordmark, E., Gilligan, R.: Basic Transition Mechanisms for IPv6 Hosts and Routers. RFC 4213, IETF Secretariat (October 2005), standards Track
32. Pastrana, S., Montero-Castillo, J., Orfila, A.: Advances in Security Information Management: Perceptions and Outcomes, chap. 7. Evading IDSs and firewalls as fundamental sources of information in SIEMS. Nova Science Publishers (January 2013)
33. Ptacek, T.H., Newsham, T.N.: Insertion, evasion, and denial of service: Eluding network intrusion detection. Tech. rep., DTIC Document (January 1998)
34. Sarrar, N., Maier, G., Ager, B., Sommer, R., Uhlig, S.: Investigating IPv6 Traffic: What Happened at the World IPv6 Day? In: Taft, N., Ricciato, F. (eds.) 13th International Conference, PAM 2012. pp. 11–20. Springer-Verlag (March 2012)
35. S.Degen et.al.: Testing the security of IPv6 implementations. Tech. rep., Ministry of Economic Affairs of the Netherlands (March 2014)
36. Steffann, S., van Beijnum, I., van Rein, R.: A Comparison of IPv6-over-IPv4 Tunnel Mechanisms. RFC 7059, IETF Secretariat (November 2013), informational
37. Tadayoni, R., Henten, A.: Transition from IPv4 to IPv6. In: 23rd European Regional Conference of the International Telecommunication Society (July 2012)
38. Taib, A.H.M., Budiarto, R.: Evaluating IPv6 Adoption in the Internet. In: 5th Student Conference on Research and Development. IEEE (December 2007)
39. Templin, F., Gleeson, T., Thaler, D.: Intra-Site Automatic Tunnel Addressing Protocol (ISATAP). RFC 5214, IETF Secretariat (March 2008), informational
40. TrendLabs: Targeted Attack Trends 2014 Report. Tech. rep., TrendMicro inc. (2015)

41. Troan, O., Carpenter, B.: Deprecating the Anycast Prefix for 6to4 Relay Routers. RFC 7526, IETF Secretariat (May 2015), best Current Practice. Obsoletes RFC 3068, 6732
42. Vidal, J.M., Castro, J.D.M., Orozco, A.L.S., Villalba, L.J.G.: Evolutions of Evasion Techniques Aigainst Network Intrusion Detection Systems. In: ICIT 2013, The 6th International Conference on Information Technology (May 2013)
43. Wu, P., Cui, Y., Wu, J., Liu, J., Metz, C.: Transition from IPv4 to IPv6: A State-of-the-Art Survey. IEEE Communications Surveys and Tutorials 15(3), 1407–1424 (2013)

Appendix 5

Publication V
M. Kont, M. Pihelgas, K. Maennel, B. Blumbergs, and T. Lepik. Frankenstack: Toward real-time Red Team feedback. In *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, pages 400–405, October 2017

The paper is included in the *Proceedings of the 2017 IEEE Military Communications Conference (MILCOM 2017)*.
DOI: 10.1109/MILCOM.2017.8170852

Frankenstack: Toward Real-time Red Team Feedback

Markus Kont
NATO Cooperative Cyber Defence Centre of Excellence
markus.kont[a]ccdcoe.org

Mauno Pihelgas
NATO Cooperative Cyber Defence Centre of Excellence
mauno.pihelgas[a]ccdcoe.org

Kaie Maennel
Tallinn University of Technology
kamaen[a]ttu.ee

Bernhards Blumbergs
NATO Cooperative Cyber Defence Centre of Excellence;
IMCS UL, CERT.LV Laboratory
bernhards.blumbergs[a]cert.lv

Toomas Lepik
Tallinn University of Technology
toomas.lepik[a]ttu.ee

***Abstract*—Cyber Defense Exercises have received much attention in recent years, and are increasingly becoming the cornerstone for ensuring readiness in this new domain. Crossed Swords is an exercise directed at training Red Team members for responsive cyber defense. However, prior iterations have revealed the need for automated and transparent real-time feedback systems to help participants improve their techniques and understand technical challenges. Feedback was too slow and players did not understand the visibility of their actions. We developed a novel and modular open-source framework to address this problem, dubbed *Frankenstack*. We used this framework during Crossed Swords 2017 execution and evaluated its effectiveness by interviewing participants and conducting an online survey. Due to the novelty of Red Team-centric exercises, very little academic research exists on providing real-time feedback during such exercises. Thus, this paper serves as a first foray into a novel research field.**

***Keywords*—automation, cyber defense exercises, education, infrastructure monitoring, real-time feedback, red teaming**

I. Introduction

Cyber defense exercises (CDX) are crucial for training readiness and awareness within the *cyber domain*. This new domain is acknowledged by NATO alongside with land, sea, air, and space [1]. Alliance nations are endorsing the development of both defensive and responsive cyber capabilities. In this context, the paper focuses on further evolving the quality and learning experience of CDX, aimed at developing cyber red teaming [2] and responsive skillset. Crossed Swords (XS) [3], a techical exercise developed by NATO Cooperative Cyber Defense Centre of Excellence (NATO CCD COE) since 2014, is used as a platform to create the proposed framework. The solution is applicable to any other CDX where standard network and system monitoring capability is available.

A. Background

XS is an intense hands-on technical CDX oriented at penetration testers working as a single united team, accomplishing mission objectives and technical challenges in a virtualized environment. While common technical CDX is aimed at exercising defensive capabilities (i.e., Blue Team – BT), XS changes this notion, identifies unique cyber defense aspects and focuses on training the Red Team (RT).

To develop and execute the exercise, multiple teams are involved: rapid response team (i.e., RT); game network and infrastructure development (Green Team – GT); game scenario development and execution control (White Team – WT); defending team user simulation (i.e., BT); and monitoring (Yellow Team – YT).

The RT consists of multiple sub-teams based on the engagement specifics, those being: network attack team, targeting network services, protocols and routing; client side attack team, aiming at exploiting human operator and maintaining access to the hosts; web application attack team, targeting web services, web applications and relational databases; and digital forensics team, performing data extraction and digital artefact collection. These sub-teams must coordinate their actions, share information and cooperate when executing attacks to reach the exercise objectives.

The main goal is to exercise RT in a stealthy fast-paced computer network infiltration operation in a responsive cyber defense scenario [4]. To achieve this, the RT must uncover the unknown game network, complete a set of technical challenges and collect attribution evidence, while staying as stealthy as possible. Note that XS is not a *capture-the-flag* competition, as the RT has to pivot from one sub-objective to another in order to achieve the final mission according to the scenario. Furthermore, Red sub-teams are not competing with each other, and rather serve as specialized branches of a single unit.

B. Problem Statement

Prior XS iterations revealed several problems with RT learning experience. Primarily, the YT feedback regarding detected attacks from the event logs and network traffic was presented at the end of every day, which was not well suited to the short, fast and technical nature of the exercise. The feedback session addressed only some noteworthy observations from the day, but RT participants need direct and immediate feedback about their activity to identify mistakes as they happen. This feedback needs to be adequately detailed, so that the RT can understand why a specific attack was detected and then improve their approach. Finally, to make the feedback faster, the slowest element in the loop—the human operator—needs to be eliminated.

Therefore, manual data analysis by the YT needs to be automated as much as possible. To achieve this, we used the same open-source tools as in the previous XS iterations, but added in event correlation, a novel query automation tool, and a newly developed visualization solution. We decided to call the framework *Frankenstack*. Fig. 1 illustrates the role of Frankenstack in the XS exercise.

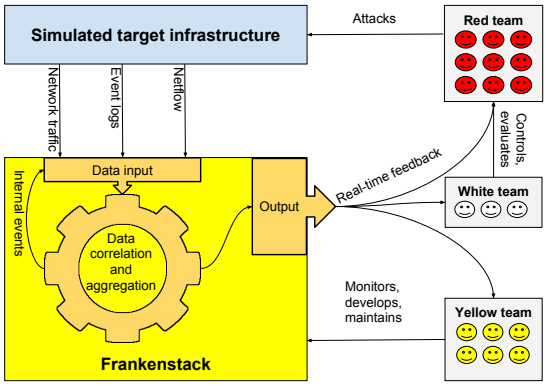

Fig. 1. High-level overview of Frankenstack

The RT has to receive timely and efficient feedback from the YT regarding detected attacks on the target systems. This feedback is critical to raise the level of stealthiness, identify the gaps of RT coordination, and analyze the tools and tactics used for computer network operations. The effectiveness of our framework was assessed during the main execution of XS 2017 (XS17), where the stack provided real-time monitoring feedback to the RT.

The remainder of the paper is organized as follows: section II provides an overview of related work, section III describes our monitoring stack, section IV presents RT feedback results, while section V discusses future work, and section VI concludes the paper.

II. Related work

For teaching purposes, the benefit of exercises and competitions is generally well accepted and documented [5], [6], [7], [8]. Unfortunately, not much research has focused on the perception of feedback which is provided to the training audience, especially in the context of monitoring technical indicators of compromise in realistic environments. Thus, this section presents research related to both measuring and improving the learning experience as well as situation awareness (SA) during cyber exercises.

Dodge et al. discussed CDX network traffic analysis in [9], a practice that is common in modern exercises not only for situational awareness (SA) but also as educational tool, for elaborating attacker campaigns, for training network analysts, etc. However, this early paper focuses on traffic capture and initial profiling, and does not consider distractions such as traffic generation, increasing infrastructure complexity, host instrumentation, data source correlation, or the need for immediate feedback. In [10], Holm et al. correlated network traffic and RT attack logs from Baltic Cyber Shield, a precursor for Locked Shields and Crossed Swords exercises. However, their goal was to improve existing metrics for vulnerability scoring, as opposed to participant education. Likewise, in [11], Brynielsson et al. conducted a similar empirical analysis on CDXs to profile attacks and create attacker personas.

In [12], Arendt et al. presented CyberPetri, a circle-packing visualization component of Ocelot, which was previously presented in [13] as a user-centered decision support visualization. They presented several use cases of the tool, but their main goal was high-level feedback to network analysts based on target system service availability reports. Although the tool was useful for high-level decision making, technical RT members are more interested in immediate effects of their attacks on target systems. Note that any single system is often a supporting pillar for more complex services, and is not noticeable to end-users. Nevertheless, modern security monitoring is built upon instrumentation of these systems, to find RT *footprints* and to trigger notification upon breaching these digital tripwires.

A paper [14] by Henshel et al. describes the assessment model for CDXs based on the Cyber Shield 2015 example, as well as integrated evaluation of metrics for assessing team proficiency. In addition to data collected during the exercise, they also conducted a pre-event expertise survey to determine possible relationships between prior expertise and exercise performance. For future assessments they suggest that near real-time analysis of the collected data is required—they stress that raw data collection is not a problem, but the capability to meaningfully analyze is the limiting factor. Manual methods do not scale with the huge amounts of incoming data. This closely coincides with our observations in section I-B and this is what we aim to improve.

Furthermore, existing academic research commonly relies on monolithic tools, which are often not accessible to the general public, thus, making experiments difficult, if not impossible, to reproduce. We seek to provide an inexpensive open-source alternative to these products. The next section describes our modular monitoring architecture.

III. Frankenstack

Commercial tools are too expensive for smaller cyber exercises, in terms of licensing fees, hardware cost and specialized manpower requirements. Detection logic in commercial tools is also not available to the general public, which hinders YT's ability to provide detailed explanations of detected attacks. Frankenstack is easy to customize as individual elements of the stack are industry standard tools which can be interchanged. Note that we opted to use a commercial tool *SpectX* as an element within Frankenstack for log filtering, due to on-site competency and developer support. However, this function could have been achieved with the open-source Elastic stack [15]. Our stack provides a clear point of reference to other researchers and system defenders who wish to compile the monitoring framework in their particular environments, as the overall architecture is novel.

The data available to us during XS included full ERSPAN (Encapsulated Remote Switched Port ANalyzer) traffic mirror from gamenet switches and NetFlow from gamenet routers. This was provided by the GT. Furthermore, we instrumented

gamenet systems to collect numerical metrics (e.g., CPU and memory usage, and network interface statistics) and logs (e.g., syslog from Linux, Event Logs from Microsoft Windows, Apache web server access logs, and audit logs from Linux command line executions). Such host instrumentations are very difficult to implement in a standard CDX with BT training focus: if the intent is to give BTs full control of a simulated infrastructure, then they also have full volition to disable these tools. However, as the XS training audience is the RT, then we could maintain control of all target systems and ensure a constant stream of monitoring data. Moreover, we complemented the list of BT data sources with various YT honeypots and decoy servers.

Detailed overview of the resulting stack, in relation to data processing pipelines, is presented in Fig. 2. The blue area represents available data sources, the gray area stands for data storage, and the yellow area denotes the YT presentation layer (i.e., visualization tools on five monitors). Blue and green elements represent target systems and all other elements outside colored boundaries are processing tools. Custom tools that we developed are highlighted with a dark yellow circle. Note that some tools, such as Moloch, are designed for both data storage and visualization, but are not presented in these respective areas because only their API components were used for processing automated queries.

We opted against using NetFlow data, as modern packet capture analyzers (e.g., Suricata, Bro, and Moloch) can fill this role, albeit by needing more processing power and memory. Additionally, these tools commonly present their output in textual log format, which we fed back into the central logging and correlation engine. Thus, the problem of identifying and displaying high-priority IDS alerts can be simplified into a log analysis problem.

Frankenstack uses event correlation for integrating various information sources as this field has been well researched in the context of log management [16], [17], [18]. We open-sourced the correlation ruleset in [19]. See Listing 1 for an example raw log entry from Snoopy Logger [20] that was converted into a more universal human-readable security event that could be presented to the general audience on various dashboards while preserving the raw message for anyone wishing to drill down. Note that specific IP addresses have been removed from this example. This generalization is necessary for handling and grouping subsequent log entries that continue describing the same event, e.g., additional commands executed on the same host via SSH.

Listing 1. Event generalization by frankenSEC

```
#INPUT
login:administrator ssh:(SRC_IP 58261 DST_IP 22)
username:administrator uid:1001 group:administrator
gid:1001 sid:6119 tty:(none) cwd:/home/administrator
filename:/usr/bin/passwd: passwd administrator

#OUTPUT
SRC_IP->[DST_IP]: Command execution by administrator
over SSH
```

Post-mortem analysis of available data sources has proven effective during prior CDXs for packet capture (PCAP) analysis, but requires a significant amount of time and manual work. Again, this clashes with the short time-frame of a CDX. Furthermore, search queries are often written ad hoc during investigations and subsequently forgotten, making analysis results difficult to reproduce. Thus, we created *Otta* [21], a novel query documentation and automation tool for periodically executing user-defined queries on large datasets and converting aggregated results into time-series metrics. *Otta* enables trend graphing, alerting, and anomaly detection for stored user-defined queries. This reduces time spent on analysis and ensures reproducibility by documenting the queries that produced the results.

We used various open-source tools for timelining metrics and log data, for displaying alerts, and presenting correlated information. There are slight differences in handling various incoming alerts. While many types of alerts (e.g., CPU and disk usage) trigger and recover automatically based on a set of thresholds, there are some types (e.g., IDS alerts) that lack the concept of a recovery threshold. Thus, the alert will never recover once raised, leading to an overabundance of information on the central dashboard. Furthermore, batch bucketing methods and timelines are lossy, as only the most frequent items are considered. The volatile nature of CDXs and an abundance of generated network traffic can therefore cause these dashboards to be too verbose to follow efficiently.

Attack maps are not usable because they rely on geographical data which is completely fictional in many CDX environments. Therefore, we developed Event Visualization Environment, or EVE, a novel web-based tool for visualizing correlated attacks in relation to gamenet infrastructure. The Alpha version of this tool has been made publicly available in [22]. EVE is a web application that shows attacks carried out by the RT in real time with a custom gamenet network map as background. Probes can send information to EVE listener in JSON format. Real-time visualization is using WebSocket technology—eliminating the need to reload the page for updated results.

EVE supports combining multiple events in a short time window, and that share the same source and destination addresses, into a unified attack. Resulting attacks are subsequently displayed as arrows connecting the circles around source and target hosts on the network map, while detailed attack information is displayed next to the network map. Using the gamenet map makes EVE a very intuitive tool for enabling participants and observers alike to comprehend CDX events on a high-level.

During the exercise EVE was available only to YT and WT members, as it revealed the entire exercise network map that RT had to discover on their own. However, EVE has a dedicated replay mode to display all the attacks condensed into a given time period, allowing participants to obtain an overview of the attacks as well as understand the pace and focus of the previous attack campaign. For instance, attacks from the previous day can be replayed in 15 minutes. EVE

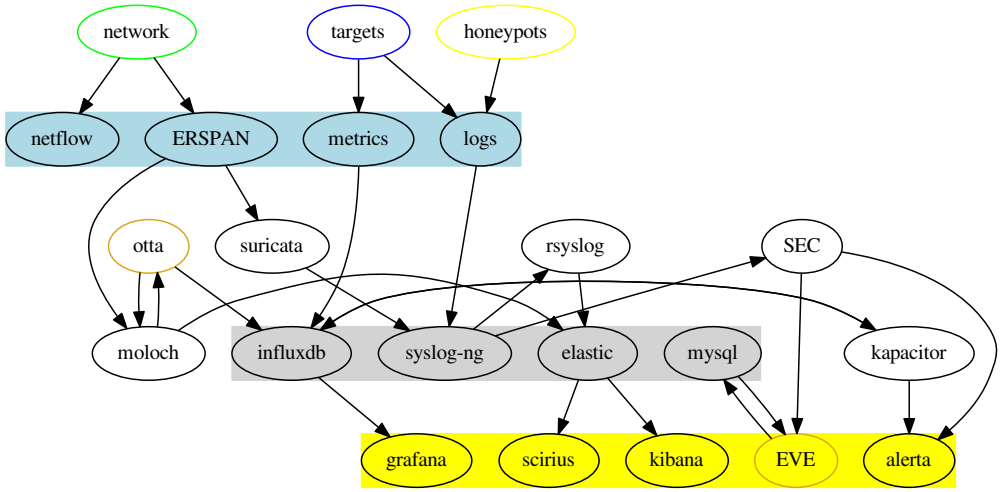

Fig. 2. Data flow between Frankenstack elements during XS17

TABLE I
DEDUPLICATION BY EVENT SOURCE

Event source	Total events	Unique events displayed	Percentage displayed
Apache	1908	35	1.83%
IDS	23790	616	2.59%
Snoopy logger	2962	40	1.35%
Total	*28660*	*691*	*2.41%*

was shown in replay mode to RT participants after the exercise concluded. This compressed replay was very effective in presenting the most prevalent problems, such as periodic beaconing during otherwise silent night periods and verbosity of particular network sub-team attacks.

Alerta [23] served as the primary dashboard to display alerts to the RT. We used the HTTP API for submitting Frankenstack events to Alerta. The RT had direct access to the Alerta web interface and could write their own filtering rules to view information relevant to their current campaign. Finally, we present Tab. I to illustrate how Frankenstack performed in deduplicating the events that were displayed to the RT on the Alerta dashboard. Note that deduplication was primarily based on the generalized event descriptions (see Listing 1).

IV. ASSESSMENT

The tools and infrastructure are essential for learning, but they do not make the exercise successful by default. Often human factors, such as how YT and RT members perceive and use the tools, have significant impact.

One essential part of the assessment was to observe the behavior of the RT members and their interaction with Frankenstack during the exercise in order to gain further insights into their progress and learning experience. We carried out qualitative interviews with RT participants, to estimate their reaction to Frankenstack and their overall learning progress. The interviews took place in casual settings during breaks in execution. Furthermore, we conducted a quantitative survey in the form of an online questionnaire. The survey consists of multiple choice or ranking style questions with the ability to provide additional comments for each question. The survey concluded by asking some general questions about meeting the training objectives and overall satisfaction with the exercise.

A. Feedback combined from interviews, survey and observations

This subsection includes the analysis of participants feedback. Improvement suggestions to learning design are presented in the following subsection IV-B.

We received 14 survey responses out of 27 participants (52%). 46% of participants had attended other exercises, but none of those exercises had attempted to provide SA via a similar toolset. The remaining 54% had not previously attended any exercise.

There were four large screens in the training room directed to the RT, displaying Alerta, Grafana, Scirius, and Suricata. A fifth screen displaying EVE was only visible to YT and WT members. Most RT members preferred to view the main screens displayed in training room, and 38% responded that they checked the screens every 60 minutes or less. Another 38% checked the screens every 30 to 50 minutes. RT members were not restricted from accessing any of the Frankenstack web interfaces. The survey revealed that learners did access the monitoring framework on their local computers when attempting new attack vectors. Thus, tools served their intended usage.

Alerta was considered most useful (46%), followed by Moloch (31%). There was no clear result for the least useful tool. The respondents expressed mixed feelings on the ease of

use of the SA tools: 38% equally agreeing and disagreeing, and the remainder (24%) being neutral.

Regarding learning impact, 79% agreed (of those 57% strongly agreed) that the SA given during exercise is useful for their learning process, while 21% were neutral. In terms of the feedback rate, 77% of the respondents considered the speed of feedback to be at the correct level, 15% considered it too slow and 8% considered it too fast. Furthermore, 57% agreed that alerts were accurate and sufficient for their learning process, while 43% were neutral about this question. However, several respondents revealed being too focused on achieving their primary objectives, and thus unable to properly switch between their tools and feedback screens.

In relation to visibility, 45% of the participants agreed that they had learned a lot about how their actions can be detected (i.e., it is useful to see simultaneously what attack method could be detected, and how), and 30% were more careful with their attacks and thus tried to be stealthier than they normally would have been. However, there were some unintended side-effects. The feedback sometimes provided insight into the network map that the RT was tasked to discover independently. For example, if the RT probes a yet unknown node on a network, the logs generated on the host might reveal the target hostname (e.g., *sharepoint* or *ftp*), which consequently implies the purpose of the system—something that would not be apparent from an IP address. Thus, there is a fine line between revealing too little or too much to the training audience.

Furthermore, some comments revealed a loss of emphasis on stealth due to exercise time constraints, i.e., RT members knowingly used more verbose techniques closer to the objective deadline. To clarify, 64% of respondents confirmed that the SA tools were not distracting them nor had negative impact, while 30% agreed that they were distracted. The remaining 6% were neutral. This confirms the challenges of providing instant feedback, as the learning potential is not fully used. The question is how this learning experience is impacting long-term behavior of the participant.

One of the key training aspects is working as a team in achieving goals. Thus, team communication and cooperation are vital. Overall, 83% of respondents indicated some improvement of the skills for these specific training objectives. However, feedback concerning the impact of SA tools on team communication and cooperation is mixed—50% perceived positive impact, whereas 21% were negative and remainder were neutral. Several respondents acknowledged less need for verbal communication, as they could see relevant information on the screens. Unfortunately, not all RT members were able to interpret and perceive this information correctly. This combined with the reduced need for communication meant that not all participants progressed as a team.

Compared to other CDXs, 50% responded that they needed less information from YT members, as they obtained relevant SA on their own. Guidance, however, is a critical success factor for learning, especially in a team setting. 64% of participants said they had sufficient help for their learning process, i.e., when they did not know how to proceed, their team members or sub-team leaders provided guidance. However, 64% is a rather disappointing result and could clearly be increased with improved learning design. Some respondents admitted that they did not know how other teams were progressing and wasted time on targets that were not vulnerable. This caused significant frustration and stress, especially when combined with the compressed timeframe of a CDX.

B. Learning improvement suggestions

Given the amount of work that goes into preparing such exercises, the level of learning potential needs to be maximized. Our analysis suggests that small learning design changes may have significant impact. This section presents the main recommendations derived from these results.

From the learning perspective, we cannot assume that participants know how to use or interpret the results. Lack of in-depth knowledge of monitoring tools (e.g., where is raw data collected, what is combined and how, what needs to be interpreted in which way, etc.) has a negative impact on learning. A dedicated training session or workshop needs to take place prior to execution. Furthermore, in the light of the survey results, inclusion of various tools into Frankenstack needs to be carefully evaluated to avoid visual distractions for RT participants. There is also a need to reduce prior system and network monitoring knowledge by making the output more self-explanatory.

Given the difficulties in switching between multiple screens whilst also trying to achieve an objective in unfamiliar network, one can easily suggest compressing the amount of presentable information to reduce the number of monitoring screens. However, this cannot be attained without reducing the amount of technical information. The purpose of Frankenstack is not to provide SA to high-level decision makers, but to present feedback to technical specialists. Thus, a better approach would be restructuring each sub-team with a dedicated monitoring station with a person manning it, allowing team members to focus on their objectives and get feedback relevant only for their actions. As such, RT members must be given a *hands-on* opportunity to use monitoring systems.

In RT exercises such as XS, there are several main objectives to be achieved by the whole RT. It is challenging to evaluate reaching objectives, since there are many steps involved in reaching a specific objective. Often the tasks or sub-objectives are divided between sub-teams (network, web and client-side) and between individuals in those sub-teams. The difficulty of a specific exploitation depends on the individual's skillset, which varies widely. Hence, there is a trade-off between assigning a task to an experienced member to increase the chance of success, versus teaching a new member. For example, an experienced network administrator is more effective in exploiting network protocols and is likely less visible while doing so, but may not learn anything new.

Discussions and feedback revealed that several respondents felt they were stuck and working alone. Division of the tasks between sub-teams and individuals also diminishes the learning potential. One training design option to alleviate this issue

would be regular *team timeouts* for reflection. Reflective team sharing is crucial for the learning success of each individual, and would overcome the project management approach where each team member focuses only on personal objectives. Higher emphasis should be on offering tips and helping those stuck on an objective to move forward whilst also keeping track of the feedback provided by Frankenstack. The coaching could also be handled in the form of a *buddy system* where RT members are not assigned a sub-task individually, but in groups of two or three. They would then have to share their knowledge and can benefit from different individual backgrounds.

Finally, it is important to have better time-planning during the execution. While it is certainly appropriate to allow for flexibility in the paths that the RT can take to solve the objectives, participants should avoid spending too much time on wrong targets. Nevertheless, the learning impact of the exercise in this format (i.e., with real-time feedback) is very positive. Only 13% of all participants' responses reported no significant change in their skills, while an overwhelming 87% perceived an improvement in their skill level, and 93% agreed that they were satisfied with exercise.

V. Future work

We encountered several unforeseen problems, as methods for assessing technical RT campaigns have to be incorporated into the game scenario itself. However, most XS17 targets had already been developed before the initial stages of this research. We plan to increase information sharing between Red and Yellow teams to improve RT progress measurement. Thus, we can develop better assessment methodologies for RT skill levels and YT feedback framework.

Development of a new dynamic version of EVE is already underway for the next XS iteration. In addition to the network map view, it can draw the network map dynamically as RT compromises new targets. Currently, EVE can only be used after the end of the exercise. However, in addition to providing more actionable alerting, the new version can also reduce RT work for mapping new systems and allow them to focus on the technical exercise.

VI. Conclusion

In this paper, we have presented the core challenges in organizing a CDX with Red Team emphasis, such as timeliness and accuracy of feedback, and ensuring participant education without compromising the game scenario. We compiled a novel stack of open-source tools to provide real-time feedback and situational awareness, and conducted surveys among the RT members to assess the effectiveness of this method.

Frankenstack feedback regarding learning impact was mainly positive. However, there are critical questions to answer when designing the RT exercises, such as what is the right balance of information to provide to the RT, does the behavior change due to monitoring or information visible (i.e., learners unconsciously limit themselves by not trying out more risky strategies, etc.). Also, some further learning design changes, and not necessarily only limited to SA, can maximize the return on the significant investment into preparing such RT exercises. We hope to spark a discussion on improving these problems.

VII. Acknowledgments

The authors would like to thank Mr. Risto Vaarandi, Mr. Hillar Aarelaid and Prof. Olaf M. Maennel for their valuable contributions. This work has been supported by the Estonian IT Academy (StudyITin.ee).

References

[1] T. Minárik, "NATO Recognises Cyberspace as a Domain of Operations at Warsaw Summit," Available: https://ccdcoe.org/nato-recognises-cyberspace-domain-operations-warsaw-summit.html.

[2] P. Brangetto *et al.*, "Cyber Red Teaming - Organisational, technical and legal implications in a military context," NATO CCD CoE, Tech. Rep., 2015.

[3] "Crossed swords exercise," Available: https://ccdcoe.org/crossed-swords-exercise.html.

[4] P. Brangetto *et al.*, "From Active Cyber Defence to Responsive Cyber Defence: A Way for States to Defend Themselves Legal Implications," Available: https://ccdcoe.org/multimedia/active-cyber-defence-responsive-cyber-defence-way-states-defend-themselves-legal.html.

[5] B. E. Mullins *et al.*, "The impact of the nsa cyber defense exercise on the curriculum at the air force institute of technology," in *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, Jan 2007, pp. 271b–271b.

[6] A. T. Sherman *et al.*, "Developing and delivering hands-on information assurance exercises: experiences with the cyber defense lab at umbc," in *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.*, June 2004, pp. 242–249.

[7] R. C. Dodge *et al.*, "Organization and training of a cyber security team," in *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, vol. 5, Oct 2003, pp. 4311–4316.

[8] G. H. Gunsch *et al.*, "Integrating cdx into the graduate program," in *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, vol. 5, Oct 2003, pp. 4306–4310.

[9] R. C. Dodge and T. Wilson, "Network traffic analysis from the cyber defense exercise," in *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, vol. 5, Oct 2003, pp. 4317–4321.

[10] H. Holm *et al.*, "Empirical analysis of system-level vulnerability metrics through actual attacks," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 6, pp. 825–837, Nov 2012.

[11] J. Brynielsson *et al.*, "Using cyber defense exercises to obtain additional data for attacker profiling," in *2016 IEEE Conference on Intelligence and Security Informatics (ISI)*, Sept 2016, pp. 37–42.

[12] D. Arendt *et al.*, "Cyberpetri at cdx 2016: Real-time network situation awareness," in *2016 IEEE Symposium on Visualization for Cyber Security (VizSec)*, Oct 2016, pp. 1–4.

[13] D. L. Arendt *et al.*, "Ocelot: user-centered design of a decision support visualization for network quarantine," in *2015 IEEE Symposium on Visualization for Cyber Security (VizSec)*, Oct 2015, pp. 1–8.

[14] D. S. Henshel *et al.*, "Predicting proficiency in cyber defense team exercises," in *MILCOM 2016 - 2016 IEEE Military Communications Conference*, Nov 2016, pp. 776–781.

[15] "Elastic stack," Available: https://www.elastic.co/.

[16] R. Vaarandi *et al.*, "Simple event correlator - best practices for creating scalable configurations," in *Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2015 IEEE International Inter-Disciplinary Conference on*, March 2015, pp. 96–100.

[17] R. Vaarandi, "Platform independent event correlation tool for network management," in *Network Operations and Management Symposium, 2002. NOMS 2002. 2002 IEEE/IFIP*, 2002, pp. 907–909.

[18] ——, "Sec - a lightweight event correlation tool," in *IP Operations and Management, 2002 IEEE Workshop on*, 2002, pp. 111–115.

[19] "Frankensec," Available: https://github.com/ccdcoe/frankenSEC.

[20] "Snoopy Logger," Available: https://github.com/a2o/snoopy.

[21] "Otta," Available: https://github.com/ccdcoe/otta.

[22] "Eve - event visualization environment," Available: https://github.com/ccdcoe/EVE.

[23] N. Satterly, "alerta," Available: http://alerta.io/.

Appendix 6

Publication VI
M. Pihelgas. Design and Implementation of an Availability Scoring System for Cyber Defence Exercises. In *14th International Conference on Cyber Warfare and Security (ICCWS 2019)*, page 329–337, 2019

The paper is included in the *Proceedings of the 14th International Conference on Cyber Warfare and Security (ICCWS 2019)*. ISBN: 978-1-912764-11-2; ISSN: 2048-9870.

Design and Implementation of an Availability Scoring System for Cyber Defence Exercises

Mauno Pihelgas
NATO Cooperative Cyber Defence Centre of Excellence, Tallinn, Estonia
Tallinn University of Technology, Tallinn, Estonia
firstname.lastname@ccdcoe.org

Abstract: Cyber defence exercises are crucial for training readiness and awareness within the *cyber domain*. This new domain is acknowledged by NATO, alongside land, sea, air and space. Alliance nations are endorsing the development of both defensive and responsive cyber capabilities. This paper discusses designing and building a reliable availability scoring system for a large international cyber defence exercise, Locked Shields. The system provides essential input for scoring the exercise participants to spark some friendly competition and motivate players. The previous solution was replaced by a modular setup that is built around a well-known open-source IT monitoring software called Nagios Core. Before embarking to develop a new system, we studied available research and looked at various other CDXs for similar implementations. Unfortunately, we did not find such full-blown scoring systems in use at the time. At least not according to the information that was provided to us. We therefore relied on best practices and prior experience to develop an automated availability scoring system. The paper provides some background information on the exercise, describes the requirements, design process and implementation of the scoring solution. The current system has been under continuous improvement since 2014 and has successfully provided the automated scoring checks for the past five exercises. In addition to success stories, several issues and problem workarounds are addressed. As such, this paper serves as a valuable resource for cyber defence exercise managers and practitioners looking to implement similar scoring solutions.

Keywords: availability, cyber exercise, monitoring, Nagios, scoring, Selenium

1. Introduction

The field of cyber defence has become increasingly important over the years and numerous cyber defence exercises (CDX) are being organised all over the world. One of the largest is Locked Shields (LS), which is a game-based real-time network defence exercise that has been organised by the NATO Cooperative Cyber Defence Centre of Excellence since 2010 (NATO Cooperative Cyber Defence Centre of Excellence 2018). Measuring and scoring the performance of the training audience (Blue Teams) is essential to ensure that everyone gives their best during the competitive exercise.

The primary focus of this paper is on the process of measuring the availability of individual services provided by Blue Team (BT) systems and passing the data to the overall scoreboard (see Figure 1). This *scoreboard* is another system, which combines all the different sub-scores into a combined score for a BT; however, this does not fall within the scope of this paper and is not to be confused with the availability scoring system. Although this paper focuses only on the LS implementation, the same scoring system could be easily implemented for other similar CDXs.

1.1 Exercise overview

The LS exercise spans two days and is built as a competitive game featuring a fictional scenario in which the defending BTs are scored on their performance in several different interdisciplinary categories, such as defending against technical cyber attacks, incident reporting, situation reporting, responding to scenario injects and keeping their systems available to the users. Scoring is an integral part of the game, because participants need to know how well they performed in the tasks set for them and compared to other teams. The final scoreboard ranking decides which team wins the annual LS exercise.

Figure 1: Example scoreboard combining all different score types

Participants represent one of five different roles: Blue Team (defence), Red Team (*bad guys*), Yellow Team (situation awareness), Green Team (exercise infrastructure), and White Team (exercise management, strategic gameplay, media, etc.). According to the scenario, the BTs assume the role of rapid-reaction teams assisting the fictional country of Berylia, which is in conflict with another fictional country, Crimsonia, represented by the Red Team (RT). The exercise takes place over just two days (8 hours per day) of intense game-play.

In 2018, each of the 22 BTs was responsible for maintaining the continuous and secure operation of 140 hosts with several monitored services on each host, amounting to 1,425 monitored services per team to calculate the availability of each required service. The notion of availability is represented as an uptime value between 0 and 1 for a particular service. This value could be represented as a Service Level Agreement (SLA) percentage for easier interpretation. The scoring system keeps track of all check results and calculates the uptime for each service.

The availability score accounts for approximately ⅓ of the total positive score points in the exercise, which is roughly the same as the amount of negative score that teams may get for successful attacks carried out by the RT. Therefore, teams need to balance their strategy between security and availability of services wisely, because the function for calculation of points lost due to downtime is exponential, not linear (discussed in section 3).

The gamenet infrastructure features a wide variety of traditional IT infrastructure and special-purpose industrial systems: Windows, Linux, and FreeBSD hosts, Siemens Programmable Logic Controllers (PLCs), Threod Flight Controllers and Ericsson Virtual EPC Packet Gateways for 4G/LTE connectivity. In addition to system administration and hardening tasks, teams are also faced with forensic and legal challenges and various injects from the game's media team. This means that the defending BTs must include specialists with very different skills to be able to field all the required expertise.

1.2 **Problem and motivation**

The development of this availability scoring system was motivated by several deciding factors. At the beginning of 2014 the author of the previous system left the LS organising team. The old system turned out to be a monolithic Perl script that contained all the information on what services to check and the logic for every single availability check. We have nothing against implementing solutions in Perl language, but the script was lacking some of the widely accepted best practices in software engineering (e.g., it is recommended to have a modular

design where smaller and simpler interconnected subsystems communicate) (Seacord 2018). Thus, the old system was difficult to understand and develop.

Primary concerns were the stability and scalability of the system. There were issues with high memory consumption with all the various libraries required for checking the BT services constantly loaded in the process. We decided to replace the previous system with a new and improved solution that, among other aspects, would provide a modular design and increased stability.

1.3 **Outline**

The remainder of this paper is organised as follows, section 2 describes the requirements of the new system, section 3 introduces the basic functionality the monitoring solution, section 4 presents the current implementation of the availability scoring, section 5 provides a brief insight into future work, section 6 gives an overview of related work, and section 7 concludes the paper.

2. Requirements

When we initially set out to design the new scoring system, we had several requirements due to the nature of the LS exercise and several of our own ideas and best practices on what is required of an exercise scoring system. The following list briefly describes the requirements and the reasoning behind the specific item.

- Stable and reliable: remain operational even in situations where the underlying network or related systems may temporarily fail.
- Active checks: the queries towards BT systems have to be initiated by the scoring server to avoid any manipulation of passive check results.
- Predictable: each BT should receive the same number of requests (checks) from the scoring engine.
- Modular: easy to add new functionality or modify existing modules, also availability of existing modules in the community.
- Customisable: since the CDX is not a regular IT environment, we do not need all the functionality offered by many monitoring tools (e.g., problem escalation, help-desk functionality, etc.).
- Integration: ability to easily integrate the solution with other pre-existing systems.
- Scalable: ability to monitor thousands of hosts and services with a short check interval.
- Well-known and documented: avoid relying solely on a single person to configure the system.
- Evasive tactics: implement various tactics to avoid being easily identified as the scoring engine and thus allowing the system to be whitelisted by the BTs.
  - Randomise IP addresses: periodically change IP addresses on selected network interfaces.
  - Randomise check intervals: periodically change check intervals to avoid creating a recognisable pattern in log files.
  - Mimic regular systems: remove any identifiers of the monitoring software.
- Preferably open-source software: we might need to modify the source code of various components (especially to implement the evasive tactics mentioned above).

It was evident that most of the first level requirements listed above could be addressed by using some well-known monitoring solutions that are already available. We tested several different solutions, such as Zabbix, Shinken, Opsview, Nagios XI, Nagios Core, op5 and Centreon. Most of the solutions offered similar functionality and were able to satisfy many or even most of the requirements but were often too overloaded with complex functionality (e.g., helpdesk and incident handling functionality) which we did not need in our situation. After testing several strong contenders in our lab environment, we decided to use Nagios Core as a central element of the new availability scoring system for LS.

However, there was a narrower set of requirements regarding various evasive tactics that none of the compared solutions was able to solve *out of the box*. Typical monitoring solutions are not evasive and work in a very recognisable and predictable manner. For instance, checks usually come from a single known IP address that is always granted access through the firewall, the time between the checks (check intervals) is usually fixed at regular intervals (e.g., 1 minute or 5 minutes), and by default the system would identify itself (e.g., in the web browser user-agent string) as the monitoring system. To avoid this issue, additional functionality had to be developed separately.

3. Basics of availability scoring

Typical monitoring systems actively check and report the state – (*OK*, *Warning*, *Critical*, or *Unknown*) – of the monitored services. There is a predefined list of services BTs are supposed to keep up and running for the entire duration of the two-day exercise. Any disruptions in the services will result in lost uptime and valuable points.

Active checks are implemented by executing various user-defined monitoring scripts that perform the necessary steps. For example, checking the availability of a web page requires the script to establish a connection to the web server hosting the site and download the content of the page for any further inspection (e.g., verifying the presence of correct content). Active checks are initiated by the scoring server and BTs have no control over when the check is executed or what is being checked within the logic of the script (see Figure 2).

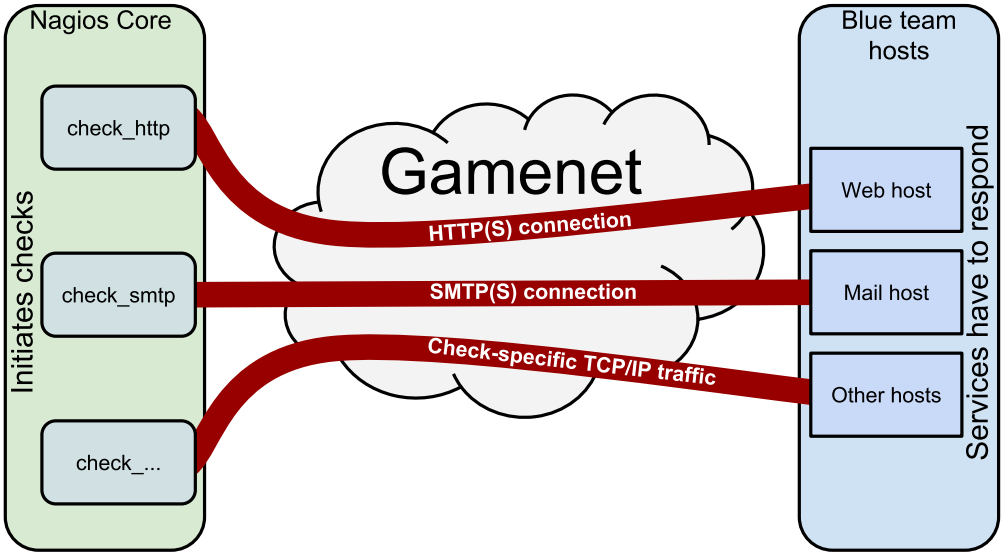

Figure 2: Basic active check process

Alternatively, passive service checks could be used in monitored systems to report the current state of the system back to the central monitoring server. For instance, in our previous example the web server would periodically execute a self-check and send the result to the scoring server. Although this is the preferred method in many standard monitoring solutions, it does not fit well with our exercise scenario, because it is in the interest of the competing BTs to portray as good an uptime as possible. It can be tempting for teams to stop sending genuine check results and send forged passive check results to our scoring server instead. We therefore prefer active checks, because they are more reliable in the adverse situations of the competitive exercise.

Listing 1 illustrates the output produced by these service checks. The output format was largely taken from the previous scoring system and it fits the purpose nicely since the vertical bar symbol is a special character in Nagios to distinguish the message from check performance data and thus cannot accidentally appear in our output.

Listing 1: Example output from active service checks

```
mail.blue05.ex|http|OK|HTTP OK: HTTP/1.1 200 OK - 322 bytes in 0.005 second response time
|1524721248|1524721248

hmi.pgc.blue13.ex|http.ipv6|OK|HTTP OK: HTTP/1.1 200 OK - 954 bytes in 0.014 second response
time|1524721248|1524721249

ws4-02.int.blue15.ex|ssh|CRITICAL|No route to host|1524721244|1524721249
```

The BTs can gather up to 20,000 points (about ⅓ of total positive score) for the availability of services. Individual services are distributed into various sub-groups and are assigned an arbitrary weight value (W) based on their significance. Any loss of uptime (U) will affect the score coefficient (SC) exponentially:

$$SC=W^{U-1},\ 0\leq U\leq 1$$

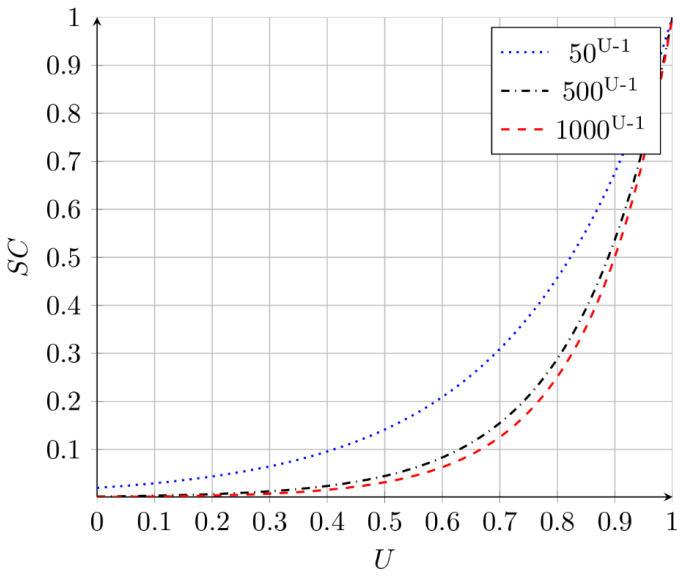

Figure 3: Score coefficient (SC) with regards to uptime (U) and three example service weight values (W)

The rationale for this harsh score penalty is that in the IT field, production system uptime is commonly expected to be in the order of 99.9% or in some cases even higher, which, for example, means that a service can be down for just about 10 minutes per week or 1.44 minutes per day. Evidently, the higher the service weight value, the steeper the score penalty for lost uptime (see Figure 3). We calculate the resulting score points according to W×SC. For instance, a service with a relatively high weight value (W) of 1,000 points will only yield 500 score points if its uptime is 90%. This is because its score coefficient (SC) quickly decreases to approximately 0.5 if the uptime (U) drops to 0.9.

Throughout the game execution, runtime scores are added up and sent to the live scoreboard. An example scoreboard can be seen in Figure 1. The availability score discussed in this section is represented as *sla score* in the figure.

4. Implementation

This section describes the hardware and software specification that the solution was deployed on in 2018, followed by a description of how the evasive techniques mentioned in section 2 have been implemented.

4.1 System description

There were five virtual machines (VM) assigned for the availability scoring system. Each VM was assigned 12 logical processors, 8GB of memory, and 60GB of disk space. We could have managed with fewer VMs, but the main requirement was to attain different vantage points (e.g., BT internal or external) in the gamenet infrastructure. For instance, depending on the type of BT service (e.g., public or private), the availability checks must be performed from a BT internal network, from an external network, or in some cases from both.

Thus, on each scoring machine, two network interfaces provided Green Team management and generic internet access, while the third network interface provided the unique network vantage point: 4G/LTE network, RT network, BT internal networks, or BT management networks.

To monitor all the required internal BT assets, we had to connect the scoring server to five different /24 subnets for every BT. This added up to 132 virtual network interfaces connected on one of the scoring servers, which is quite uncommon and created some issues that had to be overcome in the operating system configuration.

4.2 **Nagios**

In 2018, the scoring system was built on a Debian GNU/Linux 9.4 operating system using Nagios Core v4.3.4 (latest stable version at that time) which we compiled from source code. We also modified and compiled the standard nagios-plugins package (version 2.2.1) containing the check scripts provided by Nagios Developers (section 4.4 discusses the motivation behind these modifications).

We applied several of the steps outlined in the Nagios Performance Tuning guide (Nagios Enterprises 2018*a*). We made extensive use of the RAM Disk setup recommended by Nagios Enterprises (2018*b*) to minimise the number of slow disk I/O operations and keep frequently accessed files in memory. In addition to Nagios-specific files, we also used it to store other temporary files required by scripts and plugins.

4.3 **Performance assessment**

The virtualised systems worked well. System load was mostly below the critical threshold (i.e., equal to the number of CPU threads), and although there were critical peaks when restarting Nagios, these passed momentarily. In terms of memory use, the 8GB assigned to the hosts was plenty; on average the systems used about 3-4GB.

For each BT, we performed scoring checks for 140 hosts providing a total of 1,425 services on both IPv4 and IPv6. Multiply that by the number participating teams (22), and we get an impressive 3,080 hosts with 31,350 services being checked at least once per minute. During the 16 hours of game-play, 34,025,305 individual scoring checks were performed and logged. This averages at about 35,443 checks per minute.

4.4 **Evasive techniques**

The requirements introduced several situations where we needed to avoid the availability scoring system being identified by BTs. They could still attempt to identify the scoring server by making an educated guess based on the requests made by the scripts checking the services, but we tried to make it more difficult for them. To clarify why this is important, consider a situation in which the BTs were able to successfully identify the IP addresses from which the scoring server is accessing their services. They could use a whitelisting approach in the firewall and always allow access for the scoring server, but block everyone else, including the RT, from accessing the service. The scoring server would see that everything is up and functional, while no other users would be able to use the service. We describe some of the more important evasive techniques below.

4.4.1 *Randomising IP addresses*

The primary way to evade identification was to regularly change the IP addresses of the scoring server. We configured the server to change all its public IP addresses that were used to communicate with BT services from the RT networks. The change was scheduled at a random interval ranging between 5 to 10 minutes. To safeguard against IP conflicts, we used the arping utility for IPv4 and ndisc6 utility for IPv6 to verify that an IP address was available before assigning it to the interface.

Since changing an IP address could disrupt an ongoing service check, we needed to shut down Nagios before replacing any addresses. Typically, the entire process took about 7-8 seconds. This was a critical moment, because the system can end up in a state where it no longer has connectivity to the gamenet. To make sure that the new configuration was fully correct and operational, we introduced a series of connectivity tests. If the tests fail, the system reverts to its default static IP addresses, which should always work.

4.4.2 *Randomising check intervals*

For normal monitoring applications, a fixed checking interval is a sensible idea. However, in our case the BTs could predict, with an accuracy of one second, when the scoring system would perform the next availability check. This look like a heartbeat on log monitoring tools and dashboards. Inspect the timestamps in Listing 2 to see how easy it is to detect a system with a fixed check interval (e.g., 60 seconds). To counter this, we developed a simple script that would randomise the check interval within a given range between 40 and 55 seconds. Since any changes in check interval would only take effect after restarting the Nagios software, we combined this procedure with changing IP addresses. Observe the access timestamps in Listing 3 after applying the randomised check interval.

Listing 2: Typical unmodified Nagios checks examined from a BT web server access log

```
10.0.228.111 - - [23/April/2018:06:39:58 +0000] "GET / HTTP/1.1" 200 437 "-"
"check_http/v2.2.1 (nagios-plugins 2.2.1)"

10.0.228.111 - - [23/April/2018:06:40:58 +0000] "GET / HTTP/1.1" 200 437 "-"
"check_http/v2.2.1 (nagios-plugins 2.2.1)"

10.0.228.111 - - [23/April/2018:06:41:58 +0000] "GET / HTTP/1.1" 200 437 "-"
"check_http/v2.2.1 (nagios-plugins 2.2.1)"
```

Listing 3: Example of a web server log entry created by a modified Nagios check_http script

```
10.0.228.111 - - [23/April/2018:06:45:43 +0000] "GET / HTTP/1.1" 200 437 "-" "Mozilla/5.0
(X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0"

10.0.228.111 - - [23/April/2018:06:46:25 +0000] "GET / HTTP/1.1" 200 437 "-"
"Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0"
```

4.4.3 *Identification strings*

We discovered that default Nagios plugins such as *check_http, check_ssh, check_smtp* clearly identify themselves to the service they are communicating with. Log lines in Listing 2 reveal that the check_http script sends the BT web server the default user-agent string. Under normal circumstances this is good behaviour and helps anyone looking into the logs to quickly distinguish events created by the monitoring system. In our case, however, this would easily allow the BTs to identify and whitelist the scoring server's IP address. To address this, we modified several monitoring scripts to replace these identifiable strings with software versions that would exist in a regular Linux desktop computer. See Listing 3 for a log entry created by a modified version of a monitoring script using the Mozilla Firefox user-agent string.

4.4.4 *Countering BT tactics*

Over the years, we have seen BTs find ways to improve their score by trying to trick the scoring system into thinking their services are up and running when in fact they have just set up a dummy service or a similar-looking static website.

For instance, some teams have set up tools such as netcat or Portspoof that bring up fake services on ports and respond to any incoming requests. Therefore, it is essential that monitoring plugins perform more in-depth functionality checks to verify that there is in fact a correct service responding on the expected port.

Another questionable tactic has been to take an entire web site and convert it into static HTML pages. Visually the site seems normal, but any dynamic content and integration with a database has been eliminated. We have used the open-source Selenium WebDriver, a testing framework for web applications to counter this. Using Selenium allows us to write our test cases in Python for automating web browsing. For example, mimicking a customer visiting the page of an online shop: logging in, browsing a few items and adding some of them to the shopping cart, then finalising the purchase by checking out and verifying that the purchase was indeed saved under previous orders. Such test cases provide an excellent way of checking that essential functionality is present, but due to their complexity they are difficult to debug if the test case breaks and erroneously flags the site as broken.

5. Future work

Since we have been collecting exercise data such as scoreboard results, network PCAPs, scoring logs and RT activity logs for the past five years, we have gathered a substantial dataset for future research.

Our first idea is to analyse the success rate of different strategies used by BTs. For example, some teams focus more on providing excellent uptime at the expense of security, while others completely lock down their systems to protect them against the RT, but likely block the scoring system in the process. Having the training dataset, we could develop a model that would predict the score of the team based on their strategy. This could also reveal combinations of strategies that are more likely to succeed than others.

Additionally, we aim to develop a system that would better detect cheating among the participants. For instance, we should verify that the availability scoring results correspond to the RT reports. There should be a strong

correlation between the BT systems going offline after successful attacks and RT attacks failing when the BT system was already unavailable before the attack. Finding anomalies (e.g., change of expected values or hashes) in the responses from the BT systems would allow us to discover and sanction (negatively score) the excessive use of questionable tactics (see subsection 4.4.4 above) during the game. Currently all this requires manual analysis.

6. Related work

LS is by no means the only cyber defence exercise in the world. There are several different cyber exercises conducted around the globe. The *NSA Cyber Exercise* (2018) (formerly Annual Cyber Defense Exercise) hosted by the National Security Agency and *Cyber Shield 2018* (2018) hosted by the U.S. Army National Guard are probably the most similar to Locked Shields.

Other relevant exercises include *Cyber Security Awareness Week (CSAW)* (2018) challenges, *Cyber Security Challenge UK* (2018), *DEF CON Capture the Flag Contest* (2018), *Mid-Atlantic Collegiate Cyber Defense Competition (MACCDC)* (2018), *National Collegiate Cyber Defense Competition (NCCDC)* (2018), *National Cyber League (NCL)* (2018) competition, and *International Capture The Flag (iCTF)* (2018) exercise. One of the most recent additions to this list are *off-the-shelf* technical exercises provided by CybExer Technologies (2018).

With such a variety of annual events, one should also distinguish different types of cyber exercises: security challenges, capture-the-flag contests, or exercises that focus on defence. Often, there is no clear classification, but it is still worthwhile to consider what is the primary goal of the exercise. In the case of LS, training to defend various systems and networks is the main objective and BTs are the training audience.

Unfortunately, there is a limited amount of material available on the subject. Most of the sources listed above are just limited to a homepage briefly describing the past exercises from a relatively high level. This is especially the case with military exercises. The remainder of this section discusses several more elaborate sources about previous work on cyber exercises, especially the ones that mention various scoring or assessment systems.

A paper by Patriciu & Furtuna (2009) presents practical guidelines to follow when designing a new cyber security exercise. The paper aims to assist CDX newcomers who oversee designing a cyber security exercise. The step-by-step instructions cover topics such as defining the objectives, designing the network topology, creating a scenario, establishing rules, choosing the appropriate metrics, and conducting a lessons-learned procedure. The guideline has a rather brief section on recommendations for the scoring engine, stating only that there must be a clear set of rules to express the way points can be obtained or lost. Moreover, the whole scoring process has to be transparent to all participants.

Papers from Hammervik et al. (2010) and Granåsen et al. (2011) try to capture and analyse the data from the Baltic Cyber Shield exercise, which is in fact a direct precursor of LS. They address whether the vulnerability of a host influences the time required to compromise it, and whether cyber security professionals can predict the success rates of arbitrary code execution attacks.

The work by Werther et al. (2011) shares the experience from conducting capture-the-flag exercises in the MIT Lincoln Laboratory. Regarding scoring, the solution is quite like ours in LS. The scores are calculated as a weighted average of availability, confidentiality, integrity, and offence. The exercise has a scoreboard where the overall score of all participating teams is displayed. Like LS, the first 30 minutes of the exercise is not scored to allow teams to apply patches and secure their machines. However, unlike LS, attacks are allowed during this period.

A paper by Henshel et al. (2016) describes using the Cyber Shield 2015 exercise to develop the assessment model and integrated evaluation of team proficiency metrics in CDXs. In addition to using exercise data, they also conducted an expertise survey before the event to determine potential relationships between prior expertise and performance in the exercise. For future work, they stress that near real-time analysis of the exercise data is required. They conclude that raw data collection is not an issue, but the capability to manually analyse the information does not scale with the huge amounts of incoming data. Their aims and observations closely coincide with ours, thus we have already contacted the authors of this paper and will engage in future cooperation.

7. Conclusion

There is a growing trend of organising cyber defence exercises, which is not an easy task. In addition to difficulties in finding people who can develop the scenario and build up the required infrastructure to provide an exercise

on this scale, motivating the participants to give of their best is a challenge. We have found that adding a competition to the exercise is beneficial and serves this purpose well. The competition can be scored based on the performance of trainees by checking how well they keep their systems up and functional.

This paper has presented an overview of the availability scoring system for the Locked Shields cyber defence exercise. It has discussed the design and practical implementation of the system and listed some key observations from the experience gained during the past five years of development.

8. Acknowledgements

This work has been supported by the Estonian IT Academy (StudyITin.ee).

References

Cyber Security Awareness Week (CSAW) (2018), Available: https://csaw.isis.poly.edu/.

Cyber Security Challenge UK (2018), Available: http://cybersecuritychallenge.org.uk/.

Cyber Shield 2018 (2018), Available: http://www.cs18.org/.

CybExer Technologies (2018), 'Technical Exercises', Available: https://cybexer.com/.

DEF CON Capture the Flag Contest (2018), Available: https://www.defcon.org/html/links/dc-ctf.html.

Granåsen, D., Granåsen, M., Sundmark, T., Holm, H. & Hallberg, J. (2011), Analysis of a Cyber Defense Exercise using Exploratory Sequential Data Analysis, The 16th International Command and Control Research and Technology Symposium (ICCRTS).

Hammervik, M., Granåsen, D. & Hallberg, J. (2010), Capturing a Cyber Defence Exercise, The 1st National Symposium on Technology and Methodology for Security and Crisis Management.

Henshel, D. S., Deckard, G. M., Lufkin, B., Buchler, N., Hoffman, B., Rajivan, P. & Collman, S. (2016), Predicting proficiency in cyber defense team exercises, *in* 'MILCOM 2016 - 2016 IEEE Military Communications Conference', pp. 776–781.

International Capture The Flag (iCTF) (2018), Available: http://ictf.cs.ucsb.edu/.

Mid-Atlantic Collegiate Cyber Defense Competition (MACCDC) (2018), Available: http://maccdc.org/.

Nagios Enterprises (2018*a*), 'Tuning Nagios For Maximum Performance', Available: https://assets. nagios.com/downloads/nagioscore/docs/nagioscore/4/en/tuning.html.

Nagios Enterprises (2018*b*), 'Utilizing a RAM Disk in Nagios XI', Available: https://assets.nagios.com/ downloads/nagiosxi/docs/Utilizing A RAM Disk In NagiosXI.pdf.

National Collegiate Cyber Defense Competition (NCCDC) (2018), Available: http://www.nationalccdc. org/.

National Cyber League (NCL) (2018), Available: https://www.nationalcyberleague.org/.

NATO Cooperative Cyber Defence Centre of Excellence (2018), 'Locked Shields 2018', Available: https: //www.youtube.com/watch?v=meC8O9Mptz4.

NSA Cyber Exercise (2018), Available: https://www.nsa.gov/what-we-do/cybersecurity/ncx/.

Patriciu, V.-V. & Furtuna, A. C. (2009), Guide for Designing Cyber Security Exercises, *in* 'Proceedings of the 8th WSEAS International Conference on E-Activities and Information Security and Privacy', E-ACTIVITIES'09/ISP'09, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, pp. 172–177.

Seacord, R. (2018), 'Top 10 Secure Coding Practices', Available: https://wiki.sei.cmu.edu/confluence/ display/seccode/Top+10+Secure+Coding+Practices.

Werther, J., Zhivich, M., Leek, T. & Zeldovich, N. (2011), Experiences in Cyber Security Education: The MIT Lincoln Laboratory Capture-the-flag Exercise, *in* 'Proceedings of the 4th Conference on Cyber Security Experimentation and Test', CSET'11, USENIX Association, Berkeley, CA, USA, pp. 12–12.

Appendix 7

Publication VII
P. Théron, A. Kott, M. Drašar, K. Rzadca, B. LeBlanc, M. Pihelgas, L. Mancini, and A. Panico. Towards an active, autonomous and intelligent cyber defense of military systems: The NATO AICA reference architecture. In *2018 International Conference on Military Communications and Information Systems (ICMCIS)*, pages 1–9, May 2018

The paper is included in the *Proceedings of the 2018 International Conference on Military Communications and Information Systems (ICMCIS 2018)*.
DOI: 10.1109/ICMCIS.2018.8398730

Towards an Active, Autonomous and Intelligent Cyber Defense of Military Systems: the NATO AICA Reference Architecture

Paul Theron
Thales
Salon de Provence, France
paul.theron@thalesgroup.com

Alexander Kott
U.S. Army Research Laboratory
Adelphi, MD, USA
alexander.kott1.civ@mail.mil

Martin Drašar
Masaryk University
Brno, Czech Republic
drasar@ics.muni.cz

Krzysztof Rzadca
University of Warsaw
Warsaw, Poland
krzadca@mimuw.edu.pl

Benoît LeBlanc
Ecole Nationale Supérieure de Cognitique
Bordeaux, France
benoit.leblanc@ensc.fr

Mauno Pihelgas
NATO CCDCOE
Tallinn, Estonia
mauno.pihelgas@ccdcoe.org

Luigi Mancini
Sapienza University
Rome, Italy
mancini@di.uniroma1.it

Agostino Panico
Sapienza University
Rome, Italy
panico@di.uniroma1.it

***Abstract*—Within the future Global Information Grid, complex massively interconnected systems, isolated defense vehicles, sensors and effectors, and infrastructures and systems demanding extremely low failure rates, to which human security operators cannot have an easy access and cannot deliver fast enough reactions to cyber-attacks, need an active, autonomous and intelligent cyber defense. Multi Agent Systems for Cyber Defense may provide an answer to this requirement. This paper presents the concept and architecture of an Autonomous Intelligent Cyber defense Agent (AICA). First, we describe the rationale of the AICA concept. Secondly, we explain the methodology and purpose that drive the definition of the AICA Reference Architecture (AICARA) by NATO's IST-152 Research and Technology Group. Thirdly, we review some of the main features and challenges of Multi Autonomous Intelligent Cyber defense Agent (MAICA). Fourthly, we depict the initially assumed AICA Reference Architecture. Then we present one of our preliminary research issues, assumptions and ideas. Finally, we present the future lines of research that will help develop and test the AICA / MAICA concept.**

Keywords—intelligent agent, autonomy, cyber warfare, cyber security

I. RATIONALE FOR THE AICA/MAICA CONCEPT

Today, five broad types of systems coexist in Land, Sea and Air operations:

- Office and information management systems, which includes web services, emailing systems, and information management applications ranging from human resource management to logistics through maintenance and project management;
- C4ISR systems for the command of war operations, with associated Battlefield Management Systems that extend the C4ISR down to single vehicles and platoons;
- Communication systems such as SATCOM, L16, line of sight networks, software defined radios, etc.;
- Platform and life automation systems, similar to industrial systems and that provide sea vessels for instance with air conditioning, refrigeration, lifts, video surveillance, etc.;
- Weapon systems, which include both sensors and effectors of all kinds, including the Internet of Battle Things (IoBT).

On the battlefield, the future Global Information Grid will mix those technologies into complex large scale networks of massively interconnected systems, the cybersecurity supervision of which by human operators will become increasingly difficult, if not impossible.

Besides, a great number of military missions will require defense vehicles and effectors to work stealthily while some will find themselves isolated because of poor bandwidth or because communications will become untrustworthy. Isolated systems will create a specific class of problems in terms of the possibility to monitor and manage their cybersecurity. On one hand, to fully analyze their cyber-health would be possible only when connected at base during maintenance and operation preparation. On the other hand, in case of cyber-attacks, they will require immediate counter-reactions while no cybersecurity or cyber defense specialist is available.

This paper is based on NATO IST Panel activity IST-152-RTG, "Intelligent, Autonomous and Trusted Agents for Cyber Defense and Resilience."

978-1-5386-4559-8/18/$31.00 C2018 IEEE

Finally, defense infrastructures and systems engaged in battle operations must show extremely low failure rates. Counter reactions to cyber-attacks must therefore be initiated at the speed of operation of these systems, not at the (low) speed of human decision making in the presence of complex issues. In a conflict with a technically sophisticated adversary, military tactical networks will be a heavily contested battlefield. Enemy software cyber agents -- malware -- will infiltrate friendly networks and attack friendly C4ISR and computerized weapon systems.

In this context, systems' cyber defense will be organized in two manners:

- Connected systems of lesser criticality will be monitored by cybersecurity sensors, security information and event management (SIEM) systems, and security operations centers (SOCs). This will be the case of office and information management systems and of C4ISR systems under peaceful circumstances.
- Higher grade systems or configurations such as C4ISR systems deployed in combat circumstances, communication systems, life and automation systems and weapon systems require autonomous intelligent cyber defense capabilities.

To fight cyber-attacks that may target this last class of military systems, we expect that NATO needs artificial cyber hunters - intelligent, autonomous, mobile agents specialized in active cyber defense, that we call Autonomous Intelligent Cyber defense Agents (AICA). They will work in cohorts or swarms and will be capable, together, to detect cyber-attacks, devise the appropriate counter measures, and run and adapt tactically their execution.

Those friendly NATO cyber agents -- goodware -- will stealthily patrol networks, detect enemy agents while remaining concealed, and they will devise the appropriate counter-attack plan and then destroy or degrade the enemy malware. They will do so mostly autonomously, because human cyber experts will be always scarce on the battlefield, because human reactions will be too slow, and because connectivity might be nonexistent or poor.

Agents will be learning and adaptive because the enemy malware and attack patterns are constantly evolving. They will be stealthy because the enemy malware will try to find and kill them. They will work in cohorts or swarms as attacks will be sophisticated and stealthy, and only collective intelligence will stand a chance to detect the early combined signs of malware actions and positions. In addition, they will do so because combatting malware will mean fighting a variety of pieces of malware acting either simultaneously or in a sequence hard to detect, and intelligently spread across the friendly military systems and networks they attack to produce the effect sought by the enemy.

Deployed on NATO networks, the AICA friendly software agents will become a major force multiplier. The agents will augment the inevitably limited capabilities of human cyber defenders, and will team with humans when ordered or in need to do so. Without such agents, the effective defense of NATO computer networks and systems might become impossible if attackers also resort on multi agent systems to carry out their attacks. Without active autonomous intelligent cyber defense agents, a NATO C4ISR will not survive an encounter with a determined, technically sophisticated enemy.

At this time, such capabilities remain unavailable for the defensive purposes of NATO. To acquire and successfully deploy such agents, in an inter-operable manner, NATO Nations must create a common technical vision - reference architecture - and a roadmap.

II. Purpose and Methodology of the Project

Inspired by the above rationale, NATO's IST-152 Research and Technology Group (RTG) is an activity that was initiated by the NATO Science and Technology Organization and was kicked-off in September 2016. The group is developing a comprehensive, use case focused technical analysis methodology in order to produce a first-ever reference architecture and technical roadmap for active autonomous intelligent cyber defense agents. In addition, the RTG is working to identify and evaluate selected elements that may be eligible contributors to such capabilities and that begin to appear in academic and industrial research.

Scientists and engineers from several NATO Nations have brought unique expertise to this project. Only by combining multiple areas of distinct expertise along with a realistic and comprehensive approach can such a complex software agent be provided.

The output of the RTG may become a tangible starting point for acquisition activities by NATO Nations. If based on a common reference architecture, software agents developed or purchased by different Nations will be far more likely to be interoperable.

III. Main Features and Challenges of the MAICA Concept

Related research includes Mayhem (from DARPA Cyber Challenge, but also Xandra, etc.), agents from the Pechoucek's group, Professor Mancini's work on the AHEAD architecture [1] and the Aerospace Cyber Resilience research chair's research program [2], Anti-Virus tools (Kaspersky, Bitdefender, Avast, Norton, etc. etc.), HBSS, OSSEC, Various host-based IDS/IPS systems, Application Performance Monitoring Agents, Anti-DDOS systems and Hypervisors. Also, a number of related research directions include topics such as deep learning (especially if it can be computationally inexpensive), Botnet technology (seen as a network of agents), network defense games, flip-it games, the Blockchain, and fragmentation and replication. The introduction of Artificial Intelligence into military systems, such as C4ISR, has been studied, for instance by [3] and [4]. Multi Agent Systems form an important part of AI.

Since the emergence of the concept of Multi Agent Systems (e.g., [5]), MAS have been deployed in a number of contexts such as power engineering [6] and their decentralized automated surveillance [7], industrial systems [8], networked and intelligent embedded systems [9], collective robotics [10],

wireless communication [11], traffic simulation and logistics planning [12], home automation [13].

However, if the use of intelligent agents for the cyber defense of network-centric environments has already long been envisaged [14], effective research in this area is still new.

In the context of the cyber defense of friendly systems, an "agent" has been defined [2] as a piece of software or hardware, an autonomous processing unit:

- With an individual mission and the corresponding competencies, i.e. in analyzing the milieu in which the agent is inserted, detecting attacks, planning the required countermeasures, or steering and adapting tactically the execution of the latter, or providing support to other agents like for instance inter-agent communication;
- With proactivity, i.e. the capacity to engage into actions and campaigns without the need to be triggered by another program or by a human operator;
- With autonomy, i.e. a decision making capacity of its own, the capacity to function or to monitor, control and repair itself on its own, without the need to be controlled by another program or by a human operator, and the capacity to evaluate the quality of its own work and to adjust its algorithms in case of deviance from its norm or when its rewards (satisfaction of its goals) get poor;
- Driven by goals, decision making and other rules, knowledge and functions fit for its purpose and operating circumstances;
- Learning from experience to increase the accuracy of its decisions and the power of its reactions;
- With memories (input, process, output, storage);
- With perception, sensing and action, and actuating interfaces;
- Built around the adequate architecture and appropriate technologies;
- Positioned around or within a friendly system to defend, or patrolling across a network;
- Sociable, i.e. with the capacity to establish contact and to collaborate with other agents, or to enter into a cyber cognitive cooperation when the agent requires human help or to cooperate with a central Cyber C2;
- Trustworthy, i.e. that will not deceive other agents nor human operators;
- Reliable; i.e. that do what they are meant to do, during the time specified and under the conditions and circumstances of their concept of operation;
- Resilient, i.e. both robust to threats (including cyber-threats aimed at disabling or destroying the agent itself; the agent being able to repel or withstand everyday adverse events and to avoid degrading), and resistant to incidents and attacks that may hit and affect the agent when its robustness is insufficient (i.e. the agent is capable of recovering from such incidents or attacks);
- Safe, i.e., conceived to avoid harming the friendly systems the agent defends, for instance by calling upon a human supervisor or central cyber C2 to avoid making wrong decisions or to adjust their operating mode to challenging circumstances, or by relocating when the agent is the target of an attack and if relocation is feasible and allows protecting it, or by activating a fail-safe mode, or by way of self-destruction when no other possibility is available.

In the same context (ibid), a multi agent system is a set of agents:

- Distributed across the parts of the friendly system to defend;
- Organized in a swarm (horizontal coordination) or cohort (vertical coordination);
- In which agents may have homogeneous or heterogeneous roles and features;
- Interoperable and interacting asynchronously in various ways such as indifference, cooperation, competition;
- Pursuing a collective non-trivial cyber defense mission, i.e. allowing to piece together local elements of situation awareness or propositions of decision, or to split a counter-attack plan into local actions to be driven by individual agents;
- Capable of self-organization, i.e. as required by changes in circumstances, whether external (the attack's progress or changes in the friendly system's health or configuration) or internal (changes in the agents' health or status);
- That may display emergent behaviors [15], i.e. performances that are not explicitly expressed in individual agents' goals, missions and rules; in the context of cyber defense, "emergence" is likely to be an interesting feature as, consisting in the "*way to obtain dynamic results, from cooperation, that cannot easily be predicted in a deterministic way*" [15]; it can be disturbing to enemy software in future malware-goodware "tactical" combats within defense and other complex systems;

- Extensible or not, i.e. open or closed to admitting new agents in the swarm or cohort;
- Safe, trustworthy, reliable and resilient as a whole, which is a necessity in the context of cyber defense whereas in other, less challenging contexts may be unnecessary. Resilience, here, may require maintaining a system of virtual roles as described in a human context by [16].

AICA will not be simple agents. Their missions, competencies, functions and technology will be a challenging construction in many ways.

Among many such challenges, we can mention [2] working in constrained environments, the design of agents' architecture and the attribution of roles and possible specialization to each of them, agents' decision making process [17], the capacity to generate and execute autonomously plans of counter-measures in case of an attack, agents' autonomy, including versus trustworthiness, MAICA's safety to defense systems, cyber cognitive cooperation [18], agents' resilience in the face of attacks directed at them by enemy software, agents' learning capacities and the development of their functional autonomy, the specification and emergence of collective rules for the detection and resolution of cyber-attacks, AICA agents' deployment concepts and rationale, their integration into host hardware as [8] showed in industrial system contexts, etc.

IV. The Initial AICA Reference Architecture

To start the research with an initial assumption about agents' architecture, the IST-152-RTG designed the AICA Reference Architecture on the basis of classical perspective reflected in [19] and [20].

At the present moment, it is assumed to include the following functional components:

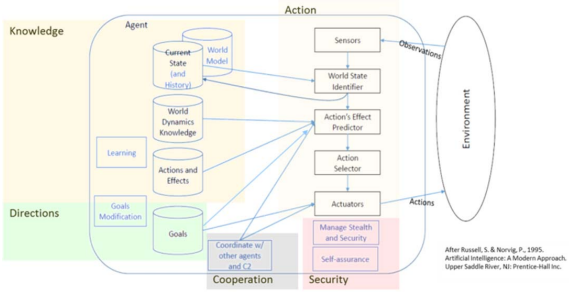

Figure 1 Assumed functional architecture of the AICA

The AICA Reference Architecture delivers five main high-level functions:

- Sensing and world state identification.
- Planning and action selection.
- Collaboration and negotiation.
- Action execution.
- Learning and knowledge improvement.

Figure 2 The AICA's main five high-level functions

A. Sensing and World state identification

DEFINITION: Sensing and World state identification is the AICA's high-level function that allows a cyber-defense agent to acquire data from the environment and systems in which it operates as well as from itself in order to reach an understanding of the current state of the world and, should it detect risks in it, to trigger the Planning and Action selection high-level function. This high-level function relies upon the "World model", "Current state and history", "Sensors" and "World State Identifier" components of the assumed functional architecture.

The Sensing and World state identification high-level function includes two functions: (1) Sensing; (2) Word state identification.

a-1. Sensing

DESCRIPTION: Sensing operates from two types of data sources: (1) External (system and device-related) current world state descriptors; (2) Internal (agent-related) current state descriptors.

Current world state descriptors, both external and internal, are captured on the fly by the agent's Sensing function. They may be double-checked, formatted or normalized for later use by the World state identification function (to create processed current state descriptors).

a-2. World state identification

DESCRIPTION: The World state identification function operates from two sources of data: (1) Processed current state descriptors; (2) Learnt world state patterns.

Learnt world state patterns are stored in the agent's world knowledge repository. Processed current state descriptors and Learnt world state patterns are compared to identify problematic current world state patterns (i.e. presenting an anomaly or a risk). When identifying a problematic current world state pattern, the World state identification function triggers the Planning and Action selection high-level function.

b. Planning and action selection

DEFINITION: Planning and action selection is the AICA's high-level function that allows a cyber-defense agent to elaborate one to several action proposals and to propose them to the Action selector function that decides the action or set of actions to execute in order to resolve the problematic world state pattern previously identified by the World state identifier function. This high-level function relies upon the "World dynamics", "Actions and effects", "Goals", "Actions' effect predictor" and "Action selector" components of the assumed functional architecture.

The Planning and action selector high-level function includes two functions: (1) Planning; (2) Action selector.

b-1. Planning

DESCRIPTION: The Planning function operates on the basis of two data sources: (1) Problematic current world state pattern; (2) Repertoire of actions (Response repertoire).

The Problematic current world state pattern and Repertoire of actions (Response repertoire) are concurrently explored in order to determine the action or set of actions (Proposed response plan) that can resolve the submitted problematic current world state pattern. The action or set of actions so determined are presented to the Action selector. It may be possible that the Planning function requires some form of cooperation with human operators (cyber cognitive cooperation, C3).

It may alternatively require cooperation with other agents or with a central cyber C2 (command and control) in order to come up with an optimal set of actions forming a global response strategy. Such cooperation could be either to request from other agents or from the cyber C2 complementary action proposals, or to delegate to the cyber C2 the responsibility of coordinating a global set of actions forming the wider response strategy.

It may be possible that the Planning function requires some form of cooperation with human operators (cyber cognitive cooperation, C3). It may alternatively require cooperation with other agents or with a central cyber C2 (command and control) in order to come up with an optimal set of actions forming a global response strategy. Such cooperation could be either to request from other agents or from the cyber C2 complementary action proposals, or to delegate to the cyber C2 the responsibility of coordinating a global set of actions forming the wider response strategy.

These aspects have been the object of an initial study in [17] where options such as offline machine learning, pattern recognition, online machine learning, escalation to a human operator, game theoretic option search, and failsafe have been envisaged, and in [18] for cyber cognitive cooperation processes.

b-2. Action selector

DESCRIPTION: The Action selector function operates on the basis of three data sources: (1) Proposed response plans; (2) Agent's goals; (3) Execution constraints and requirements, e.g., the environment's technical configuration, etc.

The proposed response plan is analyzed by the Action selector function in the light of the agent's current goals and of the execution constraints and requirements that may either be part of the world state descriptors gained through the Sensing and World state identifier high-level function or be stored in the agent's data repository and originated in the Learning and Knowledge improvement high-level function. The proposed response plan is then trimmed from whatever element does not fit the situation at hand, and augmented of prerequisite, preparatory or precautionary or post-execution recommended complementary actions. The Action selector thus produces an Executable Response Plan, and then submitted to the Action execution high-level function.

Like with the Planning function, it is possible that the Action selector function requires to liaise with human operators, other agents or a central cyber C2 (command and control) in order to come up with an optimal Executable Response Plan forming part of and being in line with a global response strategy. Such cooperation could be to exchange and consolidate information in order to come to a collective agreement on the assignment of the various parts of the global Executable Response Plan and the execution responsibilities to specific agents. It could alternatively be to delegate to the cyber C2 the responsibility of elaborating a consolidated Executable Response Plan and then to assign to specific agents the responsibility of executing part(s) of this overall plan within their dedicated perimeter. This aspect is not yet studied in the present release of the AICA Reference Architecture.

c. Collaboration and negotiation

DEFINITION: Collaboration and negotiation is the AICA's high-level function that allows a cyber-defense agent 1) to exchange information (elaborated data) with other agents or with a central cyber C2, for instance when one of the agent's functions is not capable on its own to reach satisfactory conclusions or usable results, and 2) to negotiate with its partners the elaboration of a consolidated conclusion or result. This high-level function relies upon the "Coordinate with other agents and C2" component of the assumed functional architecture.

The Collaboration and negotiation high-level function includes, at the present stage, one function: Collaboration and negotiation.

DESCRIPTION: The Collaboration and negotiation function operates on the basis of three data sources: (1) Internal, outgoing data sets (i.e. sent to other agents or to a central C2); (2) External, incoming data sets (i.e. received from other gents or from a central cyber C2); (3) Agents' own knowledge (i.e. consolidated through the Learning and knowledge improvement high-level function).

When an agent's Planning and action selector function or other function needs it, the agent's Collaboration and negotiation function is activated. Ad hoc data are sent to (selected) agents or to a central C2. The receiver(s) may be able, or not, to elaborate further on the basis of the data received through their own Collaboration and negotiation function. At this stage, when each agent (including possibly a central cyber C2) has elaborated further conclusions, it should share them with other (selected) agents, including (or possibly not) the one that placed the original request for collaboration. Once this (these multiple) response(s) received, the network of involved agents would start negotiating a consistent, satisfactory set of conclusions. Once an agreement reached, the concerned agent(s) could spark the next function within their own decision making process.

When the agent's own security is threatened the agent's Collaboration and negotiation function should help warning other agents (or a central cyber C2) of this state.

Besides, the agent's Collaboration and negotiation function may be used to receive warnings from other agents that may trigger the agent's higher state of alarm.

Finally, the agent's Collaboration and negotiation function should help agents discover other agents and establish links with them.

d. Action execution

DEFINITION: The Action execution is the AICA's high-level function that allows a cyber-defense agent to effect the Action selector function's decision about an Executable Response Plan (or the part of a global Executable Response Plan assigned to the agent), to monitor its execution and its effects, and to provide the agents with the means to adjust the execution of the plan (or possibly to dynamically adjust the plan) when and as needed. This high-level function relies upon the "Goals" and "Actuators" components of the assumed functional architecture.

The Action execution high-level function includes four functions:

- Action effector;
- Execution monitoring;
- Effects monitoring;
- Execution adjustment.

d-1. Action effector

DESCRIPTION: The Action effector function operates on the basis of two data sources:

- Executable Response Plan;
- Environment's Technical Configuration.

Taking into account the Environment's Technical Configuration, the Action effector function executes each planned action in the scheduled order.

d-2. Execution monitoring

DESCRIPTION: The Execution monitoring operates on the basis of two data sources:

- Executable Response Plan;
- Plan execution feedback.

The Execution monitoring function should be able to monitor (possibly through the Sensing function) each action's execution status (for instance: done, not done, and wrongly done). Any status apart from "done" should trigger the Execution adjustment function.

d-3. Effects monitoring

DESCRIPTION: The Effects monitoring function operates on the basis of two data sources: (1) Executable Response Plan; (2) Environment's change feedback.

It should be able to capture (possibly through the Sensing function) any modification occurring in the plan execution's environment. The associated dataset should be analyzed or explored. The result of such data exploration might provide a positive (satisfactory) or negative (unsatisfactory) environment change status. Should this status be negative, this should trigger the Execution adjustment function.

d-4. Execution adjustment

DESCRIPTION: The Execution adjustment function operates on the basis of three data sources: (1) Executable Response Plan; (2) Plan execution feedback and status; (3) Environment's change feedback and status.

The Execution adjustment function should explore the correspondence between the three data sets to find alarming associations between the implementation of the Executable Response Plan and its effects. Should warning signs be identified, the Execution adjustment function should either adapt the actions' implementation to circumstances or modify the plan.

e. Learning and knowledge improvement

DEFINITION: Learning and knowledge improvement is the AICA's high-level function that allows a cyber-defense agent to use the agent's experience to improve progressively its efficiency with regards to all other functions. This high-level function relies upon the Learning and Goals modification components of the assumed functional architecture.

The Learning and knowledge improvement high-level function includes two functions: (1) Learning; (2) Knowledge improvement.

e-1. Learning

DESCRIPTION: The Learning function operates on the basis of two data sources: (1) Feedback data from the agent's functioning; (2) Feedback data from the agent's actions.

The Learning function collects both data sets and analyzes the reward function of the agent (distance between goals and achievements) and their impact on the agent's knowledge database. Results feed the Knowledge improvement function.

e-2. Knowledge improvement

DESCRIPTION: The Knowledge improvement function operates on the basis of two data sources: (1) Results (propositions) from the Learning function; (2) Current elements of the agent's knowledge.

The Knowledge improvement function merges Results (propositions) from the Learning function and the Current elements of the agent's knowledge.

V. PRELIMINARY RESEARCH ASSUMPTIONS AND QUESTIONS: THE EXAMPLE OF LEARNING

The environment of the agent can change rapidly, especially (but not exclusively) due to an enemy action. In addition, the enemy malware, its capabilities and Tactics, Techniques and Procedures (TTP), evolve rapidly. Therefore, the agent must be capable of autonomous learning. The reasoning capabilities of the agent rely on its knowledge bases (KBs). The purpose of the learning function(s) of the agent is to modify the KBs of the agent in a way that enhances the success of the agent's actions. The agent learns from its experiences. Therefore, the most general cycle of the learning process is the following:

1. The agent possesses a KB.
2. The agent uses the KB to perform actions; he also makes observation (receives percepts). These together constitute the agent's experience.
3. The agent uses this experience to learn the desirable modifications to the KB.
4. The agent modifies the KB.
5. Repeat.

The agent's experience needs a formal representation. It may look like this sequence:

(t_1, a_1, e_1, R_1) $(t_2, a_2, NULL, NULL)$ $(t_3, NULL, e_3, R_3)$ … (t_n, a_n, e_n, R_n)

Where t_1 is the time when the agent starts to record his experience and t_n is the moment "now", an is an action, e_n is a percept, R_n is the reward of the action.

To make the representation of knowledge more compact and useful, we could divide it into shorter chunks where each chunk ends with the moment when the agent is able to determine a reward. We could call such a chunk an episode. Episode E_j is a sequence of pairs $\{a_1, e_i\}$, and the resulting reward R_j:

$E_j = (\{a_i,, e_i\}, R_j)$

The following is an example of such a short episode: a1 - check file system integrity; e1 - find unexpected file; a2 - delete file; e2 - file gone; a3 – NULL; e3 - observe Enemy C2 traffic; Reward - 0.09

A representation of this nature could be used in a case-based reasoning, or in a deep learning approach.

What exactly could an agent learn? One, fairly general option is that Learning Module learns the World Dynamics model which is a function that takes as an input a state and an action applied to that state; its output is a new state that will result from application of that action, or a distribution of states. World Dynamics Model is used in particular in "Action Selector and Predictor" module. In addition, the Learning Module can learn another function required in "Action Selector and Predictor" module, which maps the current world state to a set of feasible actions.

VI. CONCLUSIONS AND FUTURE RESEARCH

Intelligent, partly autonomous agents are likely to become primary cyber fighters on the future battlefield. Our initial exploration identified the key functions, components and their interactions for a potential reference architecture of such an agent.

The AICA Reference Architecture was derived from [19] for we needed a broad, cognition-based, all-encompassing agent structure. Future works will challenge this initial choice.

Embedding AICA agents into highly constrained military systems is also the focus of future research and this issue was not addressed at this stage yet.

And, at the present stage, the AICARA architecture is a preliminary proposal. Its feasibility as well as its power to fight malware autonomously and intelligently remain to be evaluated.

With respect to further efforts, this research group plans to have a basic proof-of-concept prototype developed and tested by 2019.The current priorities are:

- To study use cases as a reference for the research, as this will lead to clarifying the scope, concepts, functionality and functions' inputs and outputs of AICA and MAICA systems; use cases will be based on the one elaborated in the IST-152 intermediary report [21];
- To refine the initially assumed architecture by drawing further lessons from the case studies;
- To determine the set of technologies that AICAs should embark and that need to be tested during the prototyping phase;
- To define the methodology of the tests.

The sum of challenges presented by the AICA / MAICA concept appears, today, very substantial, although our initial analysis suggests that the required technical approaches do not seem to be entirely beyond the current state of the research. An empirical research program and collaboration of multiple teams should be able to produce significant results and solutions for a robust, effective intelligent agent. This might happen within a time span that could currently be assumed on the order of ten years.

REFERENCES

[1] F. De Gaspari, S. Jajodia, L. V. Mancini and A. Panico, "AHEAD: A New Architecture for Active Defense,"

SafeConfig'16, October 24 2016, Vienna, Austria, 2016.

[2] P. Théron, La cyber résilience, un projet cohérent transversal à nos trois thèmes, et la problématique particulière des Systèmes Multi Agent de Cyber Défense, Leçon inaugurale, 5 décembre 2017, ed., France, Salon de Provence: Chaire Cyber Résilience Aérospatiale (Cyb'Air), 2017.

[3] R. Rasch, A. Kott and K. D. Forbus, "AI on the battlefield: An experimental exploration," *AAAI/IAAI,* 2002.

[4] R. Rasch, A. Kott and K. D. Forbus, "Incorporating AI into military decision making: an experiment," *IEEE Intelligent Systems,* vol. 18, no. 4, pp. 18-26, 2003.

[5] J. Von Neumann, "The General and Logical Theory of Automata," in *Cerebral Mechanisms in Behavior: The Hixon Symposium, September 1948, Pasadena*, L. A. Jeffress, Ed., New York, John Wiley & Sons, Inc, 1951, pp. 1-31.

[6] S. D. McArthur, E. M. Davidson, V. M. Catterson, A. L. Dimeas, N. D. Hatziargyriou, F. Ponci and T. Funabashi, "Multi-Agent Systems for Power Engineering Applications - Part I: Concepts, Approaches, and Technical Challenges," *IEEE TRANSACTIONS ON POWER SYSTEMS,* vol. 22, no. 4, pp. 1743-1752, 2007.

[7] A. Carrasco, M. C. Romero-Ternero, F. Sivianes, M. D. Hernández and J. I. Escudero, "Multi-agent and embedded system technologies applied to improve the management of power systems," *JDCTA,* vol. 4, no. 1, pp. 79-85, 2010.

[8] M. Pechoucek and V. Marík, "Industrial deployment of multi-agent technologies: review and selected case studies," *Autonomous Agents and Multi-Agent Systems,* vol. 17, p. 397–431, 2008.

[9] W. Elmenreich, "Intelligent methods for embedded systems," in *Proceedings of the First Workshop on Intelligent Solutions in Embedded Systems*, J. 2. Vienna University of Technology 2003, Ed., Austria: Vienna, Vienna University of Technology, 2003, pp. 3-11.

[10] H.-P. Huang, C.-C. Liang and C.-W. Lin, "Construction and soccer dynamics analysis for an integrated multi-agent soccer robot system," *Natl. Sci. Counc. ROC(A),* vol. 25, pp. 84-93, 2001.

[11] J.-P. Jamont, M. Occello and A. Lagrèze, "A multiagent approach to manage communication in wireless instrumentation systems," *Measurement,* vol. 43, no. 4, pp. 489-503, 2010.

[12] B. Chen and H. H. Cheng, "A review of the applications of agent technology in traffic and transportation systems," *Trans. Intell. Transport. Sys.,* vol. 11, no. 2, pp. 485-497, 2010.

[13] J.-P. Jamont and M. Occello, "A framework to simulate and support the design of distributed automation and decentralized control systems: Application to control of indoor building comfort," in *IEEE Symposium on Computational Intelligence in Control and Automation*, Paris, France, IEEE, 2011, pp. 80-87.

[14] M. R. Stytz, D. E. Lichtblau and S. B. Banks, "Toward using intelligent agents to detect, assess, and counter cyberattacks in a network-centric environment," Institute For Defense Analyses, Alexandria, VA, 2005.

[15] J.-P. Muller, "Emergence of collective behaviour and problem solving," in *Engineering Societies in the Agents World IV*, A. Omicini, P. Petta and J. Pitt, Eds., volume 3071, Lecture Notes in Computer Science, 2004, pp. 1-20.

[16] K. Weick, "The Collapse of Sensemaking in Organizations: The Mann Gulch Disaster," *Administrative Science Quarterly,* vol. 38, no. 4, pp. 628-652, 1993.

[17] B. Blakely and P. Theron, *Decision flow-based Agent Action Planning,* Prague, 2017.

[18] B. LeBlanc, P. Losiewicz and S. Hourlier, *A Program for effective and secure operations by Autonomous Agents and Human Operators in communications constrained tactical environments,* Prague, 2017.

[19] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, Upper Saddle River, NJ: Prentice-Hall Inc, 1995.

[20] S. J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 2nd ed. ed., Upper Saddle River, New Jersey: Prentice Hall, 2003.

[21] A. Kott, L. V. Mancini, P. Theron, M. Drašar, H. Günther, M. Kont, M. Pihelgas, B. LeBlanc and K. Rzadca, "Initial Reference Architecture of an Intelligent Autonomous Agent for Cyber Defense," US Army Research Laboratory, ARL-TR-8337, March 2018, available from https://arxiv.org/abs/1803.10664, Adelphi, MD, 2018.

Appendix 8

Publication VIII
P. Théron, A. Kott, M. Drašar, K. Rzadca, B. LeBlanc, M. Pihelgas, L. Mancini, and F. de Gaspari. Reference Architecture of an Autonomous Agent for Cyber Defense of Complex Military Systems. In *Adaptive Autonomous Secure Cyber Systems*, pages 1–21, Cham, 2020. Springer International Publishing

The chapter is included in the book *Adaptive Autonomous Secure Cyber Systems*.
DOI: 978-3-030-33432-1_1

Reference Architecture of an Autonomous Agent for Cyber Defense of Complex Military Systems

Paul Theron, Alexander Kott, Martin Drašar, Krzysztof Rzadca, Benoît LeBlanc, Mauno Pihelgas, Luigi Mancini, and Fabio de Gaspari

1 Future Military Systems and the Rationale for Autonomous Intelligent Cyber Defense Agents

Modern defense systems incorporate new technologies like cloud computing, artificial intelligence, lasers, optronics, electronics and submicronic processors, on-board power-generation systems, automation systems, sensors, software defined radios

This chapter reuses portions of an earlier paper: Theron, P., et al, "Towards an Active, Autonomous and Intelligent Cyber Defense of Military Systems: the NATO AICA Reference Architecture", Proceedings of the International Conference on Military Communications and Information Systems Warsaw, Poland, 22nd - 23rd May 2018; © 2018 IEEE.

P. Theron
Aerospace Cyber Resilience Chair, Paris, France
e-mail: paul.theron@thalesgroup.com

A. Kott (✉)
U.S. Army Research Laboratory, Adelphi, MD, USA
e-mail: alexander.kott1.civ@mail.mil

M. Drašar
Masaryk University, Brno, Czech Republic
e-mail: drasar@ics.muni.cz

K. Rzadca
University of Warsaw, Warsaw, Poland
e-mail: krzadca@mimuw.edu.pl

B. LeBlanc
Ecole Nationale Supérieure de Cognitique, Bordeaux, France
e-mail: benoit.leblanc@ensc.fr

S. Jajodia et al. (eds.), *Adaptive Autonomous Secure Cyber Systems*,
https://doi.org/10.1007/978-3-030-33432-1_1

and networks, etc. They are more and more relying on software, and will also embed new hardware technologies, including high performance computers and quantum technologies, nanoparticles, metamaterials, self-reconfigurable hardware, etc.

While defense infrastructures and systems engaged on the battle ground may not fail, the multitude of high-tech features and interconnections that they embed make cyber-attacks a good way to affect their functionality and the missions in which they are involved.

Today, five broad classes of systems coexist in Land, Sea and Air operations:

- Office and information management systems: these include web services, emailing systems, and information management applications ranging from human resource management to logistics through maintenance and project management;
- C4ISR systems for the command of war operations: they include associated Battlefield Management Systems that extend the C4ISR down to single vehicles and platoons;
- Communication systems: they include SATCOM, L16, line of sight networks, software defined radios, and the Internet of Battle Things (IoBT) can be seen as a major operational innovation and extension of communication capabilities;
- Platform and life automation systems: they are similar to industrial systems and provide sea vessels or armored vehicles, for instance, with capabilities such as air conditioning, refrigeration, lifts, video surveillance, etc.;
- Weapon systems: these include sensors and effectors of all kinds, operating in all kinds of situations and contested battle grounds.

On the battlefield, these platforms and technologies will operate together in complex large scale networks of massively interconnected systems.

Autonomy can be defined as the capacity of systems to decide by themselves on their course of action in uncertain and challenging environments without the help of human operators. It should not be confused with automation, the aptitude of systems to perform set tasks according to set rules in environments where uncertainty is low and characterized [11].

Despite the fact that "Full autonomy might not necessarily be the objective" and the existence of a variety of definitions [5], the number of autonomous military systems will grow [10]. They will be able to mitigate operational challenges such as needs for rapid decision-making, high heterogeneity and/or volumes of data, intermittent communications, the high complexity of coordinated actions, or the danger of missions, while they will require persistence and endurance [11, p. 12].

M. Pihelgas
NATO Cooperative Cyber Defence Centre of Excellence, Tallinn, Estonia
e-mail; mauno.pihelgas@ccdcoe.org

L. Mancini · F. de Gaspari
Sapienza University, Rome, Italy
e-mail: mancini@di.uniroma1.it; degaspari@di.uniroma1.it

Examples of autonomous capabilities and systems [11] include:

- Unmanned Air Systems, Unmanned Surface Vehicles and Unmanned Underwater Vehicles, will be able to carry out reconnaissance or attack missions stealthily, some of them with a large endurance. For instance, the Haiyan UUV [24] is a mini-submarine built on a civilian platform that China's People's Liberation Army Navy (PLAN) sponsors to create an autonomous UUV capable of carrying out dangerous missions like minesweeping and submarine detection operations without any human intervention. Weighing only 70 kg, energy efficient and fitted with advanced computing capacities, it has an endurance of up to 30 days.
- Today's Intelligence, Surveillance & Recognition (ISR) missions request more and more high-definition (HD) images and videos being captured and transmitted back to ground stations for analysis. As HD means large volumes of raw data (possibly encrypted, which adds to volumes), communication means cannot provide the ad hoc transmission throughput (and continuity in contested environments). Autonomous sensors equipped with artificial intelligence will be capable of generating on the ground aggregated, high-level information that can be more easily transmitted to command posts as they require much less bandwidth than raw data, also lowering the human workload needed to process high volumes of complex multi-source raw data.
- Autonomous Unmanned Ground Vehicles can be employed in dealing with chemical, biological, radiological and nuclear (CBRN) threats as well as with Improvised Explosive Devices (IED), as was the case in Iraq and Afghanistan conflicts.
- The US MK-18 Mod 2 program has demonstrated significant progress in utilizing Remote Environmental Monitoring UnitS (REMUS) Unmanned Underwater Vehicles for mine countermeasure missions, thus allowing pulling military personnel away from dangerous mine fields and reducing tactical reaction times.
- Unmanned Aircrafts (UA) could be used in anti-access and area denial (A2/AD) missions to perform functions that today require the intervention of personnel such as aerial refueling, airborne early warning, ISR, anti-ship warfare, command, offensive strike facilitation (electronic warfare, communications jamming, decoys) and actions supporting defense by creating confusion, deception or attrition through decoys, sensors and emitters, target emulators. Similar functions could be used in underwater combat.
- Agile ground forces could get local tactical support from Unmanned Aircraft Systems (UAS) embarking sensors, ISR capacities, communication means, electronic warfare functions and weapon systems. These UAS would reach greater levels of efficiency and could better deal with large numbers of ground, air and possibly sea sensors and actuators if they could themselves work in swarms or cohorts and collectively adapt their action dynamically on the basis of mission and environment-related data collected in real time.
- Logistics could be another area of utilization of autonomous land, sea and air vehicles and functions. Autonomous capabilities could be used in contested changeable environments either in support and defense of friendly logistic deployment and operation, or to disturb or strike enemy logistics.

Another two fundamental issues need to be taken into account.

The first of these issues is the fact that the level of interconnectedness, and therefore of interdependence and cross-vulnerability, of military systems will increase to unseen heights [42].

The Internet of Things is increasing rapidly in both numbers and types of smart objects, and this is a durable trend with regards to Defense [11] despite the massive scale of their deployment, their meager configurability and the new (cyber) risks they create. In effect, with the shift to the IPv6 addressing standard, the number of devices that can be networked is up to 340 undecillion unique devices (340 with 36 zeroes after it) and this immense network of interconnected devices could become a global platform for massively proliferated, distributed cyber-attacks [11].

This multitude of devices will work together in clusters, likely hard to map out, likely subject to unstable changing configurations of their dependencies. These changes will occur beyond our control because of the degrees of autonomy conferred to objects in shifting operative conditions.

Massively interconnected military systems will become more and more difficult to engineer, test, maintain, operate, protect and monitor [42], which leads the authors to recommend "reducing the number of interconnections by reversing the default culture of connecting systems whenever possible" to improve cybersecurity. This recommendation, however intelligent it seems, is very likely never to be listened to . . .

Thus, cyber defending such complex systems will become arduous. For instance, they will not anymore allow for the sort of cybersecurity monitoring we currently deploy across IT and OT systems as they will prevent the implementation of classic, centralized, and even big data/machine learning-based security operations centers (SOCs).

They will also overwhelm human SOC operators' cognitive capacities as it will become impossible for the latter to get instantly a clear and adequate picture of the systems they defend, of their condition, of the adverse events taking place and of the remedies to apply and of their possible impacts.

To defend them against cyber-attacks, only locally implemented distributed and resilient swarms of cyber defense agents adapting to these frequent reconfigurations and emerging circumstances will be able to monitor and defend this vast fuzzy network, learning superior abilities from cumulated experience.

In this particular context, different from the previously exposed context of the cyber defense of a few well-identified and carefully managed autonomous mission systems, cyber defense agents will evolve themselves into more and more unknown, less and less controllable and maintainable states.

Given this last parameter, they may either show decreasing levels of efficiency or generate uncontrollable adverse effects.

The second issue stems from the fundamental military need to proceed successfully with defense missions while operational personnel of Air, Land and Sea forces are not primarily specialists of cybersecurity and cyber defense. This is not to mention that on the battlefield there will always be a scarcity of cyber competencies [22].

Cyber-attacks may cause human operators sometimes to be fooled, for instance when radar or GPS data are spoofed, or stressed, for instance when anomalies multiply while their cause appears unclear and their consequences detrimental. Studies in a variety of domains such as air, sea and ground transportation have drawn attention to this phenomenon. Attacks may trigger human errors of varying consequences. For instance, NAP [29] points out that "Inaccurate information sent to system operators, either to disguise unauthorized changes, or to cause the operators to initiate inappropriate actions, [] could have various negative effects".

The burden of cyber defending systems must therefore be relieved from unqualified operators' shoulders, while the lack of specialists of cybersecurity on the ground prohibits calling upon rapid response teams in case of trouble.

In this context, handling cyber-attacks occurring in the course of operations requires an embedded, undisturbing, seamless autonomous intelligent cyber defense technology [45]. Autonomous intelligent cyber defense agents should resolve (at - least most of) cyber-attacks without technology users being aware of issues at hand.

Only when they would reach their limits, i.e. when being unable to understand situations, to reconciliate disparate pieces of information, or to elaborate cyber defense counter-measures, such multiple agents should collaborate with human operators. NAP [28] provides inspiring examples of machine-human collaboration in a variety of contexts. Such a need for collaboration might also exist in the context of massively interconnected systems of systems evoked earlier.

2 NATO's AICA Reference Architecture: A Concept for Addressing the Need for an Autonomous Intelligent Cyber Defense of Military Systems

Inspired by the above rationale, NATO's IST-152 Research and Technology Group (RTG) is an activity that was initiated by the NATO Science and Technology Organization and was kicked-off in September 2016. The group has developed is developing a comprehensive, use case focused technical analysis methodology in order to produce a first-ever reference architecture and technical roadmap for active autonomous intelligent cyber defense agents. In addition, the RTG worked to identify and evaluate selected elements that may be eligible contributors to such capabilities and that begin to appear in academic and industrial research.

Scientists and engineers from several NATO Nations have brought unique expertise to this project. Only by combining multiple areas of distinct knowledge along with a realistic and comprehensive approach can such a complex software agent be provided.

The output of the RTG may become a tangible starting point for acquisition activities by NATO Nations. If based on a common reference architecture, software agents developed or purchased by different Nations will be far more likely to be interoperable.

Related research includes Mayhem (from DARPA Cyber Challenge, but also Xandra, etc.), agents from the Pechoucek's group, Professor Mancini's work on the AHEAD architecture [9] and the Aerospace Cyber Resilience research chair's research program [45], Anti-Virus tools (Kaspersky, Bitdefender, Avast, Norton, etc.), HBSS, OSSEC, Various host-based IDS/IPS systems, Application Performance Monitoring Agents, Anti-DDOS systems and Hypervisors. Also, a number of related research directions include topics such as deep learning (especially if it can be computationally inexpensive), Botnet technology (seen as a network of agents), network defense games, flip-it games, the Blockchain, and fragmentation and replication. The introduction of Artificial Intelligence into military systems, such as C4ISR, has been studied, for instance by Rasch et al. [35, 36]. Multi Agent Systems form an important part of AI.

Since the emergence of the concept of Multi Agent Systems, e.g., [46], MAS have been deployed in a number of contexts such as power engineering [25] and their decentralized automated surveillance [7], industrial systems [33], networked and intelligent embedded systems [16], collective robotics [19], wireless communication [21], traffic simulation and logistics planning [8], home automation [20], etc.

However, if the use of intelligent agents for the cyber defense of network-centric environments has already long been envisaged [43], effective research in this area is still new.

In the context of the cyber defense of friendly systems, an "agent" has been defined [45] as a piece of software or hardware, a processing unit capable of deciding on its own about its course of action in uncertain, possibly adverse, environments:

- With an individual mission and the corresponding competencies, i.e. in analyzing the milieu in which the agent is inserted, detecting attacks, planning the required countermeasures, or steering and adapting tactically the execution of the latter, or providing support to other agents like for instance inter-agent communication;
- With proactivity, i.e. the capacity to engage into actions and campaigns without the need to be triggered by another program or by a human operator;
- With autonomy, i.e. a decision making capacity of its own, the capacity to function or to monitor, control and repair itself on its own, without the need to be controlled by another program or by a human operator, and the capacity to evaluate the quality of its own work and to adjust its algorithms in case of deviance from its norm or when its rewards (satisfaction of its goals) get poor;
- Driven by goals, decision making and other rules, knowledge and functions fit for its purpose and operating circumstances;
- Learning from experience to increase the accuracy of its decisions and the power of its reactions;
- With memories (input, process, output, storage);
- With perception, sensing and action, and actuating interfaces;
- Built around the adequate architecture and appropriate technologies;
- Positioned around or within a friendly system to defend, or patrolling across a network;

- Sociable, i.e. with the capacity to establish contact and to collaborate with other agents, or to enter into a cyber cognitive cooperation when the agent requires human help or to cooperate with a central Cyber C2;
- Trustworthy, i.e. that will not deceive other agents nor human operators;
- Reliable; i.e. that do what they are meant to do, during the time specified and under the conditions and circumstances of their concept of operation;
- Resilient, i.e. both robust to threats (including cyber-threats aimed at disabling or destroying the agent itself; the agent being able to repel or withstand everyday adverse events and to avoid degrading), and resistant to incidents and attacks that may hit and affect the agent when its robustness is insufficient (i.e. the agent is capable of recovering from such incidents or attacks);
- Safe, i.e., conceived to avoid harming the friendly systems the agent defends, for instance by calling upon a human supervisor or central cyber C2 to avoid making wrong decisions or to adjust their operating mode to challenging circumstances, or by relocating when the agent is the target of an attack and if relocation is feasible and allows protecting it, or by activating a fail-safe mode, or by way of self-destruction when no other possibility is available.

In the same context (ibid), a multi agent system is a set of agents:

- Distributed across the parts of the friendly system to defend;
- Organized in a swarm (horizontal coordination) or cohort (vertical coordination);
- In which agents may have homogeneous or heterogeneous roles and features;
- Interoperable and interacting asynchronously in various ways such as indifference, cooperation, competition;
- Pursuing a collective non-trivial cyber defense mission, i.e. allowing to piece together local elements of situation awareness or propositions of decision, or to split a counter-attack plan into local actions to be driven by individual agents;
- Capable of self-organization, i.e. as required by changes in circumstances, whether external (the attack's progress or changes in the friendly system's health or configuration) or internal (changes in the agents' health or status);
- That may display emergent behaviors [26], i.e. performances that are not explicitly expressed in individual agents' goals, missions and rules; in the context of cyber defense, "emergence" is likely to be an interesting feature as, consisting in the "way to obtain dynamic results, from cooperation, that cannot easily be predicted in a deterministic way" [26]; it can be disturbing to enemy software in future malware-goodware "tactical" combats within defense and other complex systems;
- Extensible or not, i.e. open or closed to admitting new agents in the swarm or cohort;
- Safe, trustworthy, reliable and resilient as a whole, which is a necessity in the context of cyber defense whereas in other, less challenging contexts may be unnecessary. Resilience, here, may require maintaining a system of virtual roles as described in a human context by Weick [47].

AICA will not be simple agents. Their missions, competencies, functions and technology will be a challenging construction in many ways.

Among many such challenges, we can mention [45] working in resource-constrained environments, the design of agents' architecture and the attribution of roles and possible specialization to each of them, agents' decision making process [3], the capacity to generate and execute autonomously plans of counter-measures in case of an attack, agents' autonomy, including versus trustworthiness, MAICA's safety to defense systems, cyber cognitive cooperation [23], agents' resilience in the face of attacks directed at them by enemy software, agents' learning capacities and the development of their functional autonomy, the specification and emergence of collective rules for the detection and resolution of cyber-attacks, AICA agents' deployment concepts and rationale, their integration into host hardware as [33] showed in industrial system contexts, etc.

To start the research with an initial assumption about agents' architecture, the IST-152-RTG designed the AICA Reference Architecture [22] on the basis of classical perspective reflected in [37].

At the present moment, it is assumed to include the following functional components (Fig. 1).

The AICA Reference Architecture delivers five main high-level functions (Fig. 2):

- Sensing and world state identification.
- Planning and action selection.
- Collaboration and negotiation.
- Action execution.
- Learning and knowledge improvement.

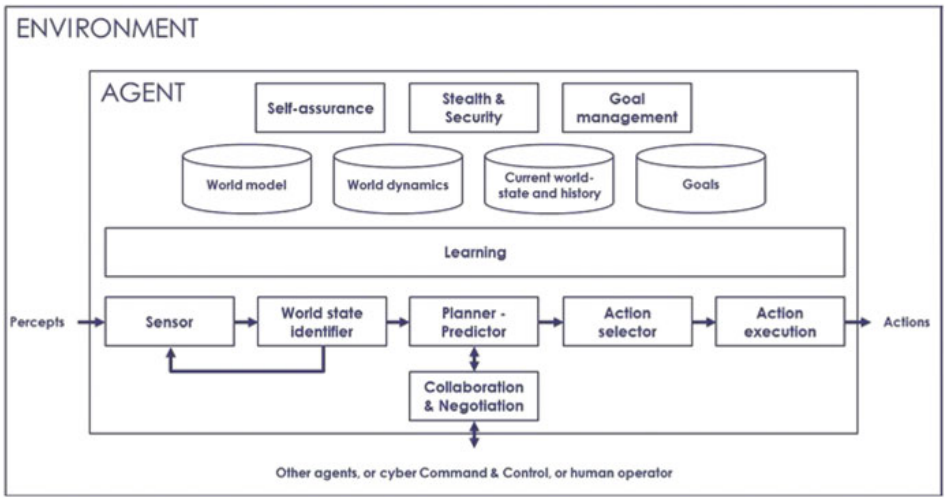

Fig. 1 Assumed functional architecture of the AICA

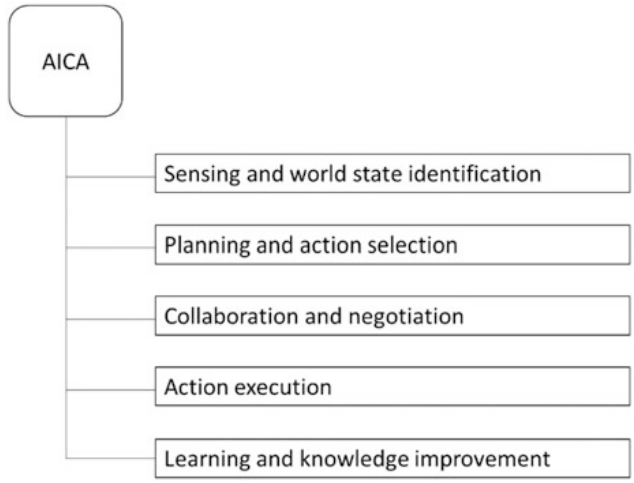

Fig. 2 The AICA's main five high-level functions

2.1 Sensing and World State Identification

Definition: Sensing and World state identification is the AICA's high-level function that allows a cyber-defense agent to acquire data from the environment and systems in which it operates as well as from itself in order to reach an understanding of the current state of the world and, should it detect risks in it, to trigger the Planning and Action selection high-level function. This high-level function relies upon the "World model", "Current world state and history", "Sensors" and "World State Identifier" components of the assumed functional architecture.

The Sensing and World state identification high-level function includes two functions: (1) Sensing; (2) Word state identification.

2.1.1 Sensing

Description: Sensing operates from two types of data sources: (1) External (system and device-related) current world state descriptors; (2) Internal (agent-related) current state descriptors.

Current world state descriptors, both external and internal, are captured on the fly by the agent's Sensing function. They may be double-checked, formatted or normalized for later use by the World state identification function (to create processed current world state descriptors).

2.1.2 World State Identification

Description: The World state identification function operates from two sources of data: (1) Processed current world state descriptors; (2) Learnt world state patterns.

Learnt world state patterns are stored in the agent's world knowledge repository. Processed current world state descriptors and Learnt world state patterns are compared to identify problematic current world state patterns (i.e. presenting an anomaly or a risk). When identifying a problematic current world state pattern, the World state identification function triggers the Planning and Action selection high-level function.

2.2 Planning and Action Selection

Definition: Planning and action selection is the AICA's high-level function that allows a cyber-defense agent to elaborate one to several action proposals and to propose them to the Action selector function that decides the action or set of actions to execute in order to resolve the problematic world state pattern previously identified by the World state identifier function. This high-level function relies upon the "World dynamics" that should include knowledge about "Actions and effects", "Goals", "Planner - Predictor" and "Action selector" components of the assumed functional architecture.

The Planning and action selector high-level function includes two functions: (1) Planning; (2) Action selector.

2.2.1 Planning

Description: The Planning function operates on the basis of two data sources: (1) Problematic current world state pattern; (2) Repertoire of actions (Response repertoire).

The Problematic current world state pattern and Repertoire of actions (Response repertoire) are concurrently explored in order to determine the action or set of actions (Proposed response plan) that can resolve the submitted problematic current world state pattern. The action or set of actions so determined are presented to the Action selector. It may be possible that the Planning function requires some form of cooperation with human operators (cyber cognitive cooperation, C3).

It may alternatively require cooperation with other agents or with a central cyber C2 (command and control) in order to come up with an optimal set of actions forming a global response strategy. Such cooperation could be either to request from other agents or from the cyber C2 complementary action proposals, or to delegate to the cyber C2 the responsibility of coordinating a global set of actions forming the wider response strategy.

It may be possible that the Planning function requires some form of cooperation with human operators (cyber cognitive cooperation, C3). It may alternatively require cooperation with other agents or with a central cyber C2 (command and control) in order to come up with an optimal set of actions forming a global response strategy. Such cooperation could be either to request from other agents or from the cyber C2

complementary action proposals, or to delegate to the cyber C2 the responsibility of coordinating a global set of actions forming the wider response strategy.

These aspects have been the object of an initial study in [3] where options such as offline machine learning, pattern recognition, online machine learning, escalation to a human operator, game theoretic option search, and failsafe have been envisaged, and in [23] for cyber cognitive cooperation processes.

2.2.2 Action Selector

Description: The Action selector function operates on the basis of three data sources: (1) Proposed response plans; (2) Agent's goals; (3) Execution constraints and requirements, e.g., the environment's technical configuration, etc.

The proposed response plan is analyzed by the Action selector function in the light of the agent's current goals and of the execution constraints and requirements that may either be part of the world state descriptors gained through the Sensing and World state identifier high-level function or be stored in the agent's data repository and originated in the Learning and Knowledge improvement high-level function. The proposed response plan is then trimmed from whatever element does not fit the situation at hand, and augmented of prerequisite, preparatory or precautionary or post-execution recommended complementary actions. The Action selector thus produces an Executable Response Plan, and then submitted to the Action execution high-level function.

Like with the Planning function, it is possible that the Action selector function requires to liaise with human operators, other agents or a central cyber C2 (command and control) in order to come up with an optimal Executable Response Plan forming part of and being in line with a global response strategy. Such cooperation could be to exchange and consolidate information in order to come to a collective agreement on the assignment of the various parts of the global Executable Response Plan and the execution responsibilities to specific agents. It could alternatively be to delegate to the cyber C2 the responsibility of elaborating a consolidated Executable Response Plan and then to assign to specific agents the responsibility of executing part(s) of this overall plan within their dedicated perimeter. This aspect is not yet studied in the present release of the AICA Reference Architecture.

2.3 Collaboration and Negotiation

Definition: Collaboration and negotiation is the AICA's high-level function that allows a cyber-defense agent (1) to exchange information (elaborated data) with other agents or with a central cyber C2, for instance when one of the agent's functions is not capable on its own to reach satisfactory conclusions or usable results, and (2) to negotiate with its partners the elaboration of a consolidated

conclusion or result. This high-level function relies upon the "Collaboration & Negotiation" component of the assumed functional architecture.

The Collaboration and negotiation high-level function includes, at the present stage, one function: Collaboration and negotiation.

Description: The Collaboration and negotiation function operates on the basis of three data sources: (1) Internal, outgoing data sets (i.e. sent to other agents or to a central C2); (2) External, incoming data sets (i.e. received from other gents or from a central cyber C2); (3) Agents' own knowledge (i.e. consolidated through the Learning and knowledge improvement high-level function).

When an agent's Planning and action selector function or other function needs it, the agent's Collaboration and negotiation function is activated. Ad hoc data are sent to (selected) agents or to a central C2. The receiver(s) may be able, or not, to elaborate further on the basis of the data received through their own Collaboration and negotiation function. At this stage, when each agent (including possibly a central cyber C2) has elaborated further conclusions, it should share them with other (selected) agents, including (or possibly not) the one that placed the original request for collaboration. Once this (these multiple) response(s) received, the network of involved agents would start negotiating a consistent, satisfactory set of conclusions. Once an agreement reached, the concerned agent(s) could spark the next function within their own decision making process.

When the agent's own security is threatened the agent's Collaboration and negotiation function should help warning other agents (or a central cyber C2) of this state.

Besides, the agent's Collaboration and negotiation function may be used to receive warnings from other agents that may trigger the agent's higher state of alarm.

Finally, the agent's Collaboration and negotiation function should help agents discover other agents and establish links with them.

2.4 Action Execution

Definition: The Action execution is the AICA's high-level function that allows a cyber-defense agent to effect the Action selector function's decision about an Executable Response Plan (or the part of a global Executable Response Plan assigned to the agent), to monitor its execution and its effects, and to provide the agents with the means to adjust the execution of the plan (or possibly to dynamically adjust the plan) when and as needed. This high-level function relies upon the "Goals" and "Action execution" components of the assumed functional architecture.

The Action execution high-level function includes four functions:

- Action effector;
- Execution monitoring;
- Effects monitoring;
- Execution adjustment.

2.4.1 Action Effector

Description: The Action effector function operates on the basis of two data sources:

- Executable Response Plan;
- Environment's Technical Configuration.

Taking into account the Environment's Technical Configuration, the Action effector function executes each planned action in the scheduled order.

2.4.2 Execution Monitoring

Description: The Execution monitoring operates on the basis of two data sources:

- Executable Response Plan;
- Plan execution feedback.

The Execution monitoring function should be able to monitor (possibly through the Sensing function) each action's execution status (for instance: done, not done, and wrongly done). Any status apart from "done" should trigger the Execution adjustment function.

2.4.3 Effects Monitoring

Description: The Effects monitoring function operates on the basis of two data sources: (1) Executable Response Plan; (2) Environment's change feedback.

It should be able to capture (possibly through the Sensing function) any modification occurring in the plan execution's environment. The associated dataset should be analyzed or explored. The result of such data exploration might provide a positive (satisfactory) or negative (unsatisfactory) environment change status. Should this status be negative, this should trigger the Execution adjustment function.

2.4.4 Execution Adjustment

Description: The Execution adjustment function operates on the basis of three data sources: (1) Executable Response Plan; (2) Plan execution feedback and status; (3) Environment's change feedback and status.

The Execution adjustment function should explore the correspondence between the three data sets to find alarming associations between the implementation of the Executable Response Plan and its effects. Should warning signs be identified, the Execution adjustment function should either adapt the actions' implementation to circumstances or modify the plan.

2.5 Learning and Knowledge Improvement

Definition: Learning and knowledge improvement is the AICA's high-level function that allows a cyber-defense agent to use the agent's experience to improve progressively its efficiency with regards to all other functions. This high-level function relies upon the Learning and Goals modification components of the assumed functional architecture.

The Learning and knowledge improvement high-level function includes two functions: (1) Learning; (2) Knowledge improvement.

2.5.1 Learning

Description: The Learning function operates on the basis of two data sources: (1) Feedback data from the agent's functioning; (2) Feedback data from the agent's actions.

The Learning function collects both data sets and analyzes the reward function of the agent (distance between goals and achievements) and their impact on the agent's knowledge database. Results feed the Knowledge improvement function.

2.5.2 Knowledge Improvement

Description: The Knowledge improvement function operates on the basis of two data sources: (1) Results (propositions) from the Learning function; (2) Current elements of the agent's knowledge.

The Knowledge improvement function merges Results (propositions) from the Learning function and the Current elements of the agent's knowledge.

3 Use Cases

The use-case of military UAVs that operate in teams illustrates a possible deployment of the AICA Reference Architecture. It is based on the AgentFly project developed within the Agent Technology Center [44].

The AgentFly project facilitates the simulation of multi agent Unmanned Aerial Vehicles (UAV). Its features include flight path planning, decentralized collision avoidance and models of UAVs, physical capabilities and environmental conditions [41]. In addition to simulation, AgentFly was implemented on a real fixed-wing Procerus UAV [32].

The basis of this use-case is the set of missions selected for the AgentFly project. It is here extended to include an adversarial cyber-attack activity against the AgentFly UAV to disrupt its mission. The use case is that a swarm of AgentFly

UAVs perform a routine tactical aerial surveillance mission in an urban area. Collaboration between AgentFly UAVs aims at collision avoidance, trajectory planning, automatic distributed load-balancing and mission assurance.

The AgentFly UAVs use case is built around the following assumptions:

- AgentFly UAVs self-assess and share information with neighboring UAVs.
- When setting up a communication channel, AgentFly UAVs have to determine whether they trust their correspondent.
- Network-wide collaboration and negotiation is affected by timing, range, and reachability issues.
- The AgentFly UAV lacks modern cyber defense capabilities and is thus vulnerable to potential cyberattacks.
- Due to environmental conditions, AgentFly UAVs might be offline for some time and later re-join the swarm when connectivity allows.
- A single AICA agent is implemented within each AgentFly UAV.
- The AICA connects with the modules of the UAV and can supervise the activity and signals in and between various UAV modules (e.g., sensors, navigation, and actuators).
- The AICA can function in isolation from other AgentFly UAVs' AICA agents present in the AgentFly UAV swarm.

Attackers have acquired a technology similar to that used in AgentFly UAVs' COMMS module. They have discovered a zero-day vulnerability that can be exploited remotely over the radio link from the ground and they plan to use the vulnerability in order to gain control over the swarm of UAVs and cut them off from the theatre's Command & Control (C2) system. The UAVs are using the COMMS module to collaborate among themselves and report to the C2 when needed.

The vulnerability lies in the functionality responsible for dynamically registering new UAV agents in the swarm upon due request. The COMMS module is interconnected with other intrinsic modules of the AgentFly UAV via a central control unit.

The adversary has set up a ground station in the area of the surveillance mission. When AgentFly UAVs enter the area, the cyberattack is launched.

The AICA detects a connection to the COMMS module and allows the incoming connection for the dynamic registration of a new UAV agent into the swarm. Due to the nature of zero-day attacks, an Intrusion Detection System (IDS) would not have any corresponding signatures to detect a compromised payload.

The AICA's Sensor monitors the entire set of modules of the AgentFly UAV.

The AICA's World-state identifier module flags the connection from a newly connected UAV agent as anomalous since it does not follow the baseline pattern that has been established out of previous connections with legitimate UAVs. It also detects a change in the UAV's system configuration and deems it anomalous because no new configurations have been received from the C2. The AICA launches, through its Sensor module, a system integrity check. A compromise within the UAV's COMMS module is detected.

The AICA decides (Planner-Selector and Action selection modules) to isolate (Action execution module) the COMMS module from other UAV modules in order

to prevent further propagation. Alerting the C2 is not possible because of the compromised COMMS module.

In order to reduce the attack surface, the AICA requests (Action execution module) that the UAV's central control unit resets the COMMS module, raises the security level and disables auxiliary functions (among others, the dynamic inclusion of new UAVs into the swarm).

The AICA performs another integrity check to verify that no other compromise exists. It keeps its Sensor and World-state identifier modules on a high-level of vigilance in relation to integrity monitoring. The AICA adds the signature of the payload that caused the anomaly into its knowledge base. And it sends out an alert along with malware signature updates to other agents as well as to the C2.

This basic, single AICA agent, use case should be expanded to Multi AICA agents deployed across the AgentFly UAV's architecture and modules. Future research will benchmark Multi AICA agents versus Single AICA agent deployments in order to assess the superiority and context of Multi AICA agent solutions.

4 Discussion and Future Research Directions

The AICA Reference Architecture (AICARA) [22] was elaborated on the basis of [37, 38].

Since the end of 70's and the early works on Artificial Intelligence (AI), the concept of agent was used by different authors to represent different ideas. This polymorphic concept was synthesized by authors such as [30, 48]. Since 1995, Russell and Norvig [38] proposed an architecture and functional decomposition of agents widely regarded as reference work in the ever-growing field of AI.

Their agent architecture can be seen as an extension of the developments in object-oriented methods for software development that culminated in the Unified Modeling Language [4] and design patterns [17]. Both concepts form the basis of modern software development.

The concept of cooperating cognitive agents [38] perfectly matches requirements for AICA agents.

First, AICA agents need to prove trustworthy, and therefore the AICA Reference Architecture is conceived as a white-box. The agent's architecture involves a set of clearly defined modules and specifies the links connecting information perception to action actuation or else the agent to external agents or a central cyber defense C2.

Second, the AICA agents must go beyond merely reactive agents because in situations of autonomy they will need to make decisions by themselves. Reactive agents are today widely used in cybersecurity and are based on rule sets in the form of "if X suspicious, then trigger Y".

Third, Russell and Norvig [38] has attributes highly required by AICA agents: autonomous decision making, learning and cooperation. This is important because these agents may operate for prolonged periods of time if deployed in autonomous weapon systems. The latter may face multiple and unknown cyber-attacks and AICA

agents, by learning and cooperating with one another, will sustain their capacity to equip the weapon system with an autonomous intelligent cyber defense.

Applied to the field of the autonomous cyber defense of military systems [38], well-known concepts must be reassessed, prototypes must be built and tested, and the superiority of the concept must now be benchmarked.

Developing the concepts described here also presents many other challenges that require research in the coming years.

Agents' integrity, agent communications' security, the inclusion of cyber defense techniques such as deception, or else identifying and selecting the right actions, are only a few of them.

4.1 Agents' Integrity

A compromise of agents can potentially threaten the entire military platform they are supposed to defend. It is paramount to harden the agents' architecture in order to minimize the chance of such compromise. Methods that assess the integrity of the agent during runtime are required.

Virtualization techniques have been successfully employed to improve systems' resiliency [2, 18]. For instance, systems such as [18] allow providing security guarantees to applications running on untrusted operating systems. It is possible to build upon such techniques in order to harden AICA agents and to maintain their functionality even under attack or in case of partial compromise. Furthermore, periodical assessment of agents' integrity can be performed through attestation techniques [15], based on a trusted hardware core (Trusted Platform Module, TPM). Such techniques allow ensuring that the software of the agent has not been altered at any time, even during the operations of the platform, and can easily scale up to millions of devices [1]. Finally, while the topic of protecting machine learning systems from adversarial examples is still relatively new, techniques such as distillation [31] could be leveraged to increase robustness.

4.2 Agent Communications' Security

Sensors are the fundamental building blocks providing the agents with a consistent world view. As such, they are a part of the AICA architecture most exposed to adversarial tampering. The AICA architecture needs to provide secure communications to ensure that the agent's world view is not corrupted.

To this end, cryptographic protocols such as random key pre-distribution [12, 13], can be employed to provide secure agent-sensor communication even when one or more sensor channels are compromised.

4.3 The Inclusion of Cyber Defense Techniques Such as Deception

Deception is a key component of active defense systems and, consequently, could be part of the AICA architecture. Active defense deception tools can be used to thwart an ongoing attack. To provide this functionality, the AICA architecture can employ deception techniques such as honeyfiles [6, 49], mock sensors [14] and fake services [34]. Moreover, implementing dynamic tools deployment and reconfiguration is required for actuating functions. To this end container technologies can be employed, such as in [9] to provide isolation and configuration flexibility.

4.4 Identifying and Selecting the Right Actions

Identifying the appropriate actions to take in response to external stimuli is one of the key requirements for the AICA architecture. The AICA agent should include autonomous decision making that can adapt to the current world state. Machine learning-based techniques can be employed [39] to this end, to devise complex plans of action [40] to mitigate an attack, and to learn from previous experiences. However, Blakely and Theron [3] have shown that a variety of techniques may be called upon by AICA agents to elaborate their decisions.

5 In Conclusion

AICA agents are required by foreseeable evolutions of military systems, and it is likely that civil systems, such as the wide-scale deployment of the Internet of Things, will generate similar demands.

The AICA Reference Architecture (AICARA) [22] is a seminal proposition to answer the needs and challenges of the situation.

NATO's IST-152 Research and Technology Group (RTG) has initiated this piece of work and in a recent meeting held in Warsaw, Poland, has evaluated that future research is likely to span over the next decade before efficient solutions be operated.

The AICARA opens discussions among the scientific community, from computer science to cognitive science, Law and moral philosophy.

Autonomous intelligent cyber defense agents may change the face of the fight against malware. This is our assumption.

References

1. Ambrosin, M. et al., 2016. *SANA: Secure and Scalable Aggregate Network Attestation.* New York, NY, USA, ACM, pp. 731–742.
2. Baumann, A., Peinado, M. & Hunt, G., 2015. Shielding Applications from an Untrusted Cloud with Haven. *ACM Trans. Comput. Syst.,* 8, Volume 33, pp. 8:1–8:26.
3. Blakely, B. & Theron, P., 2018. *Decision flow-based Agent Action Planning.* Prague, 18–20 October 2017: https://export.arxiv.org/pdf/1804.07646.
4. Booch, G., 1991. *Object-Oriented Analysis and Design with Applications.* The Benjamin Cummings Publishing Company ed. San Francisco, CA: Pearson Education.
5. Boulanin, V. & Verbruggen, M., 2017. *Mapping the development of autonomy in weapon systems,* Solna, Sweden, available at https://www.sipri.org/publications/2017/other-publications/mapping-development-autonomy-weapon-systems: SIPRI.
6. Bowen, B. M., Hershkop, S., Keromytis, A. D. & Stolfo, S. J., 2009. *Baiting Inside Attackers Using Decoy Documents.* s.l., Springer, Berlin, Heidelberg, pp. 51–70.
7. Carrasco, A. et al., 2010. Multi-agent and embedded system technologies applied to improve the management of power systems. *JDCTA,* 4(1), pp. 79–85.
8. Chen, B. & Cheng, H. H., 2010. A review of the applications of agent technology in traffic and transportation systems. *Trans. Intell. Transport. Sys.,* 11(2), pp. 485–497.
9. De Gaspari, F., Jajodia, S., Mancini, L. V. & Panico, A., 2016. *AHEAD: A New Architecture for Active Defense,* Vienna, Austria: SafeConfig'16, October 24 2016.
10. Defense Science Board, 2012. *Task Force Report: The Role of Autonomy in DoD Systems,* Washington, D.C.: Office of the Under Secretary of Defense for Acquisition, Technology and Logistics.
11. Defense Science Board, 2016. *Summer Study on Autonomy,* Washington, D.C.: Office of the Under Secretary of Defense for Acquisition, Technology and Logistics.
12. Di Pietro, R., Mancini, L. V. & Mei, A., 2003. *Random Key-assignment for Secure Wireless Sensor Networks.* New York, NY, USA, ACM, pp. 62–71.
13. Di Pietro, R., Mancini, L. V. & Mei, A., 2006. Energy Efficient Node-to-node Authentication and Communication Confidentiality in Wireless Sensor Networks. *Wireless Networks,* 11, Volume 12, pp. 709–721.
14. Disso, J. P., Jones, K. & Bailey, S., 2013. *A Plausible Solution to SCADA Security Honeypot Systems.* IEEE, Eighth International Conference on Broadband, Wireless Computing, Communication and Applications, pp. 443–448.
15. Eldefrawy, K., Francillon, A., Perito, D. & Tsudik, G., 2012. *SMART: Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust.* 19th Annual Network and Distributed System Security Symposium, February 5–8 ed. San Diego, CA: NDSS 2012.
16. Elmenreich, W., 2003. Intelligent methods for embedded systems. In: J. 2. Vienna University of Technology 2003, ed. *Proceedings of the First Workshop on Intelligent Solutions in Embedded Systems.* Austria: Vienna: Vienna University of Technology, pp. 3–11.
17. Gamma, E., Helm, R., Johnson, R. & Vlissides, J., 1994. *Design patterns: elements of reusable object-oriented software.* Reading, Massachusetts: Addison-Wesley.
18. Hofmann, O. S. et al., 2013. *InkTag: Secure Applications on an Untrusted Operating System.* New York, NY, USA, ACM, pp. 265–278.
19. Huang, H.-P., Liang, C.-C. & Lin, C.-W., 2001. Construction and soccer dynamics analysis for an integrated multi-agent soccer robot system. *Natl. Sci. Counc. ROC(A),* Volume 25, pp. 84–93.
20. Jamont, J.-P. & Occello, M., 2011. A framework to simulate and support the design of distributed automation and decentralized control systems: Application to control of indoor building comfort. In: *IEEE Symposium on Computational Intelligence in Control and Automation.* Paris, France: IEEE, pp. 80–87.
21. Jamont, J.-P., Occello, M. & Lagrèze, A., 2010. A multiagent approach to manage communication in wireless instrumentation systems. *Measurement,* 43(4), pp. 489–503.

22. Kott, A. et al., 2019. *Autonomous Intelligent Cyber-defense Agent (AICA) Reference Architecture, Release 2.0*, Adelphi, MD: US Army Research Laboratory, ARL SR-0421, September 2019, available from https://arxiv.org/abs/1803.10664.
23. LeBlanc, B., Losiewicz, P. & Hourlier, S., 2017. *A Program for effective and secure operations by Autonomous Agents and Human Operators in communications constrained tactical environments.* Prague: NATO IST-152 workshop.
24. Lin, J. & Singer, P. W., 2014. *University Tests Long-Range Unmanned Mini Sub.* [Online] Available at: https://www.popsci.com/blog-network/eastern-arsenal/not-shark-robot-chinese-university-tests-long-range-unmanned-mini-sub [Accessed 11 May 2018].
25. McArthur, S. D. et al., 2007. Multi-Agent Systems for Power Engineering Applications - Part I: Concepts, Approaches, and Technical Challenges. *IEEE TRANSACTIONS ON POWER SYSTEMS,* 22(4), pp. 1743–1752.
26. Muller, J.-P., 2004. Emergence of collective behaviour and problem solving. In: A. Omicini, P. Petta & J. Pitt, eds. *Engineering Societies in the Agents World IV.* volume 3071: Lecture Notes in Computer Science, pp. 1–20.
27. NAP, 2012. *Intelligent Human-Machine Collaboration: Summary of a Workshop,* available at http://nap.edu/13479: National Academies Press.
28. NAP, 2014. *Autonomy Research for Civil Aviation: Toward a New Era of Flight,* available at http://nap.edu/18815: National Academies Press.
29. NAP, 2016. *Protection of Transportation Infrastructure from Cyber Attacks: A Primer,* Available at http://nap.edu/23516: National Academies Press.
30. Nwana, H. S., 1996. Software agents: An overview. *The knowledge engineering review, 11(3),* pp. 205–244.
31. Papernot, N. et al., 2016. *Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks.* IEEE, 37th IEEE Symposium on Security & Privacy, pp. 582–597.
32. Pěchouček, M., Jakob, M. & Novák, P., 2010. Towards Simulation-Aided Design of Multi-Agent Systems. In: R. Collier, J. Dix & P. Novák, eds. *Programming Multi-Agent Systems.* Toronto, ON, Canada: Springer, 8th InternationalWorkshop, ProMAS 2010, 11 May 2010, Revised Selected Papers, pp. 3–21.
33. Pechoucek, M. & Marík, V., 2008. Industrial deployment of multi-agent technologies: review and selected case studies. *Autonomous Agents and Multi-Agent Systems,* Volume 17, p. 397–431.
34. Provos, N., 2004. *A Virtual Honeypot Framework.* Berkeley, USENIX Association, pp. 1–1.
35. Rasch, R., Kott, A. & Forbus, K. D., 2002. AI on the battlefield: An experimental exploration. *AAAI/IAAI.*
36. Rasch, R., Kott, A. & Forbus, K. D., 2003. Incorporating AI into military decision making: an experiment. *IEEE Intelligent Systems,* 18(4), pp. 18–26.
37. Russell, S. J. & Norvig, P., 2003. *Artificial Intelligence: A Modern Approach.* 2nd ed. Upper Saddle River, New Jersey: Prentice Hall.
38. Russell, S. J. & Norvig, P., 2010. *Artificial Intelligence: a Modern Approach.* 3rd ed. Upper Saddle River, NJ: Pearson Education.
39. Seufert, S. & O'Brien, D., 2007. *Machine Learning for Automatic Defence Against Distributed Denial of Service Attacks.* IEEE, ICC 2007 proceedings, pp. 1217–1222.
40. Silver, D. et al., 2017. Mastering the game of Go without human knowledge. *Nature,* 10, Volume 550, p. 354.
41. Sislak, D., Volf, P., Kopriva, S. & Pěchouček, M., 2012. AgentFly: Scalable, High-Fidelity Framework for Simulation, Planning and Collision Avoidance of Multiple UAVs. In: P. Angelov, ed. *Sense and Avoid in UAS: Research and Applications.* Wiley Online Library: Wiley: John Wiley&Sons, Inc., https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119964049.ch9, pp. 235-264.
42. Snyder, D. et al., 2015. *Improving the Cybersecurity of U.S. Air Force Military Systems Throughout Their Life Cycles,* Santa Monica, CA: RAND Corporation.

43. Stytz, M. R., Lichtblau, D. E. & Banks, S. B., 2005. *Toward using intelligent agents to detect, assess, and counter cyberattacks in a network-centric environment,* Alexandria, VA: Institute For Defense Analyses.
44. Tactical AGENTFLY, 2018. *Agent Technology Center.* [Online] Available at: http://agents.felk.cvut.cz/projects/agentfly/tactical [Accessed 6 June 2018].
45. Théron, P., 2017. *La cyber résilience, un projet cohérent transversal à nos trois thèmes, et la problématique particulière des Systèmes Multi Agent de Cyber Défense.* Leçon inaugurale, 5 décembre 2017, ed. Salon de Provence, France: Chaire Cyber Résilience Aérospatiale (Cyb'Air).
46. Von Neumann, J., 1951. The General and Logical Theory of Automata. In: L. A. Jeffress, ed. *Cerebral Mechanisms in Behavior: The Hixon Symposium, September 1948, Pasadena.* New York: John Wiley & Sons, Inc, pp. 1–31.
47. Weick, K., 1993. The Collapse of Sensemaking in Organizations: The Mann Gulch Disaster. *Administrative Science Quarterly,* 38(4), pp. 628–652.
48. Wooldridge, M. & Jennings, N. R., 1995. Intelligent agents: Theory and practice. *The knowledge engineering review, 10(2),* pp. 115–152.
49. Yuill, J., Zappe, M., Denning, D. & Feer, F., 2004. *Honeyfiles: deceptive files for intrusion detection.* IEEE Xplore, Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC, 10–11 June 2004, pp. 116–122.

Appendix 9

Publication IX
A. Kott, P. Théron, L. V. Mancini, E. Dushku, A. Panico, M. Drašar, B. LeBlanc, P. Losiewicz, A. Guarino, M. Pihelgas, and K. Rzadca. An introductory preview of Autonomous Intelligent Cyber-defense Agent reference architecture, release 2.0. *The Journal of Defense Modeling and Simulation*, 17(1):51–54, 2020

The article is included in the *Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*.
DOI: 10.1177/1548512919886163

Technical Note

JDMS

Journal of Defense Modeling and Simulation: Applications, Methodology, Technology
2020, Vol. 17(1) 51–54

DOI: 10.1177/1548512919886163
journals.sagepub.com/home/dms
SAGE

An introductory preview of Autonomous Intelligent Cyber-defense Agent reference architecture, release 2.0

Alexander Kott[1], Paul Théron[2], Luigi V Mancini[3], Edlira Dushku[3], Agostino Panico[3], Martin Drašar[4], Benoît LeBlanc[5], Paul Losiewicz[6], Alessandro Guarino[7], Mauno Pihelgas[8], and Krzysztof Rzadca[9]

Abstract

The North Atlantic Treaty Organization (NATO) Research Task Group IST-152 developed a concept and a reference architecture for intelligent software agents performing active, largely autonomous cyber-defense actions on military assets. The group released a detailed report, briefly reviewed in this article, where such an agent is referred to as an Autonomous Intelligent Cyber-defense Agent (AICA).

In a conflict with a technically sophisticated adversary, NATO military networks will operate in a heavily contested battlefield. Enemy malware will likely infiltrate and attack friendly networks and systems. Today's reliance on human cyber defenders will be untenable on the future battlefield. Instead, artificially intelligent agents, such as AICAs, will be necessary to defeat the enemy malware in an environment of potentially disrupted communications where human intervention may not be possible.

The IST-152 group identified specific capabilities of AICA. For example, AICA will have to be capable of autonomous planning and execution of complex multi-step activities for defeating or degrading sophisticated adversary malware, with the anticipation and minimization of resulting side effects. It will have to be capable of adversarial reasoning to battle against a thinking, adaptive malware. Crucially, AICA will have to keep itself and its actions as undetectable as possible, and will have to use deceptions and camouflage.

The report identifies the key functions and components and their interactions for a potential reference architecture of such an agent, as well as a tentative roadmap toward the capabilities of AICA.

Keywords

Intelligent agent, autonomy, cyber warfare, cyber defense, agent architecture

1. Background and introduction

To focus the attention of our research group, we have chosen to limit the scope of the problem as follows. We consider a single military platform, such as a vehicle, a vessel, or an unmanned aerial vehicle (UAV), with one or more computers residing on the platform, connected to sensors and actuators. Each computer contributes considerably to the operation of the platform or systems installed on the platform. One or more computers are assumed to have been compromised, where the compromise is either established as a fact or is suspected.

[1]US Army CCDC Army Research Laboratory, USA
[2]Thales, France
[3]Sapienza Università di Roma, Italy
[4]Masaryk University, Czech Republic
[5]Ecole Nationale Supérieure de Cognitique, France
[6]Cybersecurity and Information Systems IAC, USA
[7]StAG Srl, Italy
[8]NATO Cooperative Cyber-defense Centre of Excellence, Estonia
[9]Institute of Informatics, University of Warsaw, Poland

Corresponding author:
Alexander Kott, US Army CCDC Army Research Laboratory, 2800 Powder Mill Road, Adelphi, MD 20783, USA.
Email: alexander.kott1.civ@mail.mil

Due to the contested nature of the communications environment (e.g., the enemy is jamming the communications or radio silence is required to avoid detection by the enemy), communications between the vehicle and other elements of the friendly force are often limited and intermittent. At certain times and under some conditions, communications may be entirely impossible.[1]

Given the constraints on communications, conventional centralized cyber defense (i.e., an architecture where local sensors send cyber-relevant information to a central location where highly capable cyber-defense systems and human analysts detect the presence of malware and initiate corrective actions remotely) is often infeasible.[2] It is also unrealistic to expect that human warfighters residing on the platform, for example, a vehicle, will have the necessary skills or time available to perform cyber-defense functions locally on the vehicle, even more so if the vehicle is unmanned.[3]

Therefore, the cyber defense of such a platform, including its computing devices, will have to be performed by an intelligent, autonomous software agent.[4] The agent (or multiple agents per platform) will stealthily monitor the networks, detect the enemy agents while remaining concealed, and then destroy or degrade the enemy malware.[5] The agent will have to do so mostly autonomously, without the support of or guidance by a human expert.

In most discussions in the architecture document, the agent is considered as a monolithic piece of software. However, depending on the implementation, the agent's modules can be distributed over multiple processes or devices, or it could be implemented as a team of agents or subagents.

To fight the enemy malware that has infiltrated the friendly computer, the agent may have to take destructive actions, such as deleting or quarantining certain software. Such destructive actions are carefully controlled by the appropriate rules of engagement and are allowed only on the computer where the agent resides.

In most cases, the agent will not be able to stop the enemy from penetrating the platform's systems. However, it will be able to perform the detection of, analysis of, and response to a given threat. The actions of the agent, in general, cannot be guaranteed to preserve the availability or integrity of the functions and data of friendly computers. There is a risk that an action of the agent will ''break'' the friendly computer, disable important friendly software, or corrupt or delete important data. Developers of the agent will attempt to design its actions and planning capability to minimize the risk. This risk, in a military environment, has to be balanced against the death or destruction caused by the enemy if the agent's action is not taken.

Provisions will be made to enable a remote controller—a human or automated cyber command and control (C2) node—to fully observe, direct, and modify the actions of the agent, and even to update the agent's software as needed. However, it is recognized that such a remote control is often impossible due to the difficulties of communicating between the agent and the control node.[6] The agent, therefore, should be able to plan, analyze, and perform most or all of its actions autonomously.

Similarly, provisions should be made for the agent to collaborate with other agents (that reside on other computers); however, in many cases, because the communications are impaired or observed by the enemy, the agent has to eschew collaboration and operate alone.

The enemy malware, specifically, its capabilities and tactics, techniques, and procedures (TTPs), evolves rapidly. Therefore, the agent will be capable of autonomous learning. In the case that enemy malware knows that the agent exists and is likely to be present on the computer, the enemy malware will seek to find and destroy the agent. Therefore, the agent will possess techniques and mechanisms for maintaining a certain degree of stealth, camouflage, and concealment.[7] More generally, the agent takes measures that reduce the probability that the enemy malware will detect the agent. The agent is mindful of the need to exercise self-preservation and self-defense.

It is assumed here that the agent resides on a computer where it was originally installed by a human controller or an authorized process. We do envision a possibility that an agent may move itself (or move a replica of itself) to another computer. However, such propagation is assumed to occur only under exceptional and well-specified conditions, and takes place only within a friendly network—from one friendly computer to another friendly computer.

2. High-level architecture of the Autonomous Intelligent Cyber-defense Agent

Our initial exploration identified the key functions, components, and their interactions for a potential reference architecture (see Figure 1) of such an agent.[8] To mention just a few examples, *Sensing and World State Identification* is the Autonomous Intelligent Cyber-defense Agent (AICA) high-level decision-making function that allows a cyber-defense agent to acquire data from the environment and systems in which it operates, as well as from itself, to reach an understanding of the current state of the world. *Planning and Action Selection* is the AICA high-level decision-making function that allows a cyber-defense agent to elaborate one to several action proposals (*Planning*) and propose them to the *Action Selection* function that decides the action or set of actions to execute.[9] *Learning* is the AICA high-level function that allows a cyber-defense agent to use the agent's experience to improve progressively its efficiency with regard to all other functions. For these and other high-level functions of

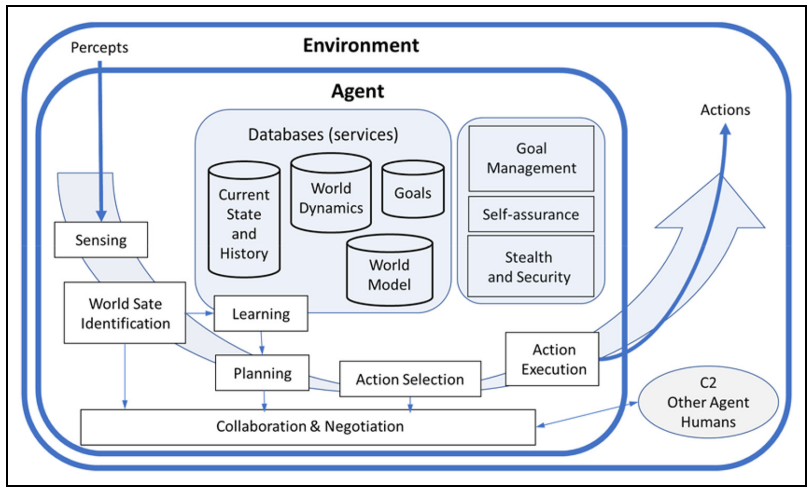

Figure 1. Autonomous Intelligent Cyber-defense Agent reference architecture—the key components of the agent. C2: command and control.

AICA, our initial analysis suggests that the required technical approaches do not seem to be far beyond the current state of research.[7,10]

3. Roadmap for Autonomous Intelligent Cyber-defense Agent development

Based on the analysis of the proposed AICA reference architecture and available technological foundation, we envision a roadmap toward initial yet viable capabilities. The first phase of the roadmap, which could last perhaps of the order of 2 years, will include the development of knowledge-based planning of actions, the execution functionality, elements of resilient operations under attack, and adaptation of the prototype agent for the execution of a small computing device. This phase would culminate in a series of Turing-like experiments that would evaluate the capability of the agent to produce plans for remediating a compromise, as compared to experienced human cyber defenders.

The second phase, which could last about 3 years, would focus on adaptive learning, the development of a structured world model, and mechanisms for dealing with explicitly defined, multiple, and potentially conflicting goals. At this stage, the prototype agent should demonstrate the capability, in a few self-learning attempts, to return the defended system to acceptable performance after a significant change in the adversary malware behavior or techniques and procedures.

The third phase, potentially about 3–4 years, would delve into issues of multiagent collaboration, human interactions, and ensuring both the stealth and trustworthiness of the agent. Cyberã physical challenges may need to be addressed as well. This phase would be completed when the prototype agents are able to successfully resolve a cyber compromise that could not be handled by any individual agent.

4. Conclusions

The report describing the AICA reference architecture has been released.[11] North Atlantic Treaty Organization (NATO) cyber defense would benefit from active encouragement of AICA development efforts. Relevant research in academia and in some government and industry research organizations is growing, and should be supported. It appears that academic institutions have already begun work toward AICA-like capabilities, and results are beginning to be available for transition to industry. NATO defense agencies should query the cybersecurity software vendors about availability of AICA-like products. Creating a multi-stakeholder working group engaging industry, academia, and governments could help facilitate the development of AICA technologies. NATO must not fall behind its adversaries in developing and deploying such capabilities.

Funding

The authors received no financial support for the research, authorship, and/or publication of this article.

References

1. Kott A, Swami A and West B. The Internet of battle things. *Computer* 2016; 49: 70–75.
2. LeBlanc B, Losiewicz P and Hourlier S. A program for effective and secure operations by autonomous agents and human operators in communications constrained tactical environments. In: Kott A, Thomas R, Dražar M, Kont M, Poylisher A, Blakely B, Theron P, Evans N, Leslie N, Singh R, Rigaki M (eds) *Toward Intelligent Autonomous Agents for Cyber Defense: Report of the 2017 Workshop by the North Atlantic Treaty Organization (NATO) Research Group IST-152-RTG*, Prague, 18-20 October 2017, pp.17–21. Aldephi, MD: US Army Research Laboratory.
3. Kott A and Alberts DS. How do you command an army of intelligent things? *Computer* 2017; 50: 96–100.
4. Guarino A. Autonomous intelligent agents in cyber offence. In: *proceedings of the 5th international conference on cyber conflict* (eds Podins K, Stinissen J and Maybaum M), Tallinn, Estonia, 4–7 June 2013, pp.1–12. Piscataway, NJ: IEEE.
5. Stytz MR, Lichtblau DE and Banks SB. *Toward using intelligent agents to detect, assess, and counter cyber-attacks in a network-centric environment*. Alexandria (VA): Institute for Defense Analyses, 2005.
6. Kott A (ed.) *Advanced technology concepts for command and control*. Bloomington, IN: Xlibris Corporation, 2004.
7. Al-Shaer E, Wei J, Hamlen KW, et al. Towards intelligent cyber deception systems. In: Al-Shaer E, Wei J, Hamlen KW and Wang C (eds) *Autonomous cyber deception: reasoning, adaptive planning, and evaluation of honeythings*. New York: Springer, 2019.
8. Theron P, Kott A, Dražar M, et al. Towards an active, autonomous and intelligent cyber defense of military systems: the NATO AICA reference architecture. In: *2018 international conference on military communications and information systems (ICMCIS)*, 22 May 2018, pp.1–9. Piscataway, NJ: IEEE.
9. Kott A, Ground L, Budd R, et al. Toward practical knowledge-based tools for battle planning and scheduling. In: Dechter R and Sutton R (eds) *AAAI/IAAI*, Edmonton, AL, 28 July 2002, pp.894–899. Menlo Park, CA: AAAI.
10. De Gaspari F, Jajodia S, Mancini LV, et al. AHEAD: a new architecture for active defense. In: Multari NJ, Singhal A and Manz DO (eds) *proceedings of the 2016 ACM workshop on automated decision making for active cyber defense (SafeConfig)*, 24 October 2016, pp.11–16. New York: ACM.
11. Kott A, Théron P, Dražar M, et al. *Autonomous Intelligent Cyber-defense Agent (AICA) Reference Architecture*. Release 2.0, Report ARL-SR-0421, US Army Research Laboratory, Adelphi, MD, September 2019.

Author biographies

Alexander Kott is the Chief Scientist of the US Army Combat Capabilities Development Command's Army Research Laboratory (ARL) in Adelphi, Maryland. Earlier, in 2003–2008, he served as a Program Manager at the Defense Advanced Research Programs Agency (DARPA).

Paul Théron works for Thales. He is the director of the French Air Force's ''Aerospace Cyber Resilience'' (Cyb'Air) research chair where he drives research on autonomous cyber defense.

Luigi V Mancini is a full professor of Computer Science at the University of Rome ''La Sapienza'', Italy, and the director of the Master degree program in Cybersecurity.

Edlira Dushku is a PhD student at Sapienza University of Rome, Italy. She works in Computer Security research group at the Department of Informatics.

Agostino Panico is a professional penetration tester, a SANS Mentor, and a PhD student at the Sapienza University of Rome, Italy, where his research focuses on penetration testing and incident handling.

Martin Drasar, PhD, is a senior researcher and the head of Proactive Security Group at Masaryk University.

Benoit LeBlanc is a professor in artificial intelligence and cognitive sciences at the National Polytechnic Institute of Bordeaux (Bordeaux INP), France, and a director of the National Engineering School of Cognitive Sciences (ENSC, Bordeaux INP).

Paul B Losiewicz is a Senior Scientific Advisor for the Cybersecurity and Information Systems Information Analysis Center (CSIAC), a US Department of Defense information analysis center. He has over 30 years of defense research and technology experience.

Alessandro Guarino is an independent cybersecurity and cyber conflict researcher, as well as the founder and CEO of StAG, a cybersecurity services company. He is active in several international standardization bodies.

Mauno Pihelgas is a researcher in the Technology Branch of the NATO Cooperative Cyber Defence Centre of Excellence (CCDCOE), where his main area of expertise is security monitoring and data mining.

Krzysztof Rzadca, PhD, is an associate professor of computer science at the Institute of Computer Science, University of Warsaw, Poland.

Appendix 10

Publication X
R. Vaarandi and M. Pihelgas. NetFlow Based Framework for Identifying Anomalous End User Nodes. In *15th International Conference on Cyber Warfare and Security (ICCWS 2020)*, page 448–456, 2020

The paper is included in the *Proceedings of the 15th International Conference on Cyber Warfare and Security (ICCWS 2020)*. ISBN: 978-1-912764-52-5.
DOI: 10.34190/ICCWS.20.035

NetFlow Based Framework for Identifying Anomalous End User Nodes

Risto Vaarandi[1] and Mauno Pihelgas[1,2]
[1]Centre for Digital Forensics and Cyber Security, Tallinn University of Technology, Estonia
[2]Technology Branch, NATO CCDCOE, Estonia
firstname.lastname@taltech.ee

Abstract: During the last two decades, cyber attacks against end users have grown significantly both in terms of number and sophistication. Unfortunately, traditional signature-based technologies such as network IDS/IPS and next generation firewalls are able to detect known attacks only, while new attack types not matching any signatures remain unnoticed. Therefore, the use of machine learning for detecting anomalous network traffic of end user nodes has become an important research problem. In this paper, we present a novel NetFlow based framework for identifying anomalous end user nodes and their network traffic patterns, and describe experiments for evaluating framework performance in an organizational network.

Keywords: detection of anomalous end user nodes, network anomaly detection, NetFlow based network monitoring

1. Introduction

For protecting workstations and laptops in corporate networks, security-aware organizations are employing specialized technologies like gateways for filtering web and e-mail traffic. Also, for lessening the risk of infection, end user nodes are often centrally managed, with unmanaged computers being blocked from connecting to corporate network. Unfortunately, many smaller organizations are using simple NAT firewalls that do not support any filtering of malicious application layer traffic. According to recent report by Symantec (2019), employees of smaller institutions are more likely to be hit by e-mail threats. Also, many organizations have no centralized patching routines and have adopted bring-your-own-device policy which makes their end user nodes much more vulnerable. This problem is exacerbated by insufficient cyber security awareness and lack of relevant training (SANS, 2018).

After an end user node has been infected, attackers can harness it for various purposes like compromising other devices in private network, launching attacks against other organizations, etc. For detecting such activities, network IDS is often used. Since most network IDS are signature based and thus able to identify previously known malicious traffic patterns only, a number of NetFlow based anomaly detection algorithms have been suggested in recent papers. However, apart from few exceptions (e.g., (Grill et al, 2015)), most previously suggested methods have not focused on detection of anomalous end user nodes in organizational networks. Furthermore, many methods can only raise an alarm about anomaly for some node or network segment without the ability to highlight malicious traffic patterns. Nevertheless, providing such information to network administrators would significantly reduce incident resolution time (Zhou et al, 2015). Finally, only few works have considered the use of multiple classifiers for anomaly detection from NetFlow data (Hou et al, 2018).

This paper addresses above research gaps and presents an unsupervised framework for detecting anomalous end user nodes in organizational networks. The framework employs several anomaly detectors for finding a total anomaly score for each node in hourly time windows, and uses LogCluster algorithm for finding network traffic patterns for anomalous nodes from NetFlow data. We have evaluated the framework during 5 months in a network of an academic institution. The remainder of this paper is organized as follows – section 2 discusses related work, section 3 presents our framework, section 4 describes performance evaluation of the framework, and section 5 outlines future work.

2. Related Work

Cisco NetFlow is a protocol which defines *flow* as a sequence of packets that share a number of common properties, most notably the source IP address, source port, destination IP address, destination port, and transport protocol. For monitoring network traffic, NetFlow *exporter* maintains a record for each flow in a

memory-based cache, with the record holding counters for packets and bytes in the flow, the union of all observed TCP flags, start and end time of the flow, and other data. NetFlow exporter sends a flow record to NetFlow *collector* when some flow-based timer expires (e.g., no packets have been seen during 60 seconds), when the flow ends (e.g., TCP connection is closed), or when flow cache becomes full. Many modern network devices can act as NetFlow exporters, and there are many commercial and open-source NetFlow exporter and collector implementations (Hofstede et al, 2014). Also, a number of approaches have been suggested for anomaly detection from NetFlow data.

Brauckhoff et al (2012) have proposed an algorithm where histograms are built from flow features (e.g., source port) during measurement intervals, where each histogram represents a flow feature distribution. At the end of each interval, Kullback-Leibler distance between distributions for current and previous interval is calculated. If the distance exceeds a threshold, alarm is raised and Apriori frequent itemset mining algorithm is used for mining patterns from suspicious flow records. Kind, Stoecklin and Dimitropoulos (2009) have described a supervised method where the training phase involves building histograms and clustering similar histograms together, in order to create models of normal behavior. For anomaly detection, a vector is computed that encodes online network behavior, and a distance of the vector from clusters is calculated.

Grill et al (2015) have suggested a method for discovering hosts with domain generation algorithm (DGA) based malware by measuring the ratio of DNS requests to the number of unique IP addresses contacted by the host. According to experiments, the ratio is high for infected hosts. Our past work (Vaarandi, 2013) proposes two unsupervised anomaly detection algorithms for organizational private networks. The first algorithm maintains behavior profile for each node which describes recently used services, and raises an alarm if the node connects to unusual service. The second algorithm clusters nodes on daily basis, in order to find node groups that consistently use the same services, and raises an alarm if node behavior deviates from the rest of the group. Muhs et al (2018) have developed a method for detecting P2P botnets which creates a communication graph from NetFlow data. Each graph node represents a host and each graph edge a probability of communication between relevant hosts. The method conducts a large number of random walks (traversals over k nodes), and calculates the probability of reaching each end node. Resulting probability distribution is then clustered with Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm (Ester et al, 1996), and according to authors, P2P bots form dense clusters.

Hofstede et al (2013) have proposed exponentially weighted moving average (EWMA) based algorithm for DDoS detection which tracks the number of flows. If unexpected change in the number of flows is detected, the algorithm can create firewall rules for blocking malicious traffic. Hou et al (2018) have suggested random forest classifiers for DDoS detection, and have found their performance superior to C4.5, SVM, and Adaboost classifiers. Paredes-Oliva et al (2012) have proposed a supervised algorithm which first identifies traffic patterns with FPmax frequent itemset mining algorithm, and then uses C5.0 classifier for finding anomalous patterns. Finally, Zhou et al (2015) have developed ENTvis tool which divides NetFlow data into timeframes, and for each timeframe calculates entropies for source IP, source port, destination IP and destination port. This information is then visualized with several techniques (e.g., visual clustering) which allows human analysts to spot anomalies and understand their nature.

3. Framework for Detecting Anomalous End User Nodes

3.1 Overview of Anomaly Detection Framework

If an end user node communicates with some port at remote node, we define *peer* as a tuple *(transport protocol ID, IP address of remote node, port number at remote node)*. Also, *peer port* is defined as a tuple *(transport protocol ID, port number at remote node)*. For protocols without ports (e.g., ICMP), 0 is used for remote port number.

For studying typical network usage patterns of end user nodes, we analyzed a 1-month NetFlow data set for 78 workstations. We divided this data set into 1 hour timeframes and investigated how many peers and peer ports each node had accessed during hourly timeframes of its activity. Firstly, total numbers of peers for the entire month remained relatively modest, and an average node accessed 1213.2 peers (largest number of peers per node was 4688).

Secondly, significant part of the flows were associated with a small fraction of frequently used peers (see Table 1). For example, an average node contacted only 51.7 peers during at least half of its activity hours, but 69.9% of flows were associated with these peers.

Table 1: Network usage patterns of end user nodes

N	*Number of peers contacted during at least N% of hourly timeframes when node was active (average over all nodes)*	*Fraction of flows associated with peers (average over all nodes)*
10	292.6	92.4%
25	145.9	83.4%
50	51.7	69.9%

Also, we discovered similar trends for peer ports – for example, an average node contacted 8.9 peer ports during at least half of its activity hours, and 96.6% of flows were associated with these peer ports. Finally, we have observed similar regularities for NetFlow data sets collected in different environments (Vaarandi, 2013).

These findings suggest that if a new flow is observed for an end user node, it is likely that the flow is associated with a peer and peer port the node has already accessed in recent past. In other words, *recent communication patterns of end user node can be harnessed for predicting its future behavior*, and we have developed an anomaly detection framework which relies on that assumption (see Figure 1).

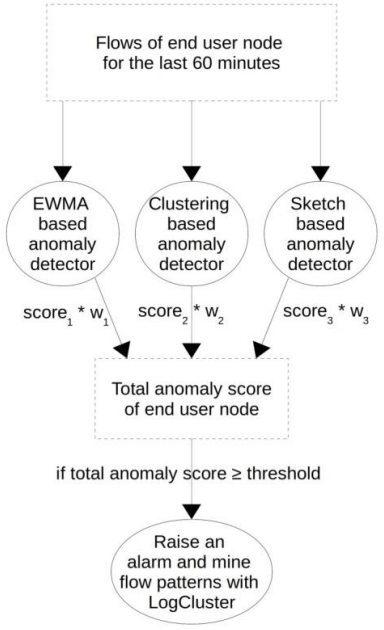

Figure 1: Overview of the framework

The framework executes once in every hour on NetFlow collector node, and processes the flows of last 60 minutes for each end user node. If the node has no flows, it will be skipped due to its inactivity. The framework

consists of three anomaly detectors which calculate anomaly scores $score_1$, $score_2$ and $score_3$ from range 0..1 for a node. The scores are aggregated into total anomaly score $score_1*w_1 + score_2*w_2 + score_3*w_3$ with non-negative weights w_1, w_2 and w_3, where $w_1 + w_2 + w_3 = 1$ (therefore, total anomaly score ranges from 0 to 1). If node's total anomaly score exceeds a given threshold, an alarm is raised and flows of that node for the last 60 minutes are mined with LogCluster algorithm, in order to identify anomalous network traffic patterns (LogCluster is known to be well suited for analyzing security logs (Vaarandi, Kont and Pihelgas, 2016)). The following subsections provide a detailed discussion of each anomaly detector from Figure 1.

3.2 EWMA Based Anomaly Detector

Since it has been shown that EWMA based anomaly detection methods are efficient for network traffic analysis (Hofstede et al, 2013), one of anomaly detectors employs a similar approach. If $X = \{x_1, x_2, ...\}$ is a time series, EWMA μ and exponentially weighted moving standard deviation σ are calculated according to Equation 1.

$$\begin{cases} \mu_1 = x_1 \\ \mu_i = \alpha * x_i + (1-\alpha) * \mu_{i-1} \ , \ i > 1 \end{cases} \qquad \begin{cases} var_1 = 0 \\ var_i = (1-\alpha) * (var_{i-1} + \alpha * (x_i - \mu_{i-1})^2) \ , \ i > 1 \\ \sigma_i = \sqrt{var_i} \end{cases}$$

Equation 1: Exponentially weighted moving average and standard deviation

In Equation 1, α ranges from 0 to 1, with larger values of α giving more weight to recent observations of X. Values close to 0 are distributing weight more evenly, and EWMA is known to estimate the average of last $(2/\alpha)$-1 values from X. For detecting anomalies for time series X, the following method is often used:
if $|x_i - \mu_{i-1}| > m * \sigma_{i-1}$, then x_i is regarded anomalous (m is a user defined constant and commonly set to 3).

The above approach has motivated EWMA based anomaly detector for identifying unexpected increases in the number of peers and peer ports, and the volume of traffic exchanged with peers (according to section 3.1, end user nodes communicate with modest number of peers). For each node E, the anomaly detector tracks the following six features:

- *Peers* – number of unique peers for node E per 1 hour
- *RarePeers* – number of unique peers for node E per 1 hour, so that E has not communicated with these peers during the last N hours of its activity (during our experiments, we have set $N = 50$)
- *PeerPorts* – number of unique peer ports for node E per 1 hour
- *LogFlows* – $log_{10}M$, where M is the number of flows for node E per 1 hour
- *LogPackets* – $log_{10}M$, where M is the number of packets sent and received by node E per 1 hour
- *LogBytes* – $log_{10}M$, where M is the number of bytes sent and received by node E per 1 hour

The anomaly detector raises an alarm for any of above features if $x_i - \mu_{i-1} > m * \sigma_{i-1}$, i.e., the value of a feature increases significantly. We have employed the settings $m = 3$ and $\alpha = 0.05$ for anomaly detector, and with α=0.05, μ_{i-1} estimates the average feature value over previous 39 hours (that is the approximate working week length for end users, and end user nodes are often switched off outside office hours). The purpose of the *RarePeers* feature is to detect unexpected increase of the number of unusual peers, even if the overall number of peers stays within expected boundaries. Also, instead of tracking the number of flows, packets, and bytes, these values have been converted to logarithmic scale for lessening the number of false positives. Finally, each feature contributes equally to the anomaly score reported by the detector. For example, if for node E alarms have been raised for *RarePeers* and *LogFlows* features, anomaly score 1/3 will be reported for node E.

3.3 Clustering Based Anomaly Detector

Unfortunately, EWMA based anomaly detector is not aware of the surrounding context which might influence the severity of detected anomalies. For example, while downloading unusually large amount of data is an anomaly if observed for one node only, the same simultaneous behavior change by many nodes can have a

benign root cause (e.g., centralized patching of all workstations). The purpose of the clustering based anomaly detector is to find groups of similarly behaving nodes and report outliers as anomalous.

For clustering purposes, each node is represented by a vector with the following attributes – the number of peers, rare peers, and peer ports (defined like *Peers*, *RarePeers*, and *PeerPorts* features in the previous section); total number of bytes, packets, and flows; total number of bytes and packets for outgoing traffic; total number of bytes and packets for incoming traffic. Before clustering, all features are standardized by removing the mean and scaling to unit variance. For clustering the nodes, DBSCAN algorithm (Ester et al, 1996) is used which takes *minPts* and ε input parameters.

DBSCAN regards data point as *core point* if it has at least *minPts* points (including itself) within distance ε (we have used Euclidean distance for anomaly detector). A point q is *reachable* from core point p if either: (1) q is within distance ε from p, or (2) there exist core points $p_1,...,p_k$, so that:

- $p_1 = p$,
- p_i is within distance ε from p_{i-1} $(1 < i \leq k)$,
- q is within distance ε from p_k.

If p is a core point, DBSCAN creates a cluster from p and all points that are reachable from it (therefore, each cluster will contain at least *minPts* points). Points that are not reachable from any other point are regarded outliers.

For favoring the creation of larger clusters with many nodes, the anomaly detector executes DBSCAN with settings *minPts* = 10 and ε = 5. For nodes belonging to clusters, anomaly score 0 is returned. Also, if there are k outliers among n nodes, anomaly score $1-(k/n)$ is returned for each outlier node. Therefore, if there are only few outlier nodes, their anomaly scores are close to 1, while with many outliers (i.e., when anomalous behavior is more common) their anomaly scores will be significantly lower.

3.4 Sketch Based Anomaly Detector

Unlike two previous anomaly detectors that are based on well-known algorithms, this subsection presents a novel anomaly detector which employs one-row sketches for summarizing past communication patterns of each end user node, in order to predict its future behavior. The anomaly detector has three components with the following goals:

- Similarity and entropy based anomaly detection for peers
- Similarity and entropy based anomaly detection for peer ports
- Entropy based anomaly detection for local ports

Each component can raise an alarm which will contribute equally to node's anomaly score (e.g., if two components raise an alarm for a node, its anomaly score is 2/3). The following paragraphs will outline the work of the first peer-related component for node E (the second component works similarly, and the third will be described in the end of this section).

The anomaly detector will start its work with two sketches $S = (s_1,...,s_n)$ and $V = (v_1,...,v_n)$ which are vectors of n counters (we have set n = 100000 for peers, and n = 10000 for peer ports and local ports). The purpose of vector S is to capture recent average communication patterns with peers for E, while V captures this information for the last hour.

When anomaly detector is executed for the flows of E from last 60 minutes, counters of vector V are initialized to zero and updated in the following way:

1. for each flow, a peer is extracted and hashed into a value from range $1..n$,
2. if the value is k, counter v_k is incremented.

If $V_1, V_2, \ldots$ denote above hourly vectors in the order of creation, and v_{ji} denotes the i^{th} counter of vector V_j, the i^{th} counter of vector S is maintained as an EWMA for time series $\{v_{1i}, v_{2i}, \ldots\}$ (see Equation 1). In other words, each subsequently calculated vector $V = (v_1, \ldots, v_n)$ is used for updating $S = (s_1, \ldots, s_n)$ according to Equation 2.

$$s_i := \alpha * v_i + (1 - \alpha) * s_i \ , \ 1 \leq i \leq n$$

Equation 2: Vector update procedure

Therefore, counter v_i is equal to the number of flows during the last hour for a group of one or more peers, while counter s_i estimates the average number of flows during last $(2/\alpha)$-1 hours for the same peer group. Thus, vectors S and V represent distributions of flows over peer groups. Since vector S reflects recent communication patterns with peers that serve as a good predictor for the future behavior of the node (see section 3.1), significant difference between vectors S and V can be regarded as an anomaly.

For measuring the difference between two vectors, we experimented with several distance functions and discovered that cosine similarity produces best results (see Equation 3).

$$cosim(S, V) = \frac{\sum_{i=1}^{n} s_i * v_i}{\sqrt{\sum_{i=1}^{n} s_i^2} * \sqrt{\sum_{i=1}^{n} v_i^2}}$$

Equation 3: Cosine similarity

Cosine similarity measures the angle between two vectors, and ranges from 0 to 1 since counters of S and V are non-negative. If $cosim(S,V) = 1$, vectors S and V have the same direction (e.g., vectors (1,0,23,0) and (2,0,46,0)), and if $cosim(S,V) = 0$, vectors are orthogonal. Therefore, values close to 1 indicate that during the last 60 minutes distribution of flows over peer groups is similar to distribution of previous observations, while smaller values of cosine similarity indicate deviations from past measurements.

The use of cosine similarity introduces the following issue – if node behavior changes frequently and has no clear baseline, similarity between S and V remains low and becomes meaningless for anomaly detection. For avoiding false alarms for such nodes, the anomaly detector calculates the similarity $cosim(S,V)$ before each update of S with Equation 2, and maintains EWMA $cavg$ for past similarity values according to Equation 1. Before each update of $cavg$ with current similarity value $cosim(S,V)$, anomaly detector raises an alarm only if $cosim(S,V) < T_{sim}$ and $cavg > T_{avg}$. In other words, alarm is generated if similarity between S and V falls below threshold T_{sim}, with average past similarity $cavg$ being sufficiently high and exceeding T_{avg} (we have used the settings $\alpha = 0.05$, $T_{sim} = 0.5$ and $T_{avg} = 0.8$ for anomaly detector).

In addition, we have also implemented entropy based anomaly detection for vector S, in order to identify nodes with constantly changing behavior (since $cavg$ is usually smaller than threshold T_{avg} for such nodes, similarity based anomaly detection tends to be disabled for them). According to section 3.1, end user nodes are mostly communicating with relatively small number of peers which results in relatively few counters of vector S having larger values, while many counter values remain zero. On the other hand, if node behavior is frequently changing, counter values in S would be less different. In previous research papers, entropy based techniques have been successfully used for capturing such regularities (Zhou et al, 2015).

If X is a discrete random variable that can take k values with probabilities $p_1, \ldots, p_k$ ($p_1 + \ldots + p_k = 1$), *normalized information entropy* of X is defined by Equation 4.

$$norment(X) = \frac{-\sum_{i=1}^{k} p_i * log(p_i)}{log(k)}$$

Equation 4: Normalized information entropy

Note that *norment*(*X*) ranges from 0 to 1, with values close to 1 indicating that distribution of *X* is close to uniform. Also, smaller values indicate that *X* takes some values with significantly higher probabilities. In order to employ entropy based anomaly detection for *S*, we first find its normalized vector $\hat{S} = (1/j) * S$, where $j = s_1+\ldots+s_n$ (note that $\hat{s}_1+\ldots+\hat{s}_n = 1$). After that, the entropy of *S* is calculated according to Equation 5.

$$norment(S) = \frac{-\sum_{i=1}^{n} \hat{s}_i * log(\hat{s}_i)}{log(n)}$$

Equation 5: Calculating sketch entropy

The anomaly detector calculates the entropy of *S* after each update with Equation 2, and raises an alarm if entropy is higher than T_{ent} (for peers and peer ports, we have set T_{ent} to 0.5 and 0.3 respectively). Finally, since typical end user nodes do not use fixed local ports for communicating with remote services but ports are selected randomly, the anomaly detector also maintains vector *S* for local ports of each end user node. Since *S* is expected to have a high entropy, the anomaly detector raises an alarm if the entropy falls below T_{ent2} (we have set T_{ent2} to 0.2 during experiments).

3.5 Mining Flow Patterns with LogCluster

If node's total anomaly score for last 60 minutes exceeds a given threshold, flows of last 60 minutes for this node are mined with LogCluster algorithm, in order to detect prominent traffic patterns that capture the nature of anomaly. Since LogCluster is designed for mining textual log files, each binary flow record is converted into a textual line with the following keyword-value based format: *proto <transport protocol ID> srcip <source IP address> srcport <source port> dstip <destination IP address> dstport <destination port> tcpflags <flagstring>*. For easing the conversion process, we have used Perl based LogCluster implementation with advanced input preprocessing features (https://ristov.github.io/logcluster/).

For detecting patterns from log file with LogCluster, the user has to specify *support threshold* value which ranges from 1 to *n*, where *n* is the number of lines in the log file. If support threshold is *s*, LogCluster identifies patterns that match at least *s* lines in the log file. During the first pass over the data set, LogCluster will split each line into words, and identify *frequent words* in the data set (words which appear in at least *s* lines). During the second data pass, LogCluster extracts all frequent words from each line, so that the combination of frequent words identifies a *cluster candidate* for the given line. For each cluster candidate, LogCluster memorizes the number of lines for this candidate, and summary information about the location of infrequent words for all lines of the candidate. After the data pass, cluster candidates with at least *s* lines are selected as clusters, and reported to the end user as line patterns. For example, the following pattern represents DNS queries from node 10.3.7.22 to server 192.168.1.1 port 53/UDP (note that **{1,1}* is a wildcard which matches exactly one word):

*proto 17 srcip 10.3.7.22 srcport *{1,1} dstip 192.168.1.1 dstport 53 tcpflags NA*

As discussed in (Vaarandi, Blumbergs and Kont, 2018), finding a good support threshold value for LogCluster is a non-trivial issue. However, we have discovered that if NetFlow data set contains *n* flow records, support threshold value $\sqrt{n}$ allows to adequately highlight the nature of anomalous traffic in most cases. Also, we have configured LogCluster to use word weight based heuristics for joining patterns with the same nature (Vaarandi, Kont and Pihelgas, 2016). For example, in the case of two similar patterns that reflect DNS queries from 10.1.1.1 to servers 192.168.1.1 and 192.168.1.2, LogCluster would join them into the following single pattern:

*proto 17 srcip 10.1.1.1 srcport *{1,1} dstip (192.168.1.1|192.168.1.2) dstport 53 tcpflags NA*.

4. Evaluation

We have implemented our anomaly detection framework in Perl and Python, and measured its performance during 5 months (October 2018 – February 2019, 151 days) in a network of an academic institution with over 200 workstations and laptops. The framework used *softflowd* NetFlow exporter (https://code.google.com/archive/p/softflowd/) on a dedicated Linux host for monitoring Internet traffic of all end user nodes without sampling, and flow data was collected with *nfdump* NetFlow collector (https://github.com/phaag/nfdump).

When evaluating the framework, we used each anomaly detector with an equal weight of 1/3 and set anomaly score threshold to 0.5 (see Figure 1). During 5 months, the framework generated 1026 alarms about 33 end user nodes. However, over 90% of alarms were triggered for 4 hosts, and for 25 hosts only 1-3 alarms were raised. Figure 2 depicts some anomalous traffic patterns the framework discovered with LogCluster.

```
# Patterns detected for Internet measurement probe 10.1.1.1. The first pattern reflects ICMP echo (ping) packets
# sent to large number of Internet hosts (type 8, code 0), while the second pattern reflects ICMP response packets
# from Internet hosts: ICMP echo reply (type 0, code 0), ICMP network unreachable (type 3, code 0), ICMP TTL
# expired in transit (type 11, code 0).

proto 1 srcip 10.1.1.1 srcport 0 dstip *{1,1} dstport 8.0 tcpflags NA

proto 1 srcip *{1,1} srcport 0 dstip 10.1.1.1 dstport (11.0|0.0|3.0) tcpflags NA


# Patterns detected for host 10.1.1.7 that runs BitTorrent client at known BitTorrent port 8999/udp, and exchanges
# data with other known BitTorrent ports 6881/udp and 51413/udp at remote hosts.

proto 17 srcip 10.1.1.7 srcport 8999 dstip *{1,1} dstport (51413|6881) tcpflags NA

proto 17 srcip *{1,1} srcport (6881|51413) dstip 10.1.1.7 dstport 8999 tcpflags NA


# Patterns detected for host 10.1.1.12 that executed TCP SYN scan of host 192.168.16.250 with nmap -sS
# (note that all TCP SYN packets were sent from a single port 44290/tcp).

proto 6 srcip 10.1.1.12 srcport 44290 dstip 192.168.16.250 dstport *{1,1} tcpflags without-ACK

proto 6 srcip 192.168.16.250 srcport *{1,1} dstip 10.1.1.12 dstport 44290 tcpflags with-ACK
```

Figure 2: Flow patterns for anomalous network traffic

When investigating the alarms more closely, we found that 638 of them (62.2%) were generated for 5 hosts which were running BitTorrent client for file sharing purposes, although such activity is not allowed by organizational policies (see the second example in Figure 2). Also, 327 alarms (31.9%) were triggered for an Internet measurement probe which was accidentally connected to end user network (see the first example in Figure 2). According to the developer of the probe, it needs at most 100 Kbit/s of bandwidth and typical consumption does not exceed couple of Kbit/s (Internet Measurement Project, 2017). Despite its negligible network footprint, the anomaly detection framework was able to flag the probe as anomalous. Also, we found that the framework generated 46 false positive alarms (4.5%), with most of them being triggered by downloads of large files.

For estimating the precision and recall of the framework for attack traffic, we conducted network scanning with *nmap* tool from one of the end user nodes during 15 hours. Altogether, the node was active for 548 hours during 5 month timeframe, and no malicious network traffic was observed for this node during remaining 533 hours. Network scanning was conducted against test targets in controlled environment, and scanning scenarios involved different scans of a single host (4 hours) and a network of 2048 hosts (11 hours). Scanning types included lightweight reconnaissance scans, TCP SYN, Xmas, Fin and Null scans, and scans for detecting operating system and service versions. We also simulated TCP and UDP flooding attacks against the network of 2048 hosts. All 15 hours of malicious activity were correctly flagged as anomalous (see the third example in

Figure 2 for some detected patterns), and no false alarms were raised for 533 hours of legitimate network activity, yielding the precision and recall of 100%.

Finally, we measured what is the impact of using multiple classifiers in the anomaly detection framework, and executed each anomaly detector from Figure 1 separately with weight 1 and anomaly score threshold set to 0.5. The EWMA based anomaly detector triggered 570 alarms, clustering based detector 4020 alarms, and sketch based detector 1696 alarms. Table 2 depicts the performance of individual detectors for a node which we used for *nmap* scanning.

Table 2: Precision and recall of individual detectors for *nmap* scanning

	Precision	*Recall*
EWMA based detector	50%	20%
Clustering based detector	37.5%	100%
Sketch based detector	88.2%	100%

The primary reason for poor recall of EWMA based detector is the fact that abnormally large values can significantly increase the average, so that the following similarly large values will no longer be seen as anomalous (this condition will persist until a sequence of smaller values will lower the average again). Also, although sketch based detector had the best precision and recall, it triggers more false positive alarms than the framework of three detectors. We have made a similar observation for other nodes in the data set, and the use of multiple classification methods is thus beneficial over a single classifier.

5. Future Work

The framework presented in this paper assumes that each node belongs to one person (or few similarly behaving persons), so that network usage habits of node owners create behavior baselines for nodes which can be employed for anomaly detection. However, if many persons with different assignments are sharing the nodes (e.g., classroom computers in a university network), it is often difficult to identify clear behavior baselines for them. A similar issue arises if IP addresses of nodes are frequently changing (e.g., due to DHCP based IP address allocation) and the same address is reused for different nodes. In order to overcome these issues, the framework needs to identify the end user node not by IP address, but rather by unique ID of the person that operates the node (e.g., an organizational user account name). Augmenting the framework with a monitoring module for detecting user IDs has been left for future work.

As for other future work, we plan to integrate other anomaly detectors into the framework and experiment with methods for adjusting detector weights dynamically. We are also planning to research other approaches for mining traffic patterns from NetFlow data sets, and algorithms for identifying node types in organizational networks. Finally, we are considering to study methods for fingerprinting end users by their network usage patterns in large organizational networks.

Acknowledgements

The authors thank Prof. Olaf M. Maennel and Prof. Rain Ottis for supporting this work.

References

Brauckhoff, D., Dimitropoulos, X., Wagner, A. and Salamatian, K. (2012) “Anomaly Extraction in Backbone Networks Using Association Rules,” IEEE/ACM Transactions on Networking, vol. 20, no. 6, pp. 1788–1799.

Ester, M., Kriegel, H.-P., Sander, J. and Xu, X. (1996) “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise,” Proceedings of the 1996 International Conference on Knowledge Discovery and Data Mining, pp. 226–231.

Grill, M., Nikolaev, I., Valeros, V. and Rehak, M. (2015) “Detecting DGA malware using NetFlow,” Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management, pp. 1304–1309.

Hofstede, R., Bartoš, V., Sperotto, A. and Pras, A. (2013) “Towards Real-Time Intrusion Detection for NetFlow and IPFIX,” Proceedings of the 2013 International Conference on Network and Service Management, pp. 227–234.

Hofstede, R., Čeleda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A. and Pras, A. (2014) “Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX,” IEEE Communication Surveys and Tutorials, vol. 16, no. 4, pp. 2037–2064.

Hou, J., Fu, P., Cao, Z. and Xu, A. (2018) “Machine Learning based DDos Detection Through NetFlow Analysis,” Proceedings of the 2018 IEEE Military Communications Conference, pp. 565–570.

Internet Measurement Project (2017) “Internet Measurement Project FAQ,” [online], University of Nevada, Reno, https://im.cse.unr.edu/?page=FAQ.

Kind, A., Stoecklin, M. Ph. and Dimitropoulos, X. (2009) “Histogram-based traffic anomaly detection,” IEEE Transactions on Network and Service Management, vol. 6, no. 2, pp. 110–121.

Muhs, D., Haas, S., Strufe, T. and Fischer, M. (2018) “On the Robustness of Random Walk Algorithms for the Detection of Unstructured P2P Botnets,” Proceedings of the 2018 International Conference on IT Security Incident Management and IT Forensics, pp. 3–14.

Paredes-Oliva, I., Castell-Uroz, I., Barlet-Ros, P., Dimitropoulos, X. and Solé-Pareta, J. (2012) “Practical Anomaly Detection based on Classifying Frequent Traffic Patterns,” Proceedings of the 2012 IEEE INFOCOM Workshops, pp. 49–54.

SANS (2018) “2018 SANS Security Awareness Report: Building Successful Security Awareness Programs,” [online], https://www.sans.org/security-awareness-training/reports/2018-security-awareness-report.

Symantec (2019) “Internet Security Threat Report,” [online], Volume 24, February 2019, https://www.symantec.com/content/dam/symantec/docs/reports/istr-24-2019-en.pdf.

Vaarandi, R. (2013) “Detecting Anomalous Network Traffic in Organizational Private Networks,” Proceedings of the 2013 IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support, pp. 285–292.

Vaarandi, R., Kont, M. and Pihelgas, M. (2016) “Event Log Analysis with the LogCluster Tool,” Proceedings of the 2016 IEEE Military Communications Conference, pp. 982–987.

Vaarandi, R., Blumbergs, B. and Kont, M. (2018) “An Unsupervised Framework for Detecting Anomalous Messages from Syslog Log Files,” Proceedings of the 2018 IEEE/IFIP Network Operations and Management Symposium, pp. 1–6.

Zhou, F., Huang, W., Zhao, Y., Shi, Y., Liang, X. and Fan, X. (2015) “ENTVis: A Visual Analytic Tool for Entropy-Based Network Traffic Anomaly Detection,” IEEE Computer Graphics and Applications, vol. 35, no. 6, pp. 42–50.

Appendix 11

Publication XI
M. Pihelgas and M. Kont. Frankenstack: Real-time Cyberattack Detection and Feedback System for Technical Cyber Exercises. In *2021 IEEE CSR Workshop on Cyber Ranges and Security Training (CRST)*. IEEE, July 2021. (Accepted paper)

Frankenstack: Real-time Cyberattack Detection and Feedback System for Technical Cyber Exercises

Mauno Pihelgas
Technology Branch
NATO Cooperative Cyber Defence Centre of Excellence
Tallinn, Estonia
mauno.pihelgas@ccdcoe.org

Markus Kont
Research and Development
Stamus Networks
Tallinn, Estonia
markus@stamus-networks.com

***Abstract*—This paper describes a situation awareness framework, *Frankenstack*, that is the result of a multi-faceted endeavor to enhance the expertise of cybersecurity specialists by providing them with real-time feedback during cybersecurity exercises and verifying the performance and applicability of monitoring tools during those exercises. Frankenstack has been recently redeveloped to improve data collection and processing functions as well as cyberattack detection capability. This extensive R&D effort has combined various system and network security monitoring tools into a single cyberattack detection and exercise feedback framework.**

Although Frankenstack was specifically developed for the NATO CCD COE's Crossed Swords exercise, the architecture provides a clear point of reference for others who are building such monitoring frameworks. Thus, the paper contains many technical descriptions to reduce the gap between theoretical research and practitioners seeking advice on how to implement such complex systems.

***Index Terms*—automation, cyber exercises, cyber ranges, Frankenstack, monitoring, NATO Cyber Range, real-time feedback, security training, technical architecture**

I. Introduction

Cybersecurity exercises (CSXs) are key to enhancing cybersecurity operator readiness while also improving situation awareness (SA) in the cyber domain. Crossed Swords (XS) [1] is an annual interdisciplinary CSX directed at training participants for responsive cyber-kinetic operations. Although NATO nations have begun acknowledging the necessity of both defensive and offensive cyber capabilities, there are few exercises that tackle such a controversial subject. XS is organized by the NATO CCD COE in the NATO Cyber Range and utilizes a hybrid approach between cyber-physical and simulated infrastructure.

Cyber-exercise-specific SA systems are designed to improve SA during cyber exercises. While traditional SA systems are oriented toward cyber defenders, CSX-specific SA systems have been designed to provide situation awareness feedback not only to cyber defenders, but all participating teams alike.

CSXs introduce some unique requirements. The exercise environment must support the SA systems which measure the performance of participants. It is also important to assess the learning experience of participants to improve future iterations of the exercise.

978-1-7281-5684-2/20/$31.00 ©2021 IEEE

Stealth is an important factor when conducting cyber operations. During XS, the target Blue team (BT) hosts and networks are closely monitored for any indicators of compromise (IoC) which are then narrated to the training audience for learning purposes. Since 2016, a small team of cybersecurity monitoring specialists, the Yellow team (YT) in the exercise jargon, has been working to develop and improve the real-time CSX-specific framework known as Frankenstack that automates the entire cyberattack detection and feedback cycle for the training audience. The initial version of Frankenstack was described in our 2017 paper [2]. Since then, XS has been used as a platform to further develop, test, and validate the framework. The open-source solution is widely applicable to any other CSX where standard exercise technical infrastructure monitoring capability is available.

A. Problem statement

Some of the problems that inspired the creation of the initial version of Frankenstack were due to issues with the participants' learning experience. During the earlier iterations of XS, the YT feedback sessions regarding detected malicious activity were presented only at the end of each day. Due to limited time, these sessions were only able to summarize the primary observations from that day. This did not suffice as the technical training audience needed direct and faster feedback about their actions to pinpoint mistakes as they happened. This immediate feedback also needed to be adequately detailed so that the participants could understand how and why a particular attack was detected.

Additionally, most of the data analysis in YT was done manually by team members. This was slow and sometimes inconsistent in how attacks were followed up. Thus, another objective was to provide uniform feedback that would be clear and understandable for all participants.

Moreover, since the defending BT in XS is mostly just passively observing the situation, it often remained unclear to the participants whether any of their recent attacks would have affected the security posture of the adversary. While this reflects experience in the real world, it did not help participants' learning experiences during the CSX.

Furthermore, commercial SA systems are often too expensive to be acquired solely for cyber exercises due to

license fees, hardware cost, and vendor-specific knowledge. The proprietary detection logic in commercial tools is often unavailable which again restricts YT's ability to understand and provide meaningful explanations of detected attacks.

Finally, the initial version of Frankenstack was too complex and involved several overlapping utilities (e.g., syslog-ng and rsyslog). While each tool had its purpose, the extra multiplicity rendered the data pipeline difficult to set up and maintain. The primary aim of the redevelopment was to reduce complexity and replace the use of many smaller utilities with tailor-made solutions that are described in this paper.

B. Structure

The remainder of this paper is organized as follows: section II provides some general background information about the XS exercise, section III presents an overview of related work, section IV describes the improvements and the technical architecture of the newly developed Frankenstack framework, section V discusses our efforts at providing relevant real-time feedback and appropriate visualizations for exercise participants, section VI briefly outlines the collaboration with industry partners, section VII defines future work, and section VIII concludes the paper.

II. Exercise background

Crossed Swords is an annual CSX that has been developed and organized by the NATO CCD COE since 2014. Although it started out as a primarily technical exercise, it has evolved into an interdisciplinary cyber exercise that involves technical, strategic, operational, and legal training aspects. It features a fictional scenario involving two notional countries, Berylia and Crimsonia. While most cyber exercises focus on training the defensive capability for Blue teams, XS reverses the role of the training audience, who now assumes the role of the Offensive Cyber Operations (OCO) team that is exercising a responsive cyber-kinetic scenario. The OCO team must work in close cooperation to discover an unknown network, complete a set of challenges, and collect evidence from the network for proper cyberattack attribution. Under such conditions, attribution, especially in the cyber domain, is increasingly hard to establish [3]. Another goal for the OCO team is to stay as stealthy as possible to avoid being detected by the monitoring stack which is described in this paper.

To develop and carry out the exercise, multiple teams are engaged: game network and infrastructure development (Green team – GT); game scenario development and execution control (White team – WT); defending team user simulation (BT); exercise monitoring and situation awareness (Yellow team – YT); and exercise training audience (OCO team). Note, that the *OCO team* was previously referred to as the Red team (RT), but since XS 2020 the terminology has been updated to better reflect current policies. Due to the reversed roles of the participants, the *defending* BT is actually the adversary that prompted the OCO team's responsive action.

Since offensive measures often take place in a cyber-physical space, the XS scenario portrays this by offering a variety of challenges that require a diverse set of skills and effective communication between members of the OCO team. In addition to physical devices such as programmable logic controllers (PLCs), IP cameras, radio devices, the exercise mimics realistic computer networks with a variety of different hosts (e.g., servers, workstations, and network devices) and operating systems. For instance, the networking subteam is responsible for attacking network services, protocols and routing; the client-side subteam targets human operators and attempts to gain foothold in the adversary's internal network segments; web experts attempt to compromise web services, applications and any associated databases; the digital forensics subteam performs data extraction and artefact collection; and kinetic forces provide support in operations that require a kinetic element such as physical surveillance, hardware extraction, forced entry, target capture, etc.

It is important to distinguish XS from capture-the-flag exercises. The participating subteams are not competing with one another, but rather serve as dedicated segments of a single military detachment. The exercise scenario is developed in a way that all subteams must coordinate their actions and share intelligence to achieve their objectives and advance in the exercise environment.

III. Related work

Research on exercises with an emphasis on offensive operations (such as XS) is almost non-existent. This is likely due to the high sensitivity of offensive cyber operations. However, there is an increasing amount of research based on other defensive CSXs and work that describes CSX-specific SA systems. Although not directly applicable in XS, the CSX-specific elements in those tools could still be considered relevant.

Känzig et al. [4] sought to detect command and control (C&C) channels in large networks without prior knowledge of the network characteristics. They leverage the notion that while benign traffic differs, malicious traffic bears similarities across networks. They trained a random forest classifier on a set of computationally efficient features designed for the detection of C&C traffic. They verified their approach using the NATO CCD COE's Locked Shields exercise datasets. Results revealed that the if the LS18 Swiss Blue team had used the system, they would have discovered 10 out 12 C&C servers in the first few hours of the exercise.

[5] Klein et al. compared two different machine learning techniques—the unsupervised autoencoder and the supervised gradient boosting machine—on a partially labelled cyberdefense exercise dataset. Both techniques were able to classify known intrusions as malicious while, surprisingly, also discovering 50 previously unknown attacks.

In [6], Arendt et al. presented CyberPetri, a redesign of the pre-existing Ocelot SA tool [7] which was used to provide real-time SA during the 2016 Cyber Defense Exercise and provide high-level feedback to network analysts based on exercise target systems' service availability reports. The authors note scaling to large datasets as a limitation. The exercise

participants' feedback revealed that the tool was useful for the exercise White team for high-level decision making, but that technical specialists were more interested in improved exploratory capability for specific events and time windows.

A paper [8] from Henshel et al. proposed a performance assessment model of human cyberdefense teams and verified its applicability during the Cyber Shield 2015 exercise. While exercise data was captured during the game, most of the analysis was done after the event. For future iterations, the authors stress the need for real-time analysis of the collected data to adapt training and assessment methods already during the exercise. The ability to analyze the collected data was the primary limiting factor, as operators were not able to keep up with the huge amounts of incoming data.

Maennel focuses on measuring and improving learning effectiveness at cyber exercises [9]. This follows work that was described in the learning feedback section of the initial Frankenstack publication [2]. Furthermore, a recent paper [10] by Ernits et al. discusses how technical event logs and metrics from the exercise game system can be transformed to measure skills and learning outcomes.

Another publication [11] on team-learning assessment by Maennel et al. proposes an unobtrusive method based on mining textual information and metrics from situation reports submitted by teams during cyber exercises. Since these reports are regularly filed by Blue teams as a part of the exercise, this approach would enable gathering relevant information without disturbing the teams by conducting regular surveys and questionnaires throughout the exercise.

Chmelař describes the analysis of the XS 2020 exercise data using the MITRE ATT&CK knowledge base [12] to create reports of the OCO team progress [13]. Although the reports were created as a post-mortem analysis, the author proposes that they could theoretically provide in-game overview and visualizations during the XS exercise.

IV. Frankenstack

The Frankenstack SA framework features a near real-time feedback loop for the OCO team participants: any OCO action that is discovered on the game network and target hosts is analyzed in the automated data processing pipeline and if considered malicious is reported back to the feedback dashboard as an indicator of compromise. This all happens automatically without any human-interaction from the YT operators, allowing OCO members to immediately try again to improve their attack technique to attempt avoiding detection. Naturally, YT is still present to continuously improve the detection capability, reduce false positive alerts, and make sure the entire framework works as intended.

The framework also provides information about participants' progress to the exercise leadership allowing them to control the pace of the scenario more precisely. Figure 1 illustrates Frankenstack's technical architecture and various data flows in the exercise environment.

A. External data sources

Frankenstack requires multiple external data sources for its operation: full network traffic mirror, event logs, network configuration information, and the host asset database.

1) Network packet capture: Receiving a network traffic mirror is crucial for network security monitoring components. The mirrored network traffic was provided via a GRE (Generic Routing Encapsulation) tunnel from the virtual switches in the VMware NSX Data Center software-defined networking solution. Capturing traffic from virtual switches instead of edge routers meant that Frankenstack's IDS component (i.e., Suricata [14]) also had visibility in internal network segments, not just traffic that was traversing between network routers and perimeter firewalls. All traffic was captured and indexed using the Arkime full packet capture and analysis tool which was later used by the YT operators to extract new IoC samples directly from the indexed PCAPs to improve existing detection capability [15]. Note, that Arkime was re-branded from Moloch in November 2020.

2) Event logs: We collected event logs from the in-game (*gamenet*) systems wherever possible (e.g., Event Logs from Windows, Apache and nginx web server logs, and syslog from Linux). Windows Event logging was extended with additional rules for Microsoft Sysinternals Sysmon [16] from public GitHub sources [17], [18]. Instead of implementing AppArmor and SELinux for enhanced Linux auditing, we opted to use a small library called Snoopy Logger [19]. This was because configuring AppArmor or SELinux typically involves increasing the base level of security on the system. However, we did not want to interfere or impair any of the pre-configured vulnerabilities that were planted on the target systems.

Such host instrumentation is difficult to sustain in a standard defense-oriented CSX with BTs as the training audience. If the aim is to give BTs full control of their gamenet infrastructure, then it is difficult to ensure that BTs do not disable or reconfigure these tools. However, as the XS training audience is the OCO team, then YT could maintain supervisory control of all BT systems and ensure a constant stream of event log data.

3) Asset collection: Another critical piece of information is up-to-date knowledge about various network configurations and hosts in these networks. Since this is highly specific to the exercise and the underlying technical infrastructure, we developed a custom script that extracts this information from the Cyber Range provisioning API and the VMware vSphere API. While the provisioning API contains the information about how networks and hosts should be configured when they are deployed by Ansible [20], it lacks any knowledge of changes introduced after the initial deployment. Therefore, the list of all provisioned hosts and any static IP information can be easily collected from the provisioning API. However, if a host has been configured to obtain an IP address using DHCP, then the actual IP address must be retrieved from the machine during runtime. IPv6 link-local addresses are also assigned

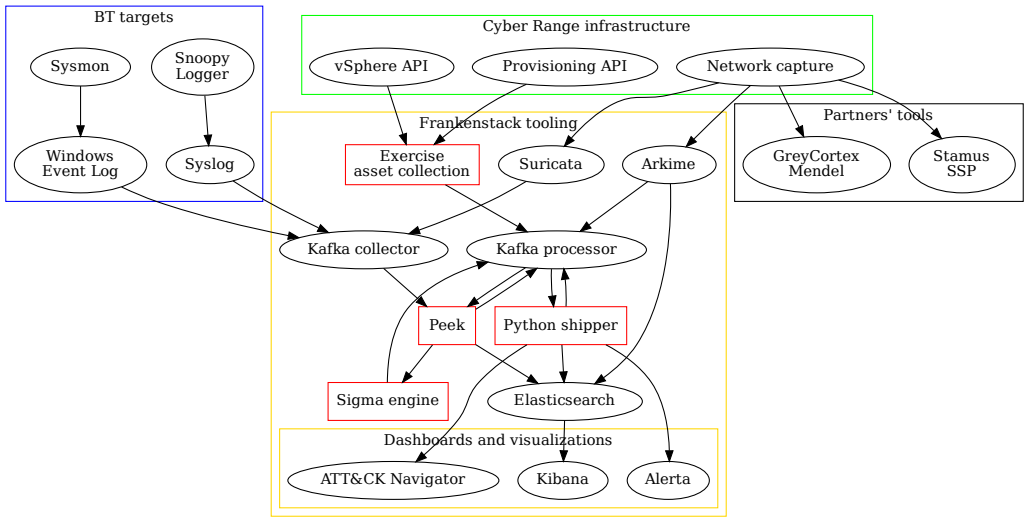

Fig. 1. XS technological perspective. Green area marks data sources originating from the exercise backend infrastructure, i.e., Green team assets (described in sections IV-A1 and IV-A3). Blue area indicates tools and data sources deployed on BT hosts and networks (described in section IV-A2). Black area displays partner tools (described in chapter VI). Yellow area illustrates Frankenstack components and the data flow between them (described in IV-B, IV-C and IV-D). YT Dashboards and visualizations are described in chapter V. Red nodes represent novel contributions developed solely by the authors of the paper.

dynamically. The script periodically collected the most recent IP information from all hosts using the vSphere API.

When information is gathered, we establish an asset profile for each known host in the exercise environment. Among other information, this profile maps all known IPv4 and IPv6 addresses to a particular host. This is key because when it comes to correlating events from multiple data sources, Frankenstack aims to be agnostic as to whether it has to match a hostname (e.g., from event logs) or on an IP address (e.g., from an IDS alert). This problem is something that many network monitoring solutions struggle with because it is not a trivial task to associate IPv4 and IPv6 network sessions even if they originate from the same host [21]. Trying to combine IDS alerts and event logs complicates things even further. For this reason, we attempt to enrich all events with an asset host name as a common identifier for all subsequent event correlation.

B. Distributed event streaming

The initial design of the event processing pipeline described in [2] relied primarily on Syslog-ng [22] to collect, store and forward events. This works well for systems and applications that are set up and configured beforehand so that proper syslog rules can be created. However, during the annual hackathons that precede the XS event, contributing partners in the YT would often need to integrate their own tools and scripts that needed to analyze the same incoming data feeds or a subset of past events to provide an alternative assessment to the main data processing pipeline. Having events stored as files on the central log collection server is not ideal for this. Alternatively, Elasticsearch [23] can be used to query historic data, but there is no good way to re-stream all incoming events in real time.

We analyzed various distributed message streaming tools (e.g., RabbitMQ and ZeroMQ) but opted for using Apache Kafka [24] as a central collection point for all emitted messages. Kafka fulfils the requirement for a multi-producer and multi-consumer event feeds. The general design concept in the Frankenstack framework is that all new events should be produced to Kafka and for any further processing, tools should consume the corresponding events from Kafka. Therefore, Kafka always has all the information and relevant stages of the data analysis processes.

C. Data processing components

Postmortem analysis of available datasets is an irreplaceable method during incident analysis. Although it often gives valuable insight about cyberattacks, it requires a substantial amount of time and manual work. Unfortunately, this conflicts with the short time span of the XS exercise and is not a viable method to keep track of the training audience. Frankenstack processes structured input data that has been sent to Kafka and applies event normalization, enrichment, and correlation for combining various information sources into a single stream of meaningful events.

All relevant event streams in Frankenstack have been configured to output structured JSON events at the source or to be transformed into JSON during the pre-processing phase. Unfortunately, the JSON events emitted by distinct sources feature varying structure which has to be individually handled for every input. Until 2019, Frankenstack employed SEC [25] as the primary data normalization, enrichment, and correlation tool. Bearing in mind that SEC and its rule language was initially designed for complex event correlation tasks on unstructured messages, it soon became difficult to handle complex nested JSON data structures in the SEC rule language. For example, any changes in input JSON key values resulted in the need to edit numerous textual rules. Therefore, instead of using the conventional SEC rule syntax, we had to write Perl code snippets into most FrankenSEC rules [26]. Although it was possible to accomplish what we required using custom Perl functions, the rule writing and management soon became infeasible to maintain.

Alternatively, processing complex nested data structures in a fully-fledged programming language seemed more approachable. In hindsight, our primary issue was that we attempted to correlate events too soon in the data processing pipeline—SEC is an event correlation tool, not a programming environment or a data normalizer. Unfortunately, in our ruleset we tried to accommodate many of the data processing and transformation tasks which should have been completed prior to sending events into SEC. This significantly hindered the SEC event correlation rule-writing process. The lack of proper post-processing meant that even minor changes in the input event structure resulted in the need to rewrite a large portion of the rules.

In the initial version of Frankenstack we had to integrate multiple logging tools (e.g., Syslog-ng, Rsyslog [27], and Logstash [28]) and custom scripts to implement a CSX-specific data normalization and enrichment tool. Configurations to process and route the event stream became overly complex as no single tool was readily able to satisfy all requirements. As a replacement we developed a novel data normalization and enrichment tool called Peek [29]. A fully customizable tool reduced this overhead significantly—we had to maintain only one tool which was fully under our control. Peek was able to replace generic log processing and event routing tools such as Logstash and Syslog-ng within our framework, albeit only for our particular exercise use-case.

The FrankenSEC ruleset was replaced by the Sigma ruleset [30] in XS 2020. Sigma is an open-source project that has gained a wide support and adoption from the information security community in the past few years. The project defines a simple rule structure in YAML format. Sigma does not do any pattern matching or alerting by itself. Rather, it acts as a technical translation layer and IoC sharing format. See Listing 1 for an example sigma rule that we developed for detecting base64 encoded scripts being executed in Windows machines. The rule triggers if the string *' -FromBase64String '* is detected within the *ScriptBlockText* field of a Windows Event.

Since Frankenstack aims to be vendor-agnostic, we built a custom real-time rule matching engine in Go [31] that uses Sigma rules. That engine is built as separate module and is publicly available in GitHub [32]. A detailed description and benchmarks of our Sigma rule engine are available in our recent whitepaper [33].

Listing 1. Sigma rule to detect base64 encoded PowerShell scripts.

```
title: Encoded ScriptBlock Command Invocation
author: Mauno Pihelgas
description: Detects suspicious PowerShell invocation
             command parameters
detection:
  condition: selection
  selection:
    winlog.event_data.ScriptBlockText:
    - ' -FromBase64String '
falsepositives:
  - Penetration tests
  - Very special PowerShell scripts
fields:
  - winlog.event_data.ScriptBlockText
id: 697e4279-4b0d-4b14-b233-9596bc1cacda
level: high
logsource:
  product: windows
  service: powershell
status: experimental
tags:
  - attack.execution
  - attack.defense-evasion
  - attack.t1059.001
```

The interface to the OCO feedback dashboards was accomplished with a comprehensive Python post-processing and event shipping module which ensured that all processed events sent to the dashboards (i.e., Alerta [34], ATT&CK Navigator [35] and Kibana [36]) conformed to a uniform structure and were mapped to the MITRE ATT&CK adversary tactics and techniques knowledge base. This post-processing script also implemented a simple baselining functionality to identify security events that occur under normal system use (e.g., execution of scheduled tasks, system updates, etc.). Later, during the exercise, such events were automatically filtered and not displayed on the dashboards. The same post-processing tool can be leveraged to easily create additional filters to suppress benign or false positive events which may occur during the exercise.

During the three days of XS exercise in December 2020, the Frankenstack framework received a total of 673,225 input security events. 85% of those were from Windows machines which are highly verbose, especially with the added Sysmon logging rulesets. The remaining 15% were logs emanating from Linux machines and Suricata IDS. To reiterate, manual inspection of such a large number of events for providing near real-time feedback to exercise participants would prove extremely difficult. Therefore, automated event processing steps such as filtering, correlation, and deduplication are of key importance within Frankenstack.

D. Determining the attacker

There is one crucial element that has to be determined for every event before submitting it further—the attacker and

victim assets. To reiterate a problem from the early days of Frankenstack development, one primary concern the YT faced was automatically determining the direction of the attacks, i.e., identifying the victim and the attacker in a particular cyberattack. With Suricata IDS alerts, relying on the *source* and *target* fields does not yield an expected result—the *source* and *target* fields in IDS rules just signify the direction of traffic for which the detection match conditions are written. This meant that whenever the rule writers had written a rule that detects a response of an attack (e.g., sensitive data leaving the victim node), we would have erroneously classified the victim as the attacker. With this approach, we could only connect the relevant nodes, but lacked the directionality between them.

Fortunately, one of the core Suricata developers had been part of YT since 2016. He escalated this issue and for the following XS iteration there was a preliminary fix available. An improved solution has now been adopted into the mainstream version of the Emerging Threats (ET) rules [37]. ET rules now contain a metadata field called *attack_target*, which reveals the victim-side counterpart of the attack. Currently there are approximately 15,000 rules which contain the *attack_target* metadata keyword. This development has been presented at security conferences such as Hack.Lu [38] and SuriCon [39].

Peek now uses those keywords and enriches each atomic message with metadata to determine event shipper and, if applicable, proper event source and attack target. Our exercise asset collection tool enables us to build an asset database for threat hunting and map individual addresses from alerts to known assets. Source and destination information is inserted to every message metadata, along with event directionality flag (i.e., inbound, outbound, lateral, or local). The post-processing script is then able to process metadata-enriched messages and report affected assets to the feedback dashboards.

V. Visualizations

During XS, numerous large screens are installed in the training room directed at the OCO team. Their purpose is to provide visual feedback from various tools taking up any of the valuable screen real estate from the training audience.

Frankenstack comprises a set of open-source tools for visualizing log data, time-series metrics, and alerts. There are slight differences in handling various types of alerts: for example, alerts for CPU and memory usage trigger and recover automatically based on a predefined set of thresholds, while security events (e.g., IDS/IPS alerts) are only triggered based on some detection condition but lack the concept of a recovery threshold. Thus, such security alerts will never receive a recovery event, leading to an overabundance of information on the central dashboard. This requires special handling and conditions for timing out stale alerts.

Correlation and deduplication of recurring events is crucial for creating useful visualizations. Due to the volatile nature of CSXs, visualization tools can overflow with too much information for users to follow. For example, a network scan using the *nmap* tool can trigger a large amount of security events over a short period of time. While event correlation can collect and continuously combine those events, it should not wait indefinitely for the scanning to end before emitting the alert to the dashboard. The aim is to notify the OCO of their activity as it happens. Therefore, the length of the correlation window must be kept relatively short. With an effective deduplication functionality, sending the same alert multiple times does not cause any issues.

Alerta [34] is used as the primary feedback dashboard to present any malicious activity to exercise participants. The entire feedback cycle is completed by emitting the enriched event from the correlation engine to the Alerta dashboard. Each OCO team member has access to the Alerta API and web interface to create personal filtering rules for limiting the displayed information only to what is relevant in the current stage of the attack. We set a timeout to automatically archive stale events that had no correlated activity in the last 15 minutes to avoid flooding the dashboard with events.

Kibana [36] was used to present analytical dashboards that provided insight over the entire duration of the exercise, not just the recent events view available in Alerta. The information included a summary of detected attack types and statistics of IP addresses that have been generating the most alerts. The dashboards on the large TV screens were often observed by WT members who were more interested in the progress of the exercise and overall performance of the training audience.

We also mapped all feedback events to the MITRE ATT&CK framework attack phases and techniques. This enabled us to integrate the MITRE ATT&CK Navigator application to our environment and visualize the security events the OCO team had triggered to attack the target BT environment.

VI. Testbed for vendor appliances

Although Frankenstack has been kept open source, we have not denied cooperation with existing security platforms, SIEM systems, or commercial vendor appliances. Over the years several security vendors (e.g., Cymmetria [40], Greycortex [41], and Stamus Networks [42]) have joined the exercise YT to test their commercial products in a unique live-fire environment. We do not treat any proprietary security product as a component of Frankenstack, but rather as another monitoring data feed that can provide a different perspective into the exercise dataset.

VII. Future work

Several issues have remained a challenge. For example, to assess the progress of the OCO team more systematically, their attack campaigns would have to be incorporated into the exercise scenario itself. The exercise scenario and timeline would also have to be available in a machine-readable format so that Frankenstack could follow exercise progress and potentially adapt to non-technical scenario changes that take place as part of the storyline (e.g., dynamically adjust the level of detail provided within the feedback based on the participants' progress in the campaign).

More advanced visualizations are required for the exercise training audience and organizers to better follow the participants' progress at a higher level. The focus of Frankenstack

has been on technical feedback rather than information that would provide actionable SA for higher-level decision makers. We attempted this with EVE [43] but faced the problem of revealing too much information too early in the game, which unfortunately limited its usability during the exercise.

VIII. Conclusion

The paper outlines the new developments of the cyberattack detection and feedback framework, *Frankenstack*. This open-source cyber-exercise-specific framework is based on a combination of various open-source monitoring tools. The primary purpose of Frankenstack is to provide detection of malicious activity and fully automated real-time observations during cyber exercises where the emphasis on training the offensive skillset.

The work describes the updated technical architecture compared to an earlier version of the framework. Furthermore, improved data processing, distributed event streaming, and feedback dashboards are described. Since 2017, we have implemented and verified the performance of our framework in the annual NATO CCD COE's Crossed Swords cyber exercise.

IX. Acknowledgements

The authors would like to thank Dr. Risto Vaarandi and Dr. Bernhards Blumbergs for their valuable advice.

This work has been supported by the Estonian IT Academy (StudyITin.ee).

References

[1] NATO CCD COE, "Crossed Swords Exercise," Available: https://ccdcoe.org/exercises/crossed-swords/, 2021.

[2] M. Kont, M. Pihelgas, K. Maennel, B. Blumbergs, and T. Lepik, "Frankenstack: Toward real-time Red Team feedback," in *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, October 2017, pp. 400–405.

[3] M. Pihelgas, *Mitigating Risks arising from False-Flag and No-Flag Cyber Attacks*. NATO CCD COE Publications, 2015.

[4] N. Känzig, R. Meier, L. Gambazzi, V. Lenders, and L. Vanbever, "Machine Learning-based Detection of C&C Channels with a Focus on the Locked Shields Cyber Defense Exercise," in *2019 11th International Conference on Cyber Conflict (CyCon)*, vol. 900, 2019, pp. 1–19.

[5] J. Klein, S. Bhulai, M. Hoogendoorn, R. Van Der Mei, and R. Hinfelaar, "Detecting Network Intrusion beyond 1999: Applying Machine Learning Techniques to a Partially Labeled Cybersecurity Dataset," in *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, 2018, pp. 784–787.

[6] D. L. Arendt, D. Best, R. Burtner, and C. L. Paul, "CyberPetri at CDX 2016: Real-time network situation awareness," in *2016 IEEE Symposium on Visualization for Cyber Security (VizSec)*, October 2016, pp. 1–4.

[7] D. L. Arendt, R. Burtner, D. M. Best, N. D. Bos, J. R. Gersh, C. D. Piatko, and C. L. Paul, "Ocelot: user-centered design of a decision support visualization for network quarantine," in *2015 IEEE Symposium on Visualization for Cyber Security (VizSec)*, October 2015, pp. 1–8.

[8] D. S. Henshel, G. M. Deckard, B. Lufkin, N. Buchler, B. Hoffman, P. Rajivan, and S. Collman, "Predicting proficiency in cyber defense team exercises," in *MILCOM 2016 - 2016 IEEE Military Communications Conference*, November 2016, pp. 776–781.

[9] K. Maennel, R. Ottis, and O. Maennel, "Improving and Measuring Learning Effectiveness at Cyber Defense Exercises," in *Secure IT Systems: 22nd Nordic Conference, NordSec 2017, Tartu, Estonia, November 8-10, 2017. Proceedings*, November 2017, pp. 123–138.

[10] M. Ernits, K. Maennel, S. Mäses, T. Lepik, and O. Maennel, "From Simple Scoring Towards a Meaningful Interpretation of Learning in Cybersecurity Exercises," in *15th International Conference on Cyber Warfare and Security (ICCWS 2020)*, March 2020.

[11] K. Maennel, J. Kim, and S. Sütterlin, "From Text Mining to Evidence Team Learning in Cybersecurity Exercises," in *Companion Proceedings 10th International Conference on Learning Analytics and Knowledge (LAK20)*, March 2020.

[12] The MITRE Corporation, "MITRE ATT&CK," Available: https://attack.mitre.org/, 2021.

[13] M. Chmelař, "Utilizing MITRE ATT&CK to Create Adversary Reports of Live-Fire Cybersecurity Exercises for Feedback Purposes," Tallinn University of Technology, Tech. Rep., 2020.

[14] Open Information Security Foundation, "Suricata," Available: https://suricata-ids.org/, 2021.

[15] Arkime, "Arkime," Available: https://arkime.com/, 2021.

[16] Microsoft, "Windows Sysinternals - Sysmon," Available: https://technet.microsoft.com/en-us/sysinternals/sysmon, 2021.

[17] O. Hartong, "sysmon-modular," Available: https://github.com/olafhartong/sysmon-modular, 2021.

[18] SwiftOnSecurity, "sysmon-config," Available: https://github.com/SwiftOnSecurity/sysmon-config, 2021.

[19] B. S. Jese, "Snoopy Logger," Available: https://github.com/a2o/snoopy, 2021.

[20] Red Hat, Inc, "Ansible," Available: https://www.ansible.com/, 2021.

[21] B. Blumbergs, M. Pihelgas, M. Kont, O. Maennel, and R. Vaarandi, "Creating and Detecting IPv6 Transition Mechanism-Based Information Exfiltration Covert Channels," in *Secure IT Systems: 21st Nordic Conference, NordSec 2016, Oulu, Finland, November 2-4, 2016. Proceedings*. Springer International Publishing, 2016, pp. 85–100. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-47560-8_6

[22] One Identity LLC, "syslog-ng," Available: https://www.syslog-ng.com/, 2021.

[23] Elasticsearch B.V., "Elasticsearch," Available: https://www.elastic.co/elasticsearch, 2021.

[24] Apache Software Foundation, "Apache Kafka," Available: https://kafka.apache.org/, 2021.

[25] R. Vaarandi, B. Blumbergs, and E. Çalışkan, "Simple event correlator - Best practices for creating scalable configurations," in *Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2015 IEEE International Conference on*, March 2015, pp. 96–100.

[26] NATO CCD COE, "frankenSEC," Available: https://github.com/ccdcoe/frankenSEC, 2019.

[27] Adiscon GmbH, "The rocket-fast Syslog Server," Available: https://www.rsyslog.com/, 2021.

[28] Elasticsearch B.V., "Logstash," Available: https://www.elastic.co/logstash, 2021.

[29] NATO CCD COE, "Peek," Available: https://github.com/ccdcoe/go-peek, 2020.

[30] F. Roth, "Sigma," Available: https://github.com/Neo23x0/sigma, 2021.

[31] "The Go Programming Language," Available: https://golang.org/, 2021.

[32] M. Kont, "Sigma rule engine," Available: https://github.com/markuskont/go-sigma-rule-engine, 2020.

[33] M. Kont and M. Pihelgas, *IDS for logs: Towards implementing a streaming Sigma rule engine*. NATO CCD COE Publications, 2020.

[34] N. Satterly, "Alerta," Available: http://alerta.io/, 2021.

[35] The MITRE Corporation, "Att&ck navigator," Available: https://github.com/mitre-attack/attack-navigator, 2021.

[36] Elasticsearch B.V., "Kibana," Available: https://www.elastic.co/kibana, 2021.

[37] Proofpoint, "Emerging Threats rules," Available: https://rules.emergingthreats.net/, 2020.

[38] E. Leblond, "Finding the Bad Guys, Yes Really," Available: https://www.youtube.com/watch?v=Scntdv1Vp_0, 2017, hack.lu 2017 Presentation.

[39] ——, "Finding the Bad Guys, Yes Really," SuriCon 2017, 2017, presentation.

[40] Cymmetria, "Cyber deception & NATO red teams," Available: https://cymmetria.com/white-paper/nato-crossed-swords/, 2018.

[41] Greycortex, "Greycortex supports Crossed Shields for second year," Available: https://www.greycortex.com/blog/greycortex-supports-crossed-shields-second-year, 2019.

[42] Stamus Networks, "Stamus Networks at XS20," Available: https://twitter.com/StamusN/status/1339968510924120066, 2021.

[43] F. J. R. Melón, T. Väisänen, and M. Pihelgas, "EVE and ADAM: Situation Awareness Tools for NATO CCDCOE Cyber Exercises," in *STO-MP-SCI-300 Cyber Physical Security of Defense Systems*, 2018, pp. 10–1–10–15. [Online]. Available: https://ccdcoe.org/uploads/2018/10/EVE-ADAM-MP-SCI-300-10.pdf

Curriculum Vitae

1. Personal data

Name	Mauno Pihelgas
Date and place of birth	3 January 1988, Haapsalu, Estonia
Nationality	Estonian

2. Contact information

Address	Tallinn University of Technology, School of Information Technologies, Department of Software Science, Ehitajate tee 5, 19086 Tallinn, Estonia
E-mail	info[at]pihelgas.eu

3. Education

2014–2021	Tallinn University of Technology, School of Information Technologies, Cyber Security, PhD studies
2010–2012	Tallinn University of Technology, Faculty of Information Technology, Cyber Security, MSc *cum laude*
2010–2012	University of Tartu, Faculty of Science and Technology, Cyber Security, MSc *cum laude*
2007–2010	Estonian Information Technology College, IT Systems Development, Diploma *cum laude*

4. Language competence

Estonian	native
English	fluent
Russian	basic level
German	basic level

5. Professional employment

2013– …	NATO Cooperative Cyber Defence Centre of Excellence, Technology Researcher
2012–2020	Tallinn University of Technology, Computer Lab Assistant
2010–2013	Elion Ettevõtted AS, Monitoring Administrator
2008–2010	Microlink Eesti AS, Data Centre Duty Technician

6. Certifications

2020–2023	Red Hat Certified Specialist in Advanced Automation: Ansible Best Practices
2017–2023	Red Hat Certificate of Expertise in Ansible Automation Exam (Ansible 2)
2016–2024	GIAC Continuous Monitoring Certification (GMON)
2014–2023	Red Hat Certified System Administrator (RHEL7)
2014–2023	Red Hat Certified Engineer (RHEL7)

7. Voluntary work

2011	Vaata Maailma Foundation, Restoration of donated computers for charity
2012–2020	Robotex, Sumo Robot workshop instructor

8. Computer skills

- Operating systems: GNU/Linux, MS Windows
- Document preparation: Vim, MS Code, LaTeX, Libre Office, MS Word
- Programming languages: Python, Bash, PHP, Perl, Go
- Scientific packages: Jupyter Notebooks, JupyterLab

9. Defended theses

- 2012, A Comparative Analysis of Open-Source Intrusion Detection Systems, MSc, supervisor Dr. Risto Vaarandi, Tallinn University of Technology
- 2010, Expanding Functionality of the Robot Control Platform of The Estonian Information Technology College, Diploma, supervisor Margus Ernits, Estonian Information Technology College

10. Field of research

- Security Monitoring
- Situation Awareness
- Cyber Security Exercises
- Log Analysis

11. Scientific work
Papers

1. R. Vaarandi and M. Pihelgas. Using Security Logs for Collecting and Reporting Technical Security Metrics. In *Military Communications Conference (MILCOM), 2014 IEEE*, pages 294–299, October 2014
2. R. Vaarandi and M. Pihelgas. LogCluster - A data clustering and pattern mining algorithm for event logs. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 1–7, November 2015
3. R. Vaarandi, M. Kont, and M. Pihelgas. Event log analysis with the LogCluster tool. In *MILCOM 2016 - 2016 IEEE Military Communications Conference*, pages 982–987, November 2016
4. B. Blumbergs, M. Pihelgas, M. Kont, O. Maennel, and R. Vaarandi. Creating and Detecting IPv6 Transition Mechanism-Based Information Exfiltration Covert Channels. In *Secure IT Systems: 21st Nordic Conference, NordSec 2016, Oulu, Finland, November 2-4, 2016. Proceedings*, pages 85–100. Springer International Publishing, 2016
5. M. Kont, M. Pihelgas, K. Maennel, B. Blumbergs, and T. Lepik. Frankenstack: Toward real-time Red Team feedback. In *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, pages 400–405, October 2017
6. M. Pihelgas. Design and Implementation of an Availability Scoring System for Cyber Defence Exercises. In *14th International Conference on Cyber Warfare and Security (ICCWS 2019)*, page 329–337, 2019

7. P. Théron, A. Kott, M. Drašar, K. Rzadca, B. LeBlanc, M. Pihelgas, L. Mancini, and A. Panico. Towards an active, autonomous and intelligent cyber defense of military systems: The NATO AICA reference architecture. In *2018 International Conference on Military Communications and Information Systems (ICMCIS)*, pages 1–9, May 2018

8. P. Théron, A. Kott, M. Drašar, K. Rzadca, B. LeBlanc, M. Pihelgas, L. Mancini, and F. de Gaspari. Reference Architecture of an Autonomous Agent for Cyber Defense of Complex Military Systems. In *Adaptive Autonomous Secure Cyber Systems*, pages 1–21, Cham, 2020. Springer International Publishing

9. A. Kott, P. Théron, L. V. Mancini, E. Dushku, A. Panico, M. Drašar, B. LeBlanc, P. Losiewicz, A. Guarino, M. Pihelgas, and K. Rzadca. An introductory preview of Autonomous Intelligent Cyber-defense Agent reference architecture, release 2.0. *The Journal of Defense Modeling and Simulation*, 17(1):51–54, 2020

10. R. Vaarandi and M. Pihelgas. NetFlow Based Framework for Identifying Anomalous End User Nodes. In *15th International Conference on Cyber Warfare and Security (ICCWS 2020)*, page 448–456, 2020

11. M. Pihelgas and M. Kont. Frankenstack: Real-time Cyberattack Detection and Feedback System for Technical Cyber Exercises. In *2021 IEEE CSR Workshop on Cyber Ranges and Security Training (CRST)*. IEEE, July 2021. (Accepted paper)

Conference presentations

1. **M. Pihelgas**. 'Security Metrics - Background Study and Suggestions for Organizational Networks and IT Systems', SAS-106 Symposium on Analysis Support to Decision Making in Cyber Defence & Security: 9–10 June 2014, Tallinn, Estonia

2. B. Blumbergs, **M. Pihelgas**. 'Creating and Detecting IPv6 Transition Mechanism-Based Information Exfiltration Covert Channels', 21st Nordic Conference on Secure IT Systems (Nordsec 2016): 2–4 November 2016, Oulu, Finland

3. **M. Pihelgas**, M. Kont. 'Hedghehog in the Fog: Creating and Detecting IPv6 Transition Mechanism-Based Information Exfiltration Covert Channels', CERT-EE Symposium 2017: 29–30 May 2017, Tallinn, Estonia

4. **M. Pihelgas**. 'Scoring a Technical Cyber Defense Exercise with Nagios and Selenium', 13th Open Source Monitoring Conference (OSMC 2018): 5–8 November 2018, Nuremberg, Germany

5. M. Kont, **M. Pihelgas**. 'Frankenstack. Busting the Red Team with Duct Tape, Spit and Tears', 5th Annual SuriCon: 30 October–01 November 2019, Amsterdam, The Netherlands

6. **M. Pihelgas**. 'Frankenstack: Real-time Cyberattack Detection and Feedback System for Technical Cyber Exercises', 2021 IEEE CSR Workshop on Cyber Ranges and Security Training (CRST): 26 July 2021, Online (Accepted presentation)

Elulookirjeldus

1. Isikuandmed

Nimi	Mauno Pihelgas
Sünniaeg ja -koht	03.01.1988, Haapsalu, Eesti
Kodakondsus	Eesti

2. Kontaktandmed

Aadress	Tallinna Tehnikaülikool, Tarkvarateaduse Instituut, Ehitajate tee 5, 19086 Tallinn, Estonia
E-post	info[ät]pihelgas.eu

3. Haridus

2014–2021	Tallinna Tehnikaülikool, infotehnoloogia teaduskond, Küberkaitse, doktoriõpe
2010–2012	Tallinna Tehnikaülikool, infotehnoloogia teaduskond, Küberkaitse, MSc *cum laude*
2010–2012	Tartu ülikool, Matemaatika-informaatika teaduskond, Küberkaitse, MSc *cum laude*
2007–2010	Eesti Infotehnoloogia Kolledž, IT süsteemide arendus, rakenduskõrgharidus

4. Keelteoskus

eesti keel	emakeel
inglise keel	kõrgtase
vene keel	algtase
saksa keel	algtase

5. Teenistuskäik

2013– …	NATO Küberkaitse Koostöö Kompetentsikeskus, Teadur-nõunik
2012–2020	Tallinna Tehnikaülikool, Praktikumi assistent
2010–2013	Elion Ettevõtted AS, Monitooringu administraator
2008–2010	Microlink Eesti AS, Serverikeskuste valvetehnik

6. Erialased sertifikaadid

2020–2023	Red Hat Certified Specialist in Advanced Automation: Ansible Best Practices
2017–2021	Red Hat Certificate of Expertise in Ansible Automation Exam (Ansible 2)
2016–2024	GIAC Continuous Monitoring Certification (GMON)
2014–2023	Red Hat Certified System Administrator (RHEL7)
2014–2023	Red Hat Certified Engineer (RHEL7)

7. Vabatahtlik töö

2011	Vaata Maailma Foundation, Vanade arvutite taastamine heategevuseks
2012–2020	Robotex, Sumorobotite töötoa juhendaja

8. Computer skills

- Operatsioonisüsteemid: GNU/Linux, MS Windows
- Kontoritarkvara: Vim, MS Code, LaTeX, Libre Office, MS Word
- Programmeerimiskeeled: Python, Bash, PHP, Perl, Go
- Teadustarkvara paketid: Jupyter Notebooks, JupyterLab

9. Kaitstud lõputööd

- 2012, Võrdlusanalüüs vabatarkvaralistest ründetuvastussüsteemidest, MSc, juhendaja Dr. Risto Vaarandi, Tallinna Tehnikaülikool
- 2010, Eesti Infotehnoloogia Kolledži roboti juhtimisplatvormi funktsionaalsuse laiendamine, juhendaja Margus Ernits, Eesti Infotehnoloogia Kolledž

10. Teadustöö põhisuunad

- Monitooring
- Situatsiooniteadlikkus
- Küberharjutused
- Logianalüüs

11. Teadustegevus
Teadusartiklite, konverentsiteeside ja konverentsiettekannete loetelu on toodud inglise-keelse elulookirjelduse juures.

ISSN 2585-6901 (PDF)
ISBN 978-9949-83-719-9 (PDF)