TALLINN UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

Department of Computer Science

TUT Centre for Digital Forensics and Cyber Security

ITC70LT

Mina Gerges - 114381IVCMM

LOG MONITORING AND EVENT CORRELATION ON MICROSOFT® WINDOWS™ USING SIMPLE EVENT CORRELATOR

Master thesis

Supervisor: Risto Vaarandi

Doctor of Philosophy

Senior Researcher

Tallinn 2016

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Arvutiteaduse instituut

TTÜ Küberkriminalistika ja Küberjulgeoleku Keskus

ITC70LT

Mina Gerges - 114381IVCMM

LOGIDE MONITOORING JA SÜNDMUSTE KORRELATSIOON MICROSOFT® WINDOWS™ PLATVORMIL SEC KORRELATSIOONIMOOTORI ABIL

Magistritöö

Juhendaja: Risto Vaarandi

Filosoofiadoktor

Vanemteadur

Tallinn 2016

Declaration

I hereby declare that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Mina Gerges

……………………………………. ………………………………

(Date) (Signature)

Abstract

The aim of this thesis is providing a stable integration solution for Simple Event Correlator (SEC) Perl process to run on Microsoft Windows operating system as a service. The purpose of running SEC on Microsoft Windows is providing a reliable correlation engine to monitor event logs of the operating system and other software for critical alerts, malicious action, attack patterns and other anomalies, then to interact proactively.

SEC is coded using Perl. However Microsoft does not provide a native Perl distribution for Windows. A third party Perl distribution from the available shall be selected, based on features and performance comparison. SEC is not capable of loading Microsoft Windows event logs without the aid of a log collection tool. Miscellaneous applications provide different log formats which require normalization to standardize parsing for extracting information from events.

Features comparisons for commonly used event monitoring tools, correlation engines, log collection tools and Windows Perl distribution were conducted. The author has developed a Microsoft Windows application to wrap SEC Perl process in a Windows service with a user interface to configure SEC command parameters. Example SEC rulesets are provided as a proof of concept.

Using correlation engine on Microsoft Windows adds an extra dimension of security and provides assistance to system administrators for detecting critical incidents and other anomalies. Additionally it can be used for various tasks like software test automation.

This thesis is written in English and is 101 pages long, including 4 chapters and 30 figures.

Annotatsioon

LOGIDE MONITOORING JA SÜNDMUSTE KORRELATSIOON MICROSOFT® WINDOWS™ PLATVORMIL SEC KORRELATSIOONIMOOTORI ABIL

Selle töö eesmärgiks on luua stabiilne integratsioonilahendus SEC (Simple Event Correlator) rakenduse kasutamiseks Microsoft Windows platvormil. Töö kirjeldab SECi kasutamist Microsoft Windows keskkonnas korrelatsioonimootorina, mis jälgib operatsioonisüsteemi ja rakenduste logisid, avastamaks kriitilisi süsteemivigu, pahatahtlikke tegevusi, ründeid ja muid anomaalseid sündmuseid, ning reageerimaks avastatud sündmustele.

SEC on kirjutatud Perlis, kuid Microsoft Windows platvormil puudub standardne Microsofti poolt loodud Perli distributsioon. Kuna SECil puudub vahetu tugi Microsoft Windowsi logide lugemiseks ja logiformaatide rohkuse tõttu nõuavad logisündmused normaliseerimist, vajab SEC logide kogumiseks ja teisendamiseks eraldi tööriista.

Töös võrreldakse erinevaid Windowsi jaoks loodud Perli distributsioone, logide kogumise tööriistu, aga ka logisündmuste korrelatsiooni ja monitooringu tööriistu. Võrdluse põhjal näidatakse, et SEC on logisündmuste korrelatsiooniks sobivaim, ja samuti valitakse võrdluse põhjal logide kogumise tööriist ning SECi jaoks vajalik Perli distributsioon. Töö peamiseks tulemuseks on autori poolt loodud integratsioonilahendus SECwin, mis võimaldab SECil Microsoft Windows platvormil teenusena töötada ning loob kasutajale mugava interfeisi SECi häälestamiseks ja juhtimiseks. Töös esitatakse ka mõned SECi näidisreeglid, mis on mõeldud Microsoft Windowsi logide monitooringuks ja korrelatsiooniks.

Loodud integratsioonilahenduse abil Microsoft Windows platvormil SECi kasutamine suurendab süsteemi turvalisust ja annab süsteemiadministraatoritele täiendava võimaluse turvaintsidentide ning anomaalsete sündmuste tuvastamiseks. Loodud lahendust on võimalik kasutada ka teistel eesmärkidel, nagu näiteks tarkvaratestimise automatiseerimine.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 101 leheküljel, 4 peatükki, 30 joonist.

List of acronyms and terms

API	Application Program Interface
BSD	Berkeley Software Distribution
CA	Certificate Authority
CEE	Common Event Expression
CEP	Complex Event Processing
CLI	Command Language/Line Interface
CPU	Central Processing Unit
CRL	Certificate Revocation List
C#	C sharp – Programming language
DLL	Dynamic Link Library
DMZ	Demilitarized Zone
EPM	End Point Management
GB	Giga Byte
GELF	Graylog Extended Log Format
GHz	Giga Hertz
IETF	Internet Engineering Task Force
IIS	Internet Information Services
I/O	Input / Output
IP	Internet Protocol
IT	Information Technology

ITIL	Information Technology Infrastructure Library
ITMS	Information Technology Management Suite
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
LMS	Log Management System
LTE	Log Template Extraction
MS	Microsoft
MSI	Microsoft Installer package
OCSP	Online Certificate Status Protocol
OS	Operating System
OWASP	Open Web Application Security Project
PE	Portable Executable
PHP	Hypertext Preprocessor / Personal Home Page
PID	Process Identification number
PPM	Perl Package Management
RAID	Redundant Array of Inexpensive Disks
RAM	Random Access Memory
REST	Representational State Transfer
RFC	Request For Comments
SEC	Simple Event Correlator
SID	Security Identification

SECwin	Simple Event Correlator Windows Integration
SIEM	Security Information and Event Management
SIGABRT	Signal Abort
Sigcheck	Signature Check tool
SIGHUP	Signal Hangup
SIGINT	Signal Interrupt
SIGTERM	Signal Terminate
SIGUSR1	Signal User defined 1
SIGUSR2	Signal User defined 2
SSD	Solid State Drive
SSL	Secure Socket Layer
SQL	Standard Query Language
Sysmon	System Monitor tool
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URI	Uniform Resource Identifiers
URL	Uniform Resource Locator
VB	Visual Basic
VM	Virtual Machine
XML	Extensible Markup Language

Acknowledgments

The author would like to express his gratitude to Risto Vaarandi for his precious advice and mentoring of this thesis, and for his efforts and contributions in the domain of log mining and event correlation including the development of Simple Event Correlator.

Table of Contents

Table of Figures

1. Introduction

Whether on home-user or corporate level, Microsoft Windows is the most commonly used desktop/laptop operating system [1]. While Anti-Malware applications vary on their level of protection, the need to monitor various logs for critical alerts handling, malicious actions and attack pattern identification and then to interact proactively still arise. On corporate level, the collection of event logs and log files form a very essential part of network management, security assessment and forensics activities. The vast amount of events data can be overwhelming and renders the process of finding information quite difficult, thus log management and event correlation are essential for administration tasks and providing an insight of incidents within a network.

In order to detect technical issues affecting productivity or security threats and proactively trigger an action or alert, an event correlation solution is required. Event correlation application matches events according to predefined schemes and patterns in sequence within a period of time, in order to identify a situation and then triggers a predefined action. A more advanced approach is implementing a centralized dedicated infrastructure to perform monitoring, detection and log collection to have a wider image of problems occurring on different nodes which might be related.

1.1. Problem statement

Many small to medium sized corporates depend on Microsoft products and do not assign resources for UNIX operating systems to eliminate the cost required for human resources to administrate such infrastructure. Unfortunately many corporates and home users do not deploy a solution for log monitoring and malicious action identification [2], and mainly depend on a single anti-malware solution.

The problem addressed within this thesis is providing a free and easily deployable solution for small to medium sized corporates and home users, for log monitoring and proactive event correlation. Event correlation assists on malicious actions detection, attack patterns identification, and handling critical alerts by triggering predefined actions.

1.1.1. Problems related to running SEC on MS Windows

Although Simple Event Correlator (SEC) [3] is based on Perl which renders it cross platform, the following problems to run it on Microsoft Windows apply:

- Microsoft Windows does not integrate a native Perl engine [4], therefore several different Windows Perl distributions have been developed by third parties.
- SEC requires MS Windows service to run on boot without user interaction.
- Previous attempt to run SEC as a service had several draw backs, covered in section 2.3.6.
- SEC is a Perl script application, initiated using command line, lacking a user interface to provide an easy and fast way to set the required command parameters.

1.1.2. Problems related to alternative solutions

While several log monitoring solutions capable of running on MS Windows OS are available, they mainly focus on providing analytical reports, logs archiving and alerts generating, with very basic event correlation functionality. Most commercial event management solutions are costly and target enterprise level. Free and open source solution might lack features or flexibility. Event management solutions are centralized and database driven, as a result they consume more computing, human and financial resources. For these reasons, if the need is for threat incidents identification in real-time, a correlation engine shall be used.

1.1.3. Problems related to events format

Another problem is the variety of log formats and formatting. Different vendors adopt different log formats, such as, XML, Snare, BSD syslog, IETF syslog, JSON etc. Even within same log format, different applications might build the raw event in different ways, as an example, an xml event can have its fields in xml attributes or nodes. That increases the difficulty of the process for extracting fields’ values from raw events. Thus a standard format shall be adopted to facilitate the process of parsing logs. This problem is addressed by log normalization.

Logs are written by developers, and each application has its own message formatting, which prevents the ability to generalize event correlation and detection patterns across different software. An example of different formatting styles is the message “Login failed”, which can be seen as “Invalid login attempt”. Addressing such problem would be by devoting specific correlation rules to each solution.

1.2. Contribution

This research intends to provide a reliable distributed, centralized or hybrid event correlation implementation for small to medium sized corporates and home users running MS Windows operating system using Simple Event Correlator - a well-known mature correlation engine based on Perl.

An analytical comparison was conducted by the author covering the most common correlation engines, log monitoring and alerting systems capable of running on Microsoft Windows platform. Performing a test run and efficiency measuring for each solution’s set of features or performance analysis is out of the scope of this thesis. Therefore a comparative features analysis as per those solutions’ producers claim is considered to be satisfactory for the conducted study. However a more detailed analytical comparison was conducted between two free and claimed to be similar solutions, NXLog-ce correlation module *pm_evcorr* [5] and Simple Event Correlator [3].

Simple Event Correlator has been chosen by the author as the default correlation engine for its wide acceptance academically and industrially, flexibility, capabilities, maturity and being free of cost with open source. Although SEC is technically a Perl script, which renders it cross platform, MS Windows is known to lack a native Perl engine [4], resulting in third party developed Perl engines. The author conducted a comparison of features and performance between common MS Windows Perl distributions. The comparison promoted the use of Cygwin Perl for its UNIX system features emulation, which some SEC features depend on and higher performance, especially for disk I/O operations. In addition other available Cygwin binaries can be used for SEC actions to avoid additional software installation or writing additional scripts.

A previous attempt to run SEC as a MS Windows service took place in 2008 by one of SEC community members and has been published via SEC mailing list [6]. Unfortunately, that attempt was not of much success, as covered with more details in section 2.3.6. Accordingly, the need for developing a new tool to wrap SEC Perl process as a native MS Windows service became an essential requirement to run SEC on MS Windows flawlessly. Therefore, a new tool has been developed by the author using Microsoft .NET framework C# syntax. The installer package of the developed tool includes SEC and selective Cygwin binaries provided as a MSI. The developed application – SECwin, has been already published for general availability since June 2015 on GitHub and SourceForge. According to both sites statistics till end of April 2016, the application has been downloaded 114 times, from different worldwide locations [7] [8].

SECwin provides additional features, including:

- User interface to configure SEC parameters
- Sending interrupt signals to SEC process via user interface
- Auto-update for both of the packages, SEC and SECwin
- SEC statistics dump file rotation
- SEC Perl process watchdog
- Service status control and monitor
- SEC logs and statistics dump viewer for faster access
- Supportability for other Perl distributions
- Log Perl process statistics for CPU time, memory, loaded modules etc.

The package is available as a single MSI file, which can be either installed manually or distributed using MS group policy or equivalent for corporate usage. Furthermore the application has been compacted as a MS Windows service, Windows forms and console application all in one portable executable with no dependencies. In case a system has been loaded earlier with a Perl distribution, SECwin PE only would be required.

To facilitate logs collection and transfer over network as required, NXLog community edition has been chosen as a free logs processing tool. The author provided full configuration set for NXLog-ce to facilitate the implementation of the solution. The author as well covered the details for creating the required SSL digital certificates chain to secure the transportation of logs over the network using TLS protocol.

A proof of concept rulesets for SEC are provided for proactive event monitoring and event correlation on MS Windows. The rulesets are available online with hope of being extended by SEC community members.

2. Overview of existing solutions and related work

This chapter provides an overview of current common academic and journal published papers on log monitoring and proactive event correlation and comparison of available log monitoring solutions capable of running on MS Windows operating system.

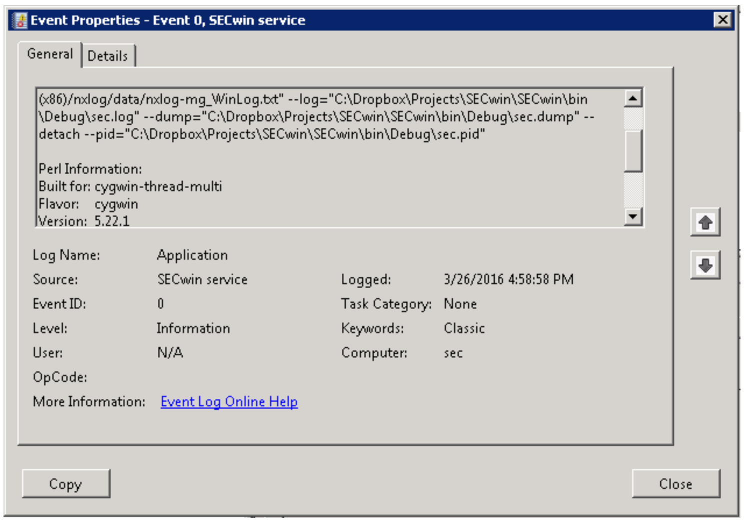
2.1. Events

An event in the context of IT is a record of incident or reporting of a status that occurs at a specific time and provides a concise description what has happened. This information is usually used for finding anomalies, detecting security threats, records for auditing, measuring performance, applications debugging and profiling etc. Hence, event logging plays an essential role in systems administration, security auditing and forensics in addition to software and systems troubleshooting. OWASP foundation has set the basic attributes for an event as: When, Where, Who, What and additional considerable records [9], in their logging cheat-sheet, in which, they have also provided many other considerations, like which events to log and where to log.

A log entry is a single event, which has been recorded and saved, thus logging is the process of registering log entries to record an event. This research focuses on Microsoft Windows platform logs whether generated by the operating system, its native applications or other third party applications running on it. Usually applications running on MS Windows write their logs either using Windows event log API or in text files stored on disk, which is usually in custom format and occasionally events are also recorded in databases. Microsoft defines an event as, any significant occurrence in the computer or in a program that requires either users to be notified or an entry added to a log [10]. Microsoft Windows event logs [11] are classified to five severities: Informational, Warning, Error, Audit Success and Audit Failure, where the latest two are security events specific. MS Windows events are based on categories known as *log name*. There are several default log names, however only few are enabled by default, which represents critical logging domains such as:

- Application: contains events from Generic application, events are classified as informational, warning or error.
- Security: contains security audit events which are classified as success or failure
- Setup: contains events about MS Windows native applications setup, such as roles, features, windows updates etc.
- System: contains events related to internal system operations and MS Windows system services operation.

Event log settings can be modified using *Group Policy Management Console* under the path *(Computer Configuration\Windows Settings\Security Settings\Event Log\)*. Fine tuning is usually required to enable more detailed logging, where changes can be applied only for MS Windows native log names. Additionally, a tool like *Sysmon* [12] - a background service that logs security-relevant process and network activities to Windows event log, can be used to extend security audit logging. Any program can write to MS Windows event log, usually to *Application* category; however a new log name can be easily created to hold log data from specific program. Similarly an event can be written by using *eventcreate* [13] console command.



```
(x86)/nxlog/data/nxlog-mg_WinLog.txt" --log="C:\Dropbox\Projects\SECwin\SECwin\bin
\Debug\sec.log" --dump="C:\Dropbox\Projects\SECwin\SECwin\bin\Debug\sec.dump" --
detach --pid="C:\Dropbox\Projects\SECwin\SECwin\bin\Debug\sec.pid"

Perl Information:
Built for: cygwin-thread-multi
Flavor:    cygwin
Version:  5.22.1
```

Log Name:	Application		
Source:	SECwin service	Logged:	3/26/2016 4:58:58 PM
Event ID:	0	Task Category:	None
Level:	Information	Keywords:	Classic
User:	N/A	Computer:	sec
OpCode:			
More Information:	Event Log Online Help		

Figure 1. Windows event log entry

Figure 1 demonstrates a screen shot of a windows log event, displayed using MS Windows native *Event Viewer*.

2.1.1. Event format

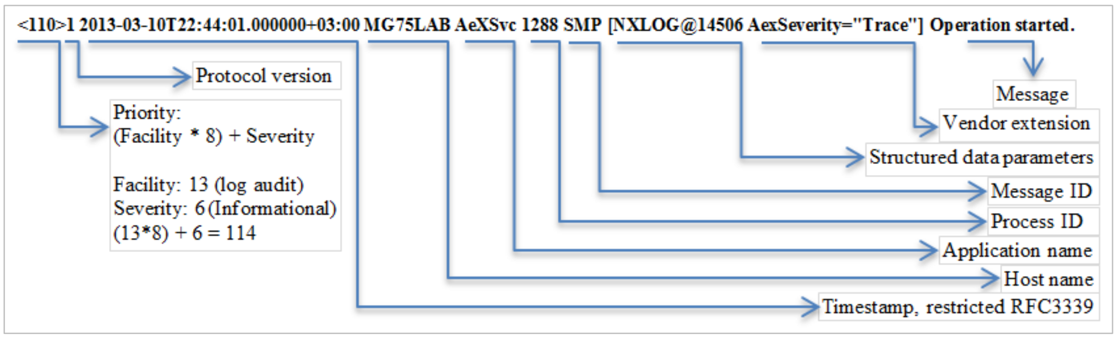
The variety of log formats has been always an issue for collecting and correlating events of different applications. There have been several approaches and extensive efforts by researchers to resolve such problem, whether by defining log mining algorithms, defining log normalization protocols or attempts to automate log normalization and utilizing generic log format protocols. Practically the issue remains, especially when attempting to parse custom event formats as a result of the lack of governing rules to evaluate applications' events format compliance against set of unified standards.

Most commercial and open source solutions, which perform log analysis, depend on their preconfigured raw event structure parser. That means supporting undefined log formats requires human interaction to define a newer log parser. While there have been extensive efforts by researchers to utilize format structures [14] [15]. A recent research [16] was conducted to define automated algorithms to extract information from generic network and security event using a new approach named by the authors *Log Template Extraction* (LTE), which is semantics aware of network and security logs to address the problem. This approach is different approach from traditional ones that focus on network protocol inferring.

This thesis' proposed implementation adopts a more traditional way for normalizing raw events, based on regular expressions and available modules in log collection tools with normalization capabilities. The targeted format of normalization should have a set of standards, which regulates the format and transportation, such as IETF syslog protocol. That would allow sharing event correlation rulesets between community members, as the event format is standardized.

While several researchers put noticeable efforts and effectively provided algorithms for log mining and format identification [17] [18] [19] [20] [21], there are ongoing efforts to come up with a unified log format. Perhaps one of the most appreciated is *Common Event Expression* (CEE) [22] initiative, which has suspended its development since November 2014 due to stopped funding. Another more matured log format is *Graylog Extended Log Format* (GELF) [23], provided by Graylog - a log management software producer, to overcome the shortcoming of syslog protocols. Other attempts have been proposed, mostly for specific purposes to serve specific types of applications, such as, web servers [15] or security application [24] [25].

One of the common log formats is IETF syslog [26] - RFC5424, which is mainly used by UNIX based systems. IETF syslog format is the advanced version of the obsoleted BSD syslog - RFC3164. Some of the advancements in IETF syslog are secure transportation of logs using TLS over TCP/IP protocol and structured data with vendor specific extensions, which allows adding additional fields to the events. Figure 2 shows an IETF syslog entry, with fields' names and basic structure.



```
<110>1 2013-03-10T22:44:01.000000+03:00 MG75LAB AeXSvc 1288 SMP [NXLOG@14506 AexSeverity="Trace"] Operation started.
```

Figure 2. IETF syslog format (RFC5424)

On MS Windows platform, event format is different [27]; it has different set of fields and events are stored in binary files managed by dedicated log service. The main fields in a Windows event are:

- LogName – the category of the log (Application, Security, System, etc.)
- Source – the source is the logging application or service.
- Level – the severity of the event (Informational, Warning or Error) and (Success or Failure) for Security events
- Event ID – a unique event identification where each ID defines a specific category.
- Logged – the date and time when the event was logged.

2.2. Event collection

2.2.1. Event collection, storage, retrieval and graphical representation

Events generated from applications are usually written in flat files on disk, while some files might be different, like MS Windows event logs files that are stored in custom binary format [27] under *“%windir%\System32\winevt\Logs”* and queried via API to retrieve values. Other applications might store their events in a database like ISPConfig - a web hosting management application based on PHP. A centralized logging system can either store the received event on flat files on disk or to a database. Needless to say, storing events in a database facilitates searching operations, in addition to the ability of creating reports. Storing events in a database would come with the challenge of the static structures of the tables, as each table contains a fixed number of columns specified during its creation. While it remains possible to store many fields in one cell using markup language such as XML, JSON or serialization in a *document-oriented* database, that would definitely have drawbacks as it renders the search process based on specific field in a combined cell very consuming of computing resources as a result of markup data parsing process used. It is worth to mention that some software successfully implemented such method using custom indexing, which in turn creates a challenge in data storage operations. Most log management solutions create different tables for each set of log types and perform normalization of collected events based on preconfigured parsing of known event formats to the application. One of the known event collection solution based on document-oriented database using JSON is ElasticSearch [28], which resolves the overhead by performing a detection of the event data structure and creates custom index.

2.2.2. Event collection tools

This section covers the commonly used, easy to configure and free log collection tools. There are many other tools available in addition to those mentioned below, even some scripts based on Visual Basic or PowerShell are present and used for log collection and transportation.

NXLog-ce [29] is a free general purpose log collection tool targeting several operating systems and capable of collecting events from various sources. Its main advantage relies on dealing with events based on fields, which allows it to easily support various log formats and easily normalize and filter events.

Beats [30] is an open source log collection tool, which supports various sources. Beats is divided into four separated applications: *Packetbeat* collects network packets data, *Topbeat* collects resource utilization data, *Filebeat* collects events from log files and *Winlogbeat* collects Windows event logs.

Logstash [31] is another open source log collection, normalization and forwarding tool by *elastic,* with real-time pipelining capabilities and support of custom plugins. It depends on plugins for input, output and filtration.

Snare agent [32] is a log collection tool presented in two releases, a commercial one which is capable of normalization and a free version which is limited to Snare own log format.

Agentless native Event Forwarding [33] is a native method in MS Windows operating system to forward events from one or more nodes to another single node, it can be enabled manually or via group policy, and it can only transport events in Windows Event Log.

Two very flexible tools should be highlighted from the list above, NXLog-ce and Logstash, due to their support of various log formats, capability of normalization and transportation.

2.3. Event correlation and Log Management

A recent study concluded that cybersecurity threats are on continuous increase, as 2014 survey reported that the total number of security incidents detected by respondents grew globally by 48 percent from 2013 [34]. This highlights the importance of efficient log monitoring Systems *LMS* and analysis techniques significance for understanding the undergoing activities within a system. Initially, logs were used by IT specialists for technical diagnosis [35], however nowadays, security audit logs are widely used in corporations with high security awareness, and even some service providers are required by law to keep an archive of logs.

Event correlation, as defined by Jakobson and Weissman [36], is a conceptual interpretation procedure, where new meaning is assigned to a set of events that happen within a predefined time interval. This also means that synthetic events can be generated and supersede original set of correlated events. In the ITIL version three framework, event correlation takes place in the Event Management process, where the event correlator tool is called a *correlation engine*.

In the field of information technology, two or more events are correlated to each other if they have a causal or other connection. On such assumption the bonds between those events are mainly logical [37]. Accordingly, clear understanding of the causality surrounding the production of a raw event is an essential preparation for building proactive event correlation rules. The approach of quantifying temporal and spatial failure correlation has been discussed [38] in an attempt to correlate failure in systems. For example, if failed login attempts from a remote machine is accompanied by LDAP system unreachable. Such two events shall be correlated logically to prevent the creation of false positive brute force attacks alerts, in particular, if the login attempts source is a node known to query the destination for service, which requires credentials validation.

2.3.1. Properties of Event Correlation Engines

Depending on the approaches adopted by a correlation engine, it can be harnessed for specific usage or can be rendered for general purpose. The properties of a correlation engine play a role in its capabilities and flexibility to perform more advanced tasks. However not all properties apply to all techniques and many techniques can be used with different properties [39]. The main properties of correlation engines are:

- **Domain Awareness:** whether a correlation engine is built for a specific domain (application usage), and knows what kind of information it processes.
- **Self-Learning vs. External Knowledge:** whether the correlated events are predefined set by operator, or the engine might be able to assume correlation based on preset knowledge base.
- **Real-time vs. Stored Data:** is the correlation engine capable of correlating time based archived events or only based on current system clock.
- **Stateless vs. Stateful:** a correlation engine capable of maintaining a memory of event history is considered statful.
- **Passive vs. Active:** A passive correlation engine can only correlate events based on previous events and internal current state, however if the engine is capable of gathering more information from external sources, it is considered active.

- **Centralized vs. Distributed:** whether the correlation is performed on a centralized system or on endpoint devices. A *hybrid* approach also can be adopted by centralizing specific events processed by distributed engines.
- **Default Policy:** Whether the correlation engine can perform an action on non-matched events.
- **Loss of Information:** an event correlation engine is lossless, if all correlation operations are lossless.
- **Transparency:** Whether the decisions taken by the correlation engine are transparent to human operator, or there is no way for auditing such decisions.
- **Robustness:** the capability of handling new and unknown situations; however this is mainly related to other adopted approaches and configured rules.
- **Maintainability:** mainly defined as the stability and meeting expected behavior by the correlation engine within different environments.
- **Deep vs. Surface Knowledge:** correlation engines can further be discerned by whether they rely on knowledge gained from observation and experience only (surface knowledge), or on knowledge based on understanding the structure and functioning of a system (deep knowledge).

2.3.2. Event Correlation Techniques

- **Dependency Graph based** [40]**:** a graph based event correlation method, where the entire IT system is represented as a graph. Network devices, servers, applications and other system components are represented as nodes; while dependencies between system components are the arcs of the graph. When a component of the system fails, the graph is used to define other affected system components.
- **Codebook based** [41]**:** a human expert creates a so-called *codebook* which consists of vectors. Each vector describes a common description (or root cause) for specific error conditions in the IT system and the components of the vector correspond to symptoms of this fault condition. When faults are observed within an IT system, vector is calculated based on these faults, and codebook is searched for finding a match or a partial match. The match is then reported to the administrators.

- **Bayesian Network based** [42]**:** a directed acyclic graph, which models the probabilistic relations between system components represented by random variables.
- **Neural Network based** [43]**:** an artificial model of intelligence created by a network of processing nodes, which performs operations on the evaluated inputs to generate outputs, which are used as inputs for other nodes.
- **Rule based** [36] [44]**:** the rules specify a condition to action relation, when one or more events match a condition, an action or more are triggered.
- **Model based** [45]**:** the representation of the structure and behaviors of a system under observation in a model.

In addition to the list above, other event correlation approaches were discussed in [46].

2.3.3. Hybrid approach of Event Correlation architecture

A comparison between centralized and distributed event correlation approaches [47] spots the light on the advantages and disadvantages for both architectures. While some researchers might favor one over the other, there are no doubts that the desired outcomes of an implementation are the final judge.

Distributed approach has been adopted using SEC by J. Myers et al. [48]. The authors stated several disadvantages of such architecture, though they provided a good use case example. The distributed approach is based on correlating events on the node producing the events itself. It is needless to say that without a centralized log management system, it will require extensive human operator resources to perform audit tasks. Furthermore, a wider correlation between several nodes is not possible.

On the other hand, centralized event correlation provides a much wider picture of incidents occurring within an environment. The centralized architecture is based on the transportation of events through the network from nodes producing the logs. That clearly generates intensive network traffic unless filtering of events takes place whether on each node or on a log servers acting as gateways. In addition to the mentioned impacts, high resources utilization is expected on the main event correlation server, which might become a scalability obstruction.

A hybrid approach combines both centralized and distributed event correlation approaches mainly on two stages. Firstly by following the distributed approach of correlating events on each node, where local actions are triggered accordingly. Secondly by transporting critical and synthetic events generated by the distributed correlation engine to a centralized log server where a correlation engine resides to trigger wider actions, which take place independently from the node. For example, if several authentication failure events within a small window of time are generated by an IP, an action is triggered to block that IP for few minutes. Then a synthetic event is generated and transported to a centralized correlation engine. The centralized correlation engine evaluates the amount of incidents received, their sources and destinations and correlates with other forwarded events to take a wider action. Assuming that the source IP address is external and attempting login on several servers in DMZ, an action can be triggered on the firewall to blacklist the attacker IP.

2.3.4. Commonly used Log Monitoring and Event Correlation solution

Comparative analysis between existing event correlation tools, has received some attention from researchers over the past years [49] [39]. However the assumption of using UNIX based systems had been always adopted. To serve this research’s goal of providing a free proactive event correlation solution, capable of running on MS Windows operating system, a descriptive features comparison has been carried out between commonly used log monitoring and event correlation solutions applicable on MS Windows OS. The features collection is based on each product’s documentations and capabilities claimed by their authors.

Splunk [50] is a log management solution, which supports several operating systems with a dedicated installer for MS Windows. While it is a commercial solution it still provides a free perpetual license with limited amount of logs to process daily. Commercial licensing depends on the expected amount of processed logs per day. Splunk provides graphical visualization and user interface for configuration. It uses textual search for real-time alerts and triggers automatic responses, which is considered as passive correlation.

ElasticSearch ELK Stack [28] is a collection set of utilities combined together to create end-to-end search and analytics platform. The platform is regarded as a stable solution by many institutions; it is used by Microsoft, Reuters, Netflix, Adobe Systems, CISCO, eBay and others. The collection consists of the following main tools:

- Logstash: flexible, open source event collection and transportation tool based on Java.
- ElasticSearch: distributed, open source search and analytics engine with high scalability features.
- Kibana: open source data visualization platform with interactive graphical interface and custom dashboards.
- Beats: set of log collection and transportation tools and framework.
- Other additional open source utilities: Beats, Watcher, Shield, Elastic Cloud, Marvel, Elasticsearch for Apache Hadoop,

Pros:

- Free and open source
- Monitoring and alerting capabilities
- A well-designed collection of free tools, which are combined together to form a powerful platform for event management
- Precise and easy to follow documentations and tutorials

Cons:

- Log filtering, which does not promote to the level of correlation engine
- Available correlation engine modules are commercial and costly
- Requires extensive resources
- Requires skilled specialists for fine tuning

LOGalyze [51] is a freeware centralized log management and network monitor with real-time data analysis capabilities based on Java.

Pros:

- Agent and Agentless log collection
- Normalization capabilities
- Prepacked set of reports
- Free

Cons:

- Requires extensive resources
- Limited correlation engine features
- Java framework is known of lacking robust security.

Esper / NEsper [52] are open-source Java and .NET based frameworks for Complex Event Processing (CEP).Esper and NEsper frameworks enable rapid development of applications that process log events. They require development of a solution and cannot act as event correlators out of the box. They also require additional third party libraries.

Simple Event Correlation [3] – SEC is a very powerful and lightweight real-time correlation engine for network management, log file monitoring, security management, fraud detection, and other tasks which involve event correlation. It is written in Perl, which means it requires extra installation of Perl distribution on Windows. It can store events to a database with the aid of extra Perl functions.

NXLog-ce (pm_evcorr) module [29] is a dedicated module within NXLog-ce log collection tool, which acts as a correlation engine. It is coded using objective C programming language. It is mentioned in its release notes that it was inspired by Simple Event Correlator, and claims superiority over it.

The two latter solutions' features are covered in a dedicated section in form of listed comparison. Each of the solutions, SEC and NXLog-ce *pm_evcorr* module, represents a lightweight event correlation solution, which can serve as a correlation engine for the implementation proposed by this study.

It is obvious from comparing the information above that most event management solutions are database dependent, focus on generating reports and visualized graphs, are heavyweight with main focus on centralized approach. This study aim is providing a lightweight free solution, which focuses on proactive event correlation by adopting active correlation approach to validate correlated events and minimize false positives, with scalability in mind. The implementation proposed within this thesis can leverage free and open source log management solutions like Elastic and LOGalize to build a full SIEM solution with powerful proactive correlation capabilities.

2.3.5. NXLog-ce *pm_evcorr* module vs. SEC

NXlog-ce provided in March 2013, release 2.3.1027 *pm_evcorr* [5] - a special processor module for event correlation inspired by SEC. While the reference manual claims several advantages over SEC, further analysis revealed weakness points overcoming its advantages in particular to the current study use case. Below comparative analyses took place between NXLog community edition version 2.9.1347 against SEC version 2.7.8.

- Regular expression vs. operating on fields' values

Technically, querying Windows event logs programmatically using Windows API, returns results as field-value pairs. NXlog-ce input module *im_msvistalog* used for collecting Windows event logs, leverages the way results are retrieved to become fields aware without parsing. This feature apparently gives NXLog-ce performance superiority. However, taking into consideration collecting logs from various applications, which might be in irregular formats, those logs will be parsed using regular expressions. Furthermore, other applications can write to MS Windows event log [53] with irregular message formatting, which keeps the need for parsing the log message using regular expression. This feature in NXLog-ce can be used for filtering events, but does not supersede using regular expression for further information extraction.

- Offline time based event correlation

NXlog-ce *pm_evcorr* module uses time field for time based event correlation, which allows stored data offline processing, unlike SEC capability of correlating event based on real time only.

- Programming language

NXlog-ce *pm_evcorr* module is coded using objective C programming language, which theoretically should be considerably faster than a scripting language requiring an engine to parse and compile it to CPU instructions. SEC is coded as a Perl script, which is an interpreted language. Perl 6 can compile its scripts for better performance, but is not fully compatible with earlier releases scripts, and its compatibility mode is very limited.

- Actions triggering

NXlog-ce *pm_evcorr* module lacks first action trigger in a pair correlation rule, thus a second rule is required to cover.

- Rules types

NXLog-ce correlation engine module rules types are roughly equivalent to *Single*, *Supress*, *Pair*, *PairWithWindow* and *SingleWithThreshold* SEC rules types. SEC has rich collection of additional rules types, such as, *EventGroup* and *Jump* rules. Additionally, SEC rules can be combined together for more advanced correlation operations.

- Synthetic events

SEC is capable of generating synthetic events, which are used to trigger other rules. NXLog-ce correlation engine module does not provide a direct alternative.

- Maintenance

NXlog-ce *pm_evcorr* module received only one update on July 2014 according to *changlog.txt* packed within version 2.9.1347 under *doc* folder. NXLog-ce mainly concentrates on its core functionality as a log collection tool. While SEC as a dedicated correlation engine still gets new features and more flexibility within irregular updates cycle approximated to twice a year.

Considering the above comparison points, NXLog-ce *pm_evcorr* module can be used as an alternative to SEC in some use cases. However, it is by no mean a substitution or competitor in particular to the use case of the solution proposed by this thesis.

2.3.6. Analysis of the previous attempt to run SEC as MS Windows service

A previous attempt to run SEC as a MS Windows service has taken place in 2008 by one of SEC community members and has been published via SEC mailing list [6]. However that attempt was not much of success for the following reasons:

- Upgrading SEC requires manual edit of its script file because the script was modified to use a custom Perl module.
- It depends on ActiveState Perl, which limits the interaction with the process resulted by the lack of UNIX systems features, such as, interrupt signals, spawn action, etc.

- It depends on ActiveState Perl custom module *Win32::Daemon* [54] contributed by Perl community member Dave Roth. The latest update of the module was in June 2003, whereas ActiveState Perl latest update was on March 2013; at the time of writing this study.
- A known issue of crash upon using shell command action *shellcmd* has not been resolved.

Those drawbacks render the attempt unreliable, emphasizing the need of a reliable solution to run SEC Perl process as a MS Windows service.

3. Log monitoring and proactive Event Correlation on Microsoft Windows using Simple Event Correlator Windows Integration

This chapter demonstrates a proposed implementation for using SEC correlation engine on Microsoft Windows. The main aspects of the developed application, which integrates SEC Perl process to run as a service, are covered. Using NXLog-ce as a log collection tool, including securing the transportation of events securely, is covered precisely. SEC rulesets are provided as a proof of concept. A final implementation layout is demonstrated at the end of this chapter.

3.1. Requirements of the correlation engine and event collection tool

This section highlights the requirements of the correlation engine and log collection tool needed for the implementation proposed in this thesis. Considering the use case and the ability to provide a multipurpose event correlation solution, the required correlation engine features are:

- Real-time processing of incoming events
- Flexible configuration
- Ability to generate synthetic events
- Efficiency
- Scalability
- Maintainability, more transparent correlation decisions and frequently updated
- Lightweight with lowest possible resource consumption
- Active correlation, ability to take additional queried information into account
- Distributable with capability of centralization
- Generalized distribution package, without separating server and client packages

Although *Esper* [52] provides a flexible framework, it would still require extensive efforts for coding a mature general purpose correlation engine. The above requirements were found to be met by SEC as it is more favored than NXlog-ce pm-evcorr module as per section 2.3.5 conclusion. Splunk correlation is passive and its free version limits the amount of logs processed daily. ElasticSearch correlation engine modules are commercial and costly, and the solution is heavyweight.

SEC has been chosen as a correlation engine for its wide acceptance in industrial and academic communities [3] [55], as a lightweight and flexible log monitoring and event correlation engine. SEC is coded using Perl, while that allows it to be cross platform, yet to run on MS Windows OS, the implementation process requires a lot of work including the installation of a Perl distribution and manual interactions to run the required command parameters. Like any application for Windows OS, it is not possible to run on boot without user-interaction or an enabled Windows service, thus a solution is required to provide a flexible and reliable integration.

For the reasons above, Simple Event Correlator Windows integrator “SECwin” [56] was developed by the author to integrate SEC as a Windows OS service. By using Cygwin Perl as default Perl distribution to leverage Cygwin UNIX emulation, yet the package is flexible to support different Perl distributions as required. SECwin includes a user interface to facilitate SEC and Perl commands’ parameters configuration and a live viewer for SEC logs and statistics dump files.

Log collection tool requirements are:

- Ability to collect windows event logs
- Log normalization capabilities
- Log transportation in a secure manner, compliance with RFC5425 is favored
- Support of multiple common log formats in particular syslog format
- Lightweight
- Well documented and easy configuration

Beats consists of many executables, each for a specific purpose, which would consume more computing resources and renders deployment harder. Snare free version supports only its own format, and no normalization. Logstash is based on Java, which is known for security issues. Java increases the consumed resources, and adds complexity to the solution to insure regular updates. On the other hand, Windows OS native event forwarding is not simple to configure, does not support other log sources, and would still require a tool to feed its contents to the SEC, which was promoted earlier as correlation engine.

NXLog community edition was selected as free log collection tool for its support for several formats, logs normalization capabilities, compatibility with RFC5424 and RFC5425, which are required for the proposed implementation. Additionally it is lightweight and easy to configure.

The author built a syntax-highlighter configuration file for Notepad++ to facilitate creating and editing NXLog-ce configuration file. The file is included in SECwin package, and is located under utilities folder under installed SECwin directory.

3.2. Microsoft Windows Perl distributions features and performance

The main Perl distributions for MS Windows according to Perl.org download page are ActiveState Perl [57], Strawberry Perl [58] and DWIM - an open source Strawberry Perl derivative. In addition to those mentioned, Cygwin [59] - a collection of GNU and Open Source tools which provide functionality similar to a Linux distribution on Windows, provides Cygwin Perl distribution. Cygwin Perl runs on Windows in a virtually emulated UNIX environment with aid of special Cygwin libraries. Other Windows Perl distributions are available, mostly derived from the main distributions mentioned above. This section provides features list for each main distribution, in particular the features impact SEC operations and the proposed implementation.

ActiveState Perl 5.22.1.2201 features:

- Easy to install as a single MSI package
- Includes package management utility *PPM,* which allows the installation of additional modules and adding extra repositories
- Availability of commercial support
- Supports user compilation modules
- Package size is 28.5 MB, on disk 117 MB

Strawberry Perl 5.22.1.2 features:

- Easy installation via single MSI package
- Includes gcc compiler plus related tools, all external libraries for compiling extra modules
- uses *cpan* for extra modules installation

- Package size is 83 MB, on disk 112 MB plus tool-chain 311 MB

Cygwin Perl 5.22.1 features:

- Easy installation via single user interface installer
- Includes *cpan*, yet modules can be compiled via Cygwin *gcc*
- Portability: although by default Cygwin is provided as an executable installer, which in turn installs the selected packages. The binaries were found to be fully portable and able to run without installation as long as they are carried with their required dependencies.
- Tarball sum size is 39 MB, on disk 129 MB
- Cygwin provides advanced UNIX emulation, which is required by a number of SEC features, such as:
    - *udgram*, *ustream*, *closeudgr* and *closestr*: used for SEC input
    - *Spawn*, *cspawn* actions: run command and uses its output as SEC input
    - Stdin: SEC input from standard input
    - Named pipe: SEC input from named pipe
    - Process fork: SEC detach and run in background
    - Signals: interrupt signals used for various SEC commands
    - Other Cygwin binaries can be used for various operations.

The conclusion from comparing the above features list is that ActiveState Perl and Strawberry Perl distributions can be considered alike from the perspective of features required to run SEC. On the other hand Cygwin Perl provides UNIX emulation features, which SEC requires for some actions and for receiving signals. The actions triggered by signals can be alternatively triggered by special SEC rules, which use *lcall* action to call SEC internal functions. While using rules to emulate signals internally can be considered an alternative, it is still not a very flexible solution. A critical issue which occurs often with ActiveState and strawberry Perl, is that NXLog-ce fails to rotate the log file because the handle created by Perl is not released efficiently. Thus the author recommends using Cygwin for its superiority in provided features.

A performance comparison took place between the mentioned distributions as a stress test using SEC rulesets. Three virtual machines were used, running simultaneously. Each VM had a different Perl distribution installed, in addition to SECwin and NXLog-ce. All VMs were assigned same virtual resources, were using same SEC rulesets and input files source, which were transported via NXLog-ce.

Hardware and Virtual machines specifications:

Bare metal: Dell PowerEdge R515, 64GB RAM, AMD Opteron™ 4280 - 16 x 2.8GHz CPU, 6 SSD RAID1 cluster, running ESXi 5.1U3.

Virtual machines: 4 virtual CPUs, 8GB RAM, running MS Windows server 2008 R2 enterprise edition.

All VMs started and stopped correlation operations on the same time, by sending defined log entries, matched by NXLog-ce correlation engine, which triggered start and stop of SECwin service. Performance information was collected from SEC statistic dump, while memory consumption, modules and handles counts were collected by SECwin. All tests were repeated many times. The rulesets were intentionally written to overload SEC, all rules were of type *single*, processing all input events, using pattern type *RegExp*. The performance test result, using actual SEC rulesets, within short period of time are: total processed input lines 319, total matched events 317, by 29 rules, in total run time of 428 seconds.

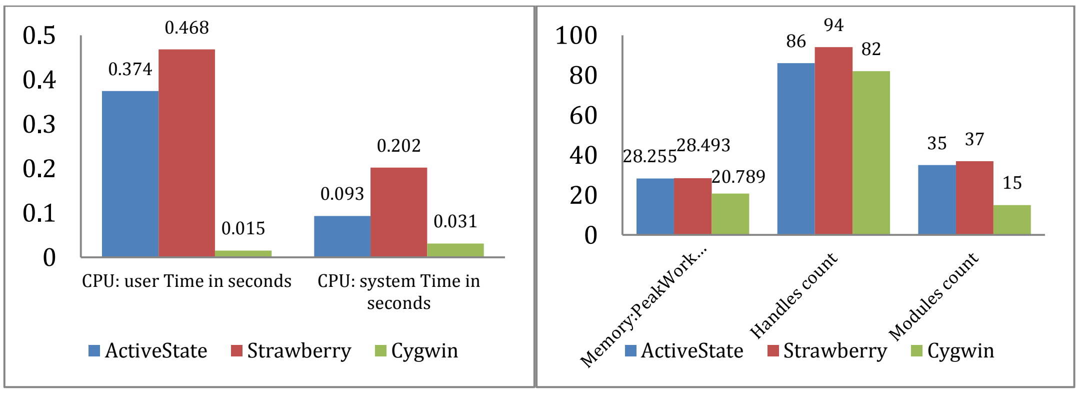


Figure 3.Perl performance using actual rulesets

In Figure 3, charts represent performance comparison. The lower numbers represent less consumed resources. The charts show Cygwin Perl resources consumption is the lowest, especially for the used CPU time.

On other attempts of running the test using same conditions described above. Whenever the test ran for longer period of time, depending on the load caused by the amount of events received from the source, the results were not identical. Numbers of processed lines by SEC were not equal, and the run time differed by few seconds because SEC took longer time to terminate due to the heavy load. For that reason, additional tests were required using different method, to define the cause.

Another stress test took place using two SEC rules. One rule creates an event matched again by itself in a loop of one million times, the other rule triggers the first one on SEC startup. The Looping rule is configured to collect SEC statistics dump at the end of the cycle.

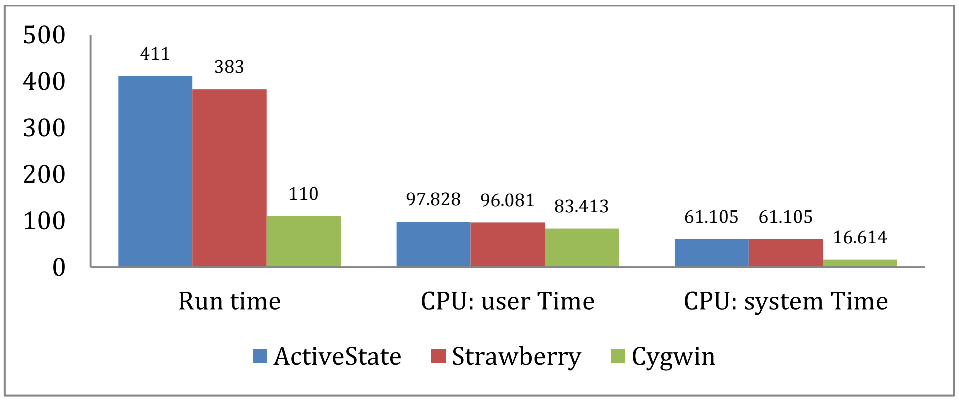


Figure 4. Perl performance using looping rule, With Disk I/O

In Figure 4, chart of consumed CPU resources and total run time of the loop operation. SEC log level was set to level 6- debug messages. Each loop produced three entries in SEC log file, which is an intensive disk operation.

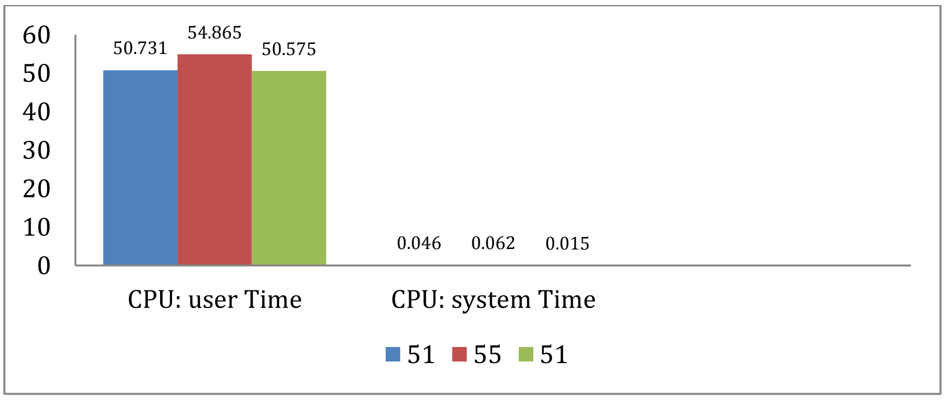


Figure 5. Perl performance using looping rule, no Disk I/O

In Figure 5, chart of consumed CPU resources and total run time of the same loop operation with SEC log level set to 1- critical messages. The loop rule did not produce any log entries in SEC log file. The chart shows that consumed CPU resources are very close, while strawberry Perl was a little behind by few seconds. CPU system time was minimal, because there were no resource consuming operations performed by the OS kernel on behalf of Perl.

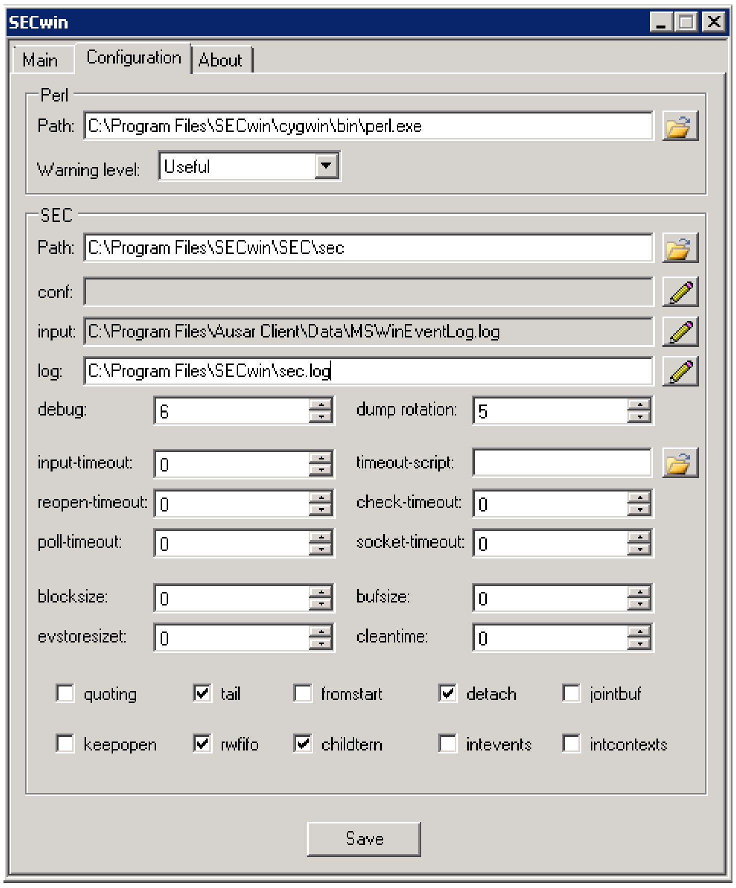
The difference between CPU user time and total SEC run time was minimal when disk operations were eliminated, while it was noticeably longer with intensive disk operations. The comparison between eliminated and intensive disk operations using the same looping SEC rule, justifies the unequal total SEC run time and processed lines count between the three Perl distributions using actual rulesets. The conclusions of all performance testing results highlight Cygwin Perl performance superiority with intensive disk operations. Disk operations cannot be avoided because log entries are usually read from files.

3.3. Simple Event Correlator Windows Integration - SECwin

As demonstrated in section 2.3.6, the previous attempt to create a MS Windows service to wrap SEC Perl process was not much of success. Hence there were no other attempts made, and such integration is required for the proposed implementation by this study. Additionally SEC community members were querying the ability of such integration, which indicates the existence of SEC usage on Microsoft Windows platform. These reasons were enough motivation for developing *SECwin* [56] as a solution, using Microsoft Windows .NET framework as it is the native programming framework for Microsoft. SECwin PE itself has a very small disk foot print of approximately 300 KB and consumes very small amount of memory approximately 6MB of private working set for the service process. SECwin resources utilization is not affected by SEC load. To facilitate the deployment process, SECwin full installer package was loaded with SEC and Cygwin Perl within the same MSI. SECwin can also get deployed as a single portable executable, using its CLI to it install itself, assuming that Perl is already installed. While the main functionality of SECwin is to act as a service wrapper for SEC Perl process, additional features were added, including:

- SEC Perl process watchdog for unexpected termination
- User interface to build SEC command’s parameters
- Sending signals to SEC via UI or System tray icon

- SEC statistics dump file rotation
- Display SEC Perl process statistics and information
- SEC log and statistics dump files viewers
- Automated update for SEC and SECwin
- Converting paths to Cygwin style
- System tray icon context menu for faster interactions



Perl
Path: C:\Program Files\SECwin\cygwin\bin\perl.exe
Warning level: Useful

SEC
Path: C:\Program Files\SECwin\SEC\sec
conf:
input: C:\Program Files\Ausar Client\Data\MSWinEventLog.log
log: C:\Program Files\SECwin\sec.log
debug: 6
dump rotation: 5
input-timeout: 0
timeout-script:
reopen-timeout: 0
check-timeout: 0
poll-timeout: 0
socket-timeout: 0
blocksize: 0
bufsize: 0
evstoresizet: 0
cleantime: 0

☐ quoting ☑ tail ☐ fromstart ☑ detach ☐ jointbuf
☐ keepopen ☑ rwfifo ☑ childterm ☐ intevents ☐ intcontexts

Figure 6. SECwin UI: SEC configuration

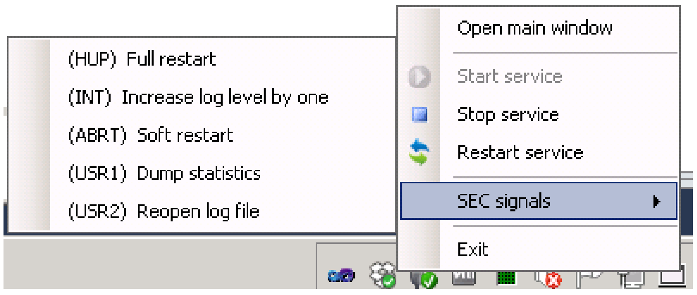
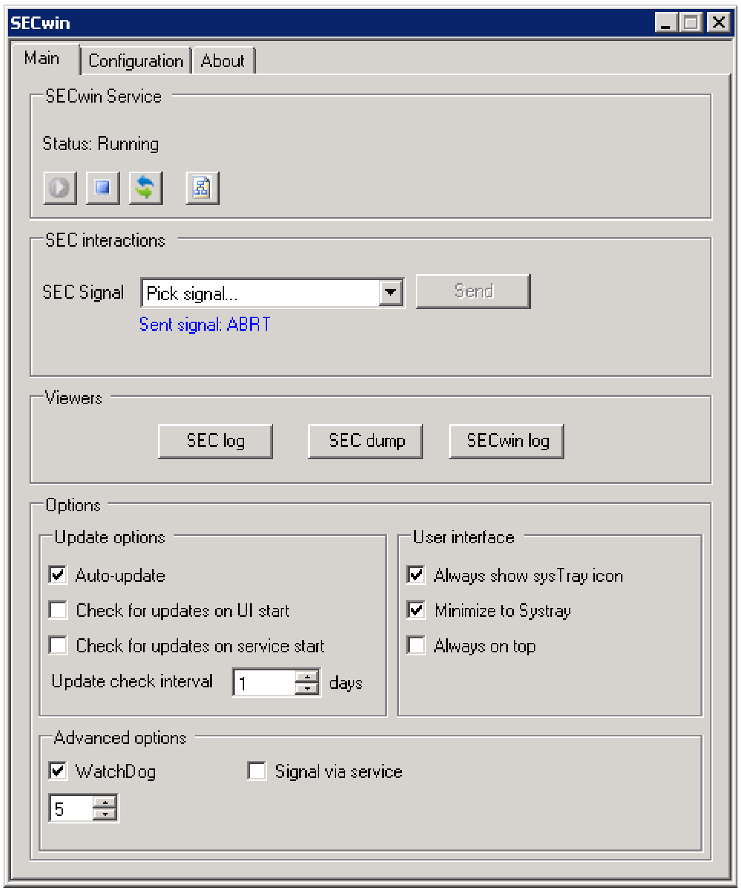


Figure 7. SECwin UI: Systray icon context menu

A screen shot in Figure 7 demonstrates system tray icon for easier and faster interactions with SEC and SECwin.


SECwin

Main | Configuration | About

SECwin Service

Status: Running

SEC interactions

SEC Signal: Pick signal... Send

Sent signal: ABRT

Viewers

SEC log | SEC dump | SECwin log

Options

Update options

☑ Auto-update

☐ Check for updates on UI start

☐ Check for updates on service start

Update check interval: 1 days

User interface

☑ Always show sysTray icon

☑ Minimize to Systray

☐ Always on top

Advanced options

☑ WatchDog: 5

☐ Signal via service

Figure 8. SECwin UI: MainTab – Service control & log viewer

In Figure 6, there is a screen shot of SECwin user interface configuration tab of SEC command parameters. Figure 8 displays the main tab in which the upper box group monitors and provides control over SECwin service status and an extra button for displaying SEC Perl process information. Second group box provides signal interactions with SEC process followed by group box to launch life viewer for SEC logs and SEC statistics dump. The lower two group boxes provide configurations for user interface behavior, update settings and more advanced options for SECwin application.

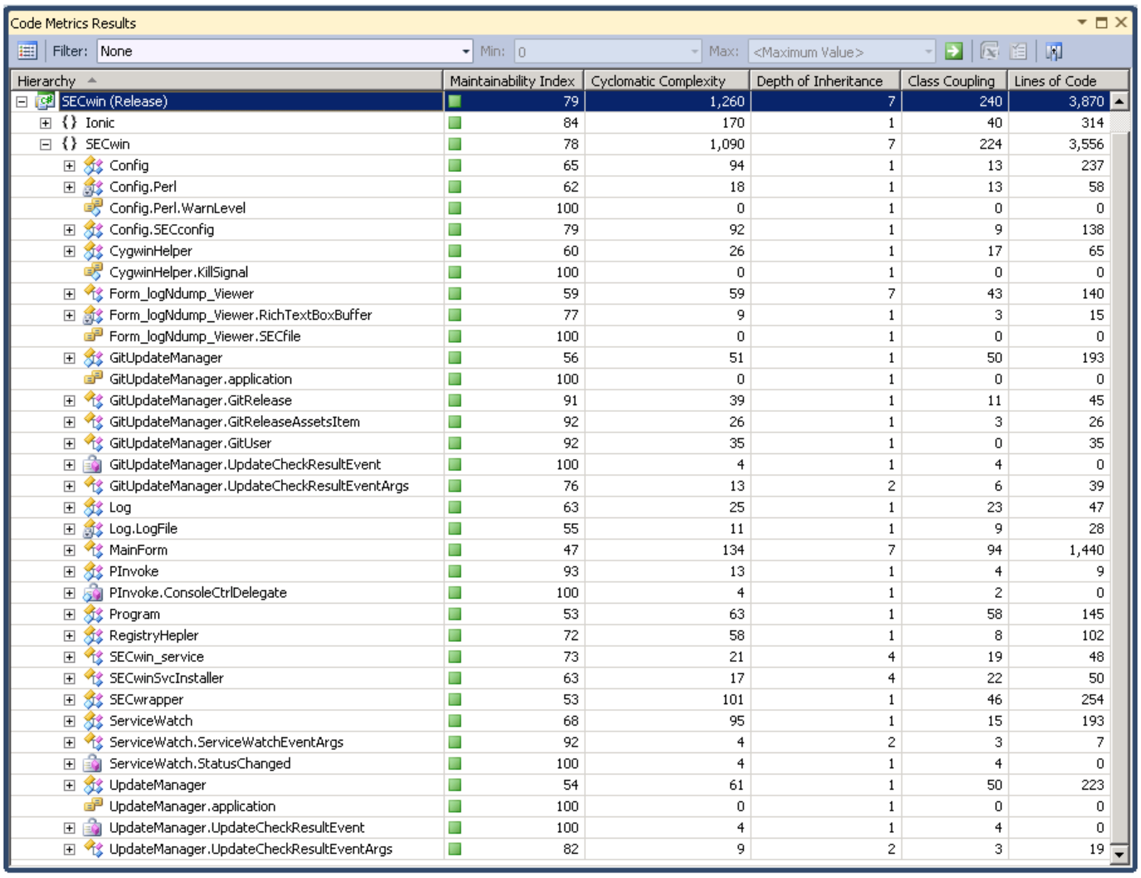
3.3.1. Design and development

Technically SECwin is a MS Windows service which starts a Perl process running SEC script as an external application. While the main functionality sounds simple, several technical challenges were met during the development process. Whether these challenges were related to SECwin ability to handle different Perl distribution or were driven by limitations of the operating system or .NET platform, they had to be addressed to produce a successful reliable solution.

SECwin solution consists of two projects, SECwin portable executable and SECwin installer, which builds MSI package for installing SECwin and deploying SEC and Cygwin. SECwin is compiled to a single portable executable to avoid having three different executable files, one for user interface, second one for Windows service and a third one as CLI. Also the coding avoided any dependency on third party dynamic libraries. This design was adopted to achieve portability of a single PE file with no dependencies, precisely for cases when a Perl distribution is already installed on the system.

SECwin source code is approximately 3900 lines divided into several files, each contains one or more classes as per the standards of object oriented programming. More detailed coding metrics are displayed in Figure 9. The developed code generated sets of internal API calls, which are demonstrated within the figures of code snippets in the following sections. The internal API facilitates future development of the application, and allows developing additional plugins.

SECwin configurations are stored in Windows registry, thus deploying or changing configuration on several nodes can be achieved by group policy or Windows compatible registry file.


Code Metrics Results

Filter: None Min: 0 Max: <Maximum Value>

Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Code
SECwin (Release)	🟩 79	1,260	7	240	3,870
Ionic	🟩 84	170	1	40	314
SECwin	🟩 78	1,090	7	224	3,556
Config	🟩 65	94	1	13	237
Config.Perl	🟩 62	18	1	13	58
Config.Perl.WarnLevel	🟩 100	0	1	0	0
Config.SECconfig	🟩 79	92	1	9	138
CygwinHelper	🟩 60	26	1	17	65
CygwinHelper.KillSignal	🟩 100	0	1	0	0
Form_logNdump_Viewer	🟩 59	59	7	43	140
Form_logNdump_Viewer.RichTextBoxBuffer	🟩 77	9	1	3	15
Form_logNdump_Viewer.SECfile	🟩 100	0	1	0	0
GitUpdateManager	🟩 56	51	1	50	193
GitUpdateManager.application	🟩 100	0	1	0	0
GitUpdateManager.GitRelease	🟩 91	39	1	11	45
GitUpdateManager.GitReleaseAssetsItem	🟩 92	26	1	3	26
GitUpdateManager.GitUser	🟩 92	35	1	0	35
GitUpdateManager.UpdateCheckResultEvent	🟩 100	4	1	4	0
GitUpdateManager.UpdateCheckResultEventArgs	🟩 76	13	2	6	39
Log	🟩 63	25	1	23	47
Log.LogFile	🟩 55	11	1	9	28
MainForm	🟩 47	134	7	94	1,440
PInvoke	🟩 93	13	1	4	9
PInvoke.ConsoleCtrlDelegate	🟩 100	4	1	2	0
Program	🟩 53	63	1	58	145
RegistryHepler	🟩 72	58	1	8	102
SECwin_service	🟩 73	21	4	19	48
SECwinSvcInstaller	🟩 63	17	4	22	50
SECwrapper	🟩 53	101	1	46	254
ServiceWatch	🟩 68	95	1	15	193
ServiceWatch.ServiceWatchEventArgs	🟩 92	4	2	3	7
ServiceWatch.StatusChanged	🟩 100	4	1	4	0
UpdateManager	🟩 54	61	1	50	223
UpdateManager.application	🟩 100	0	1	0	0
UpdateManager.UpdateCheckResultEvent	🟩 100	4	1	4	0
UpdateManager.UpdateCheckResultEventArgs	🟩 82	9	2	3	19

Figure 9. SECwin code metrics

3.3.2. Command Line Interface

SECwin command line interface accepts the following parameters:

- *Install* – SECwin will copy itself to provided destination, and then registers the copied PE as a Windows service. If there is no destination provided, the default location is *\Program Files\SECwin\*. Upon the first launch of SECwin user interface, it will offer to download SEC.
- *Uninstall* – unregisters the PE as a windows service.
- *Update* – performs update to the specified application, options are: SECwin, SEC or all. Accepts an optional additional parameter *checkonly*, which provides information about the availability of an update without any further actions.
- *Console* – displays a console window to print out debugging information
- *Debug* – attaches the process to a debugger by performing a debug break. Accepts optional additional parameter *wait* to pause launching the application till a debugger is attached instead of a debugger break action. Debugger break causes the application to through a break exception.

- *SvcWatchDog* – monitors SECwin service and restarts it if stopped, unless system is shutting down, or feature is disabled.

3.3.3. Handling different Perl distributions

Although SECwin default Perl distribution is Cygwin, it still fully supports other Perl distributions, which are categorized by the code as UNIX emulated, like Cygwin and non-emulated like ActiveState Perl or Strawberry Perl. It is very important to define the distribution category to be able to handle different ways of handling Perl process. It makes difference in the code flow if the running Perl process is UNIX emulated or not as covered in the following section. An example of such code flow decision is to decide the way of terminating SEC gracefully or if SEC parameters with paths, such as *input* and *conf*, shall be converted to UNIX style.

Determining Perl distribution is performed by executing a new process of Perl with parameter (-V) to retrieve version details. The output of the command is parsed and then accordingly a public Boolean value is set in the configuration class to determine if Cygwin Perl is used.

3.3.4. Terminating SEC Perl process gracefully

Terminating SEC Perl process gracefully has to be handled via an interrupt signal, as any attempt to exit Perl process programmatically causes an ungraceful termination of SEC. The method call *Process.CloseMainWindow()* fails because Perl is a console application. Similarly using API call *WINAPI::SendMessage(WM_CLOSE)* causes an unexpected termination of SEC. Microsoft Windows operating system is limited to two types of interrupt signals only for console applications [60], *CTRL+C* and *CTRL+BREAK*. The signal *CTRL+C* is equivalent to *SIGINT*, thus in case Cygwin Perl is used and SEC process is forked using *detach* parameter, the signal will be treated for log level change. Accordingly, to perform a graceful termination for SEC on Cygwin Perl, an external process for Cygwin Kill.exe with appropriate parameters to send *SIGTERM* to SEC must be executed. This process is covered with more details in section 3.3.6. In both of the cases, using Cygwin kill.exe or sending *CTRL+C* signal, watchdog functionality, which is covered in section 3.3.5, has to be internally suspended to avoid restart loop.

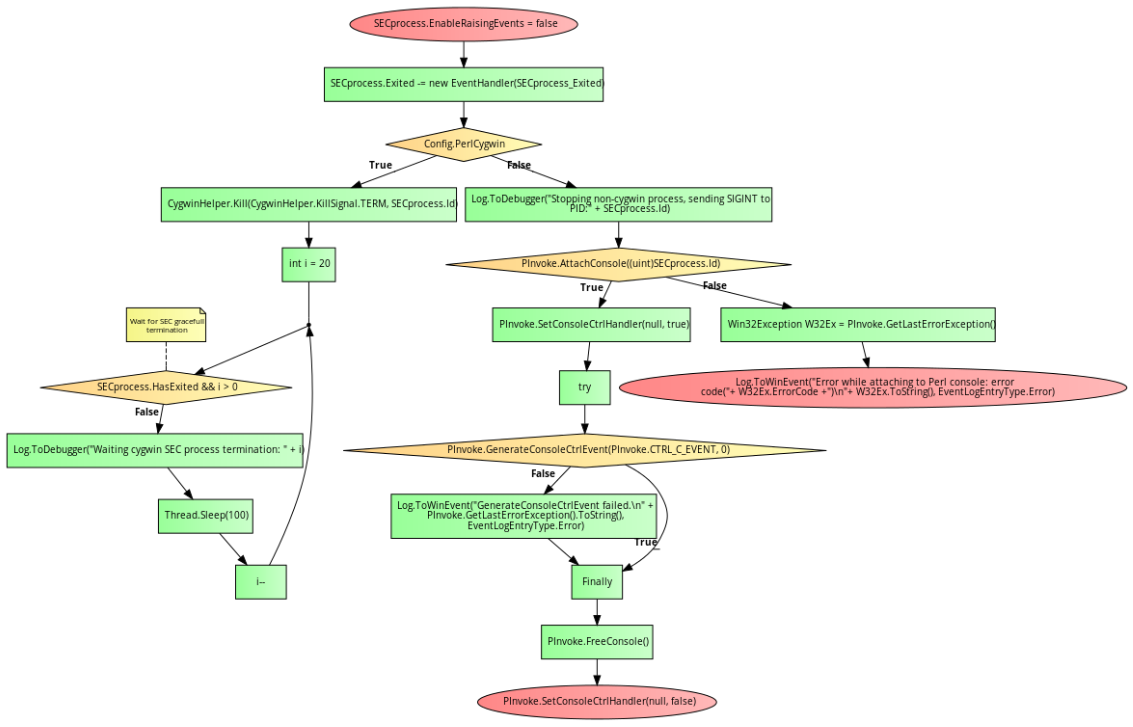

SECprocess.EnableRaisingEvents = false
SECprocess.Exited -= new EventHandler(SECprocess_Exited)
Config.PerlCygwin
True
False
CygwinHelper.Kill(CygwinHelper.KillSignal.TERM, SECprocess.Id)
Log.ToDebugger("Stopping non-cygwin process, sending SIGINT to PID: " + SECprocess.Id)
int i = 20
PInvoke.AttachConsole((uint)SECprocess.Id)
True
False
Wait for SEC gracefull termination
PInvoke.SetConsoleCtrlHandler(null, true)
Win32Exception W32Ex = PInvoke.GetLastErrorException()
SECprocess.HasExited && i > 0
False
try
Log.ToWinEvent("Error while attaching to Perl console: error code("+ W32Ex.ErrorCode +")\n"+ W32Ex.ToString(), EventLogEntryType.Error)
Log.ToDebugger("Waiting cygwin SEC process termination: " + i)
PInvoke.GenerateConsoleCtrlEvent(PInvoke.CTRL_C_EVENT, 0)
False
Thread.Sleep(100)
Log.ToWinEvent("GenerateConsoleCtrlEvent failed.\n" + PInvoke.GetLastErrorException().ToString(), EventLogEntryType.Error)
True
i--
Finally
PInvoke.FreeConsole()
PInvoke.SetConsoleCtrlHandler(null, false)

Figure 10. SECwin flowchart: Terminating SEC gracefully

The flowchart in Figure 10 displays the code execution logic of terminating SEC process gracefully.

3.3.5. Watchdog: Handling SEC Perl process unexpected termination

An unexpected termination of an application occurs with no deny, whether by a process crash or other causes such as files corruption, misconfigured SEC ruleset, ruleset subroutine, or defective Perl modules. Considering few reports on SEC mailing list related to crash occurrences of SEC Perl process running on Windows platform, precaution measures must be taken to handle such situations. Due to that, watchdog feature was developed, which handles unexpected termination of SEC Perl process by restarting it with threshold of maximum attempts as a safeguard. Tentatively the idea is simple, however, this particular feature had several challenges, such as *detach* SEC parameter, which forks SEC into a new process, causing false-positive detection for unexpected termination.

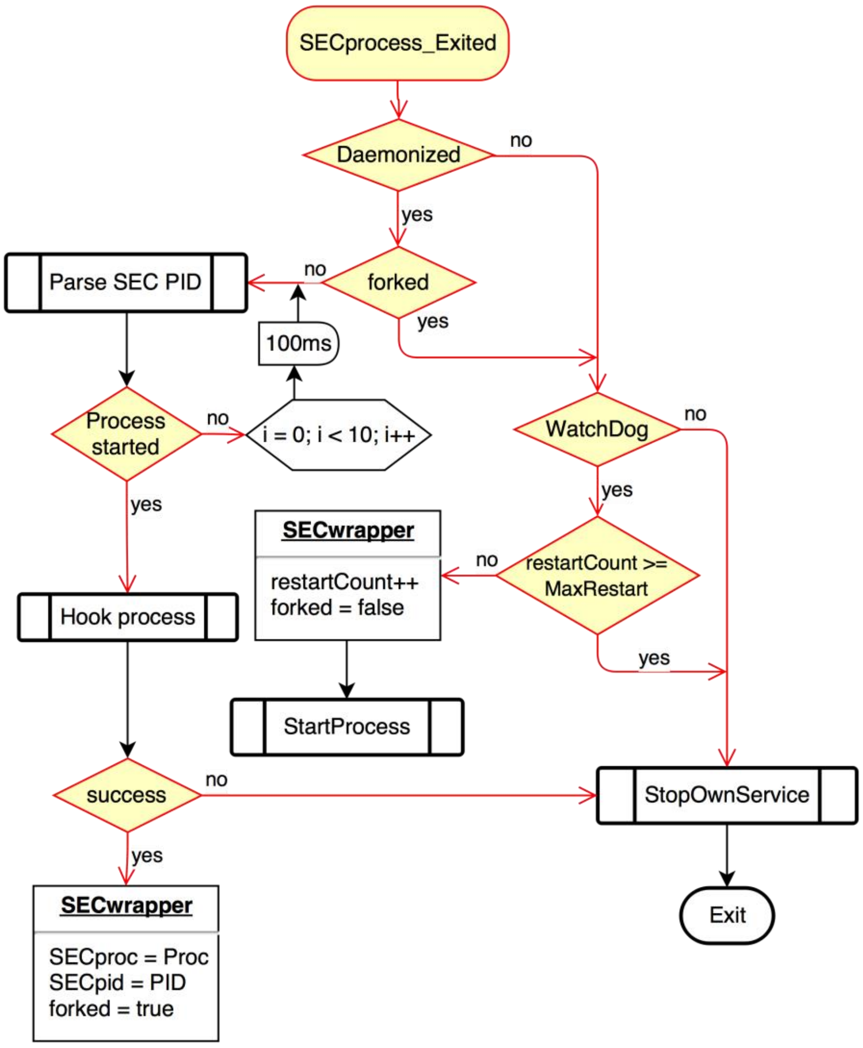


Figure 11. SECwin flowchart: Handling unexpected termination

Figure 11 illustrates the flowchart for handling SEC Perl process exit. Depending on Perl distribution type and the used SEC parameters, the logic can differentiate between unexpected termination and a graceful termination resulted from fork action. If *detach* SEC parameter is used in combination with Cygwin Perl, *Daemonized* – a Boolean value in the configuration class constructed by combining both of the conditions, will change the execution logic for handling SEC process exit. If the process is *daemonized* and is not yet *forked* – new process is not created yet, SECwin will detach from the exited process, reads new PID from SEC file, then attaches the new process. If process is *daemonized* and was already *forked* – new process has been created earlier, or not *daemonized*, that indicates an unexpected termination. In the latter case, if SECwin watchdog feature is enabled, it will restart SEC process, while the restart attempts are less than the threshold of restart counts.

3.3.6. Sending interrupt signals

Sending interrupt signal to SEC is only supported by Cygwin Perl for its UNIX emulation feature. It is concluded from section 3.3.4 that sending interrupt signals is performed by executing new process of Cygwin program kill.exe. Unfortunately it was found by practice that the exit code from Cygwin kill is not reliable when captured from Windows domain and there are no return values from the command upon success, except for fail error. Thus if there are no errors returned, success is assumed. Because MS Windows service context has no access to user context, to execute such process successfully from a service, *ProcessStartInfo.UseShellExecute* value must be set to false. At the time of writing this section, there is ongoing development of a new feature to emulate sending interrupt signals to other Perl distributions by using SEC ruleset which calls internal SEC subroutines *sigx_handler()* to emulate signal condition internally for SEC and *check_signals()* to handle the emulated signal status.

3.3.7. Handling orphaned SEC Perl process

An orphaned SEC Perl process is an instance which SECwin is not aware of. Usually this situation occurs as a result of unexpected termination of SECwin service process, as graceful termination zeros Perl PID, which SECwin stores in registry. Another cause of orphaned SEC processes is a failed graceful termination of SEC during SECwin service stopping. To handle this scenario, all running processes are evaluated during SECwin service start. If a Perl process is detected running SEC script from same path configured by SECwin, it is considered orphaned. If SEC command parameters are identical to SECwin configuration, the orphaned process it attached to SECwin, and then SIGHUP signal is sent. If attaching fails or the orphaned process parameters do not match SECwin configuration, SECwin will attempt to exit the process gracefully, if that fails, the process is killed ungracefully.

As SECwin service is considered a sensitive operation, it is monitored by another SECwin process instance using CLI *SECwin.exe /svcWatchDog*. SECwin service watchdog is also monitored by the service process. That means there are two processes monitoring each other for failure.

3.3.8. SECwin service status monitor

Microsoft .NET framework does not provide a service monitor class, though it provides the ability for a thread to suspend, waiting for a service to be in a specific state. The author has taken advantage of such feature and developed a new class, which can monitor a service using *EventWaitHandle* and *ServiceController.WaitForStatus(enum ServiceControllerStatus)* methods in a multithreaded operation. Each thread will be monitoring one of the seven possible Windows service statuses. On status change, the suspended thread loop proceeds and a delegate event is raised. This class is used by simple initialization and event listener to any or specific service status.

```
ServiceWatch SECwinStatusWatch = new ServiceWatch("SECwin");
SECwinStatusWatch.OnStatusChanged +=new
                ServiceWatch.StatusChanged(OnStatusChanged);
SECwinStatusWatch.EnableRaisingEvents = true;
private void OnStatusChanged(ServiceController sc,
                                  ServiceWatchEvtArgs args)
{
    this.Invoke((MethodInvoker)delegate
       {
           //Handle status change
       });
}
```

Figure 12. SECwin code: Service watch class usage

The code snippet in Figure 12 demonstrates the use of the developed class for watching SECwin service status. The class is used by the watchdog of SECwin service, and for displaying notifications to administrator on service status change.

3.3.9. Windows paths case insensitivity effect on Cygwin Perl

Unlike UNIX systems, Windows paths are case insensitive. Cygwin emulation of UNIX environment requires windows paths modification to case sensitive. Otherwise any attempt of accessing files or folders will fail. Additionally, Cygwin emulates drives as mounts in */cygdrive/x*. Converting a Windows path to a Cygwin compatible path is performed in two stages, first by converting the path to case sensitive, then prepend Cygwin drives mount folder.

```
static String CaseSensetiveDirPath(DirectoryInfo dirInfo)
{
  DirectoryInfo parentDirInfo = dirInfo.Parent;
  if (null == parentDirInfo)
        return dirInfo.Name;
  return Path.Combine(CaseSensetiveDirPath(parentDirInfo),
        parentDirInfo.GetDirectories(dirInfo.Name)[0].Name);
}
```

Figure 13. SECwin code: building case sensitive path

The code snippet in Figure 13 demonstrates a recursive programmatic method building a case sensitive path, using *DirectoryInfo* object from .NET framework.

```
try
{
  string parent = path.Substring(0, path.LastIndexOf("\\"));
  string files = path.Substring(parent.Length +1,
                            path.Length - parent.Length-1);
  DirectoryInfo dirInfo = new DirectoryInfo(parent);
  files = CaseSensetiveFile(files);
  string CaseSensetivePath = (dirInfo.Exists) ?
        Path.Combine(CaseSensetiveDirPath(dirInfo),files):path;
  string drive = path.Substring(0, 1).ToLower();
  string CygwinizedPath = CaseSensetivePath.Remove(0, 3);
  CygwinizedPath = CygwinizedPath.Replace('\\', '/')
                          .Replace(" ", "\\ ");
  return string.Format("\"/cygdrive/{0}/{1}\"", drive,
                          CygwinizedPath);
}
catch (Exception ex)
{
  Log.ToWinEvent(string.Format("Error Cygwinizing
           path:({0})\n\nException: {1}", path,ex.ToString()),
                 EventLogEntryType.Error);
  return path;
}
```

Figure 14. SECwin code: Converting Windows path to Cygwin style

The case sensitive path is converted to Cygwin style path. In Figure 14, a code snippet to prepend Cygwin drives mount and correct folders separator to slash. The path end is checked, if it is a folder, an existing file, or files filter using asterisk.

3.3.10. SEC statistics dump file rotation

During the time of SECwin development, SEC was creating a single statistics dump file and overwrites it each time it receives SIGUSR1 to dump statistics. Thus, keeping historical statistics information required manual copying and renaming each time a dump file is created. SECwin implemented statistics dump file rotation, which copies the file to a different destination, while appending to its name the creation timestamp. A feature to specify maximum amount of rotated files is configurable by the user interface. When maximum allowed count of rotated files is reached, the oldest files get deleted based on actual creation time not the tailed timestamp in file name. SEC has integrated similar approach later on, inspired by SECwin feature. SEC enables appending UNIX timestamp to file name if parameter *dumpfts* is used, however it does not provide the ability to set the maximum amount of files to keep.

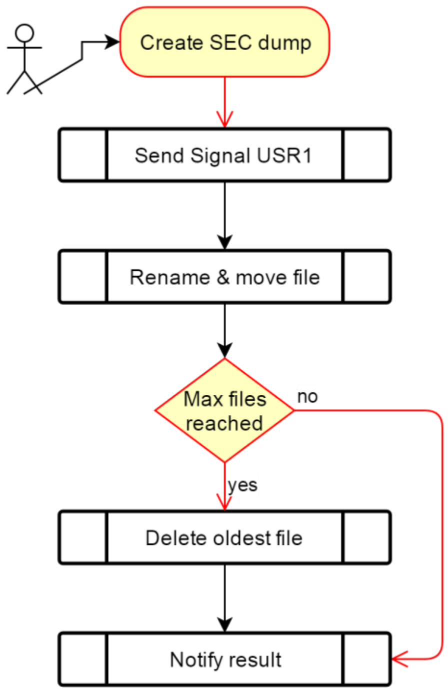


Figure 15. SECwin flowchart: SEC dump file roration

The flowchart in Figure 15 shows the basic actions taken when end user sends USR1 signal to SEC process via SECwin user interface. Statistics dump files are stored and rotated in a dedicated folder.

3.3.11. Auto-update feature

Automated update feature is divided into two stages for both applications, SEC and SECwin. First stage is checking for update availability using github.com API, and then casting JSON response to objects for gathering full information about latest release, as Figure 16 shows. The current version of the installed application is compared against the latest release information. If latest release version value is higher than the currently installed one, the second stage is initiated by executing a new SECwin process instance for downloading and installing the newer version.

Updating SEC comes with a small technical challenge. Microsoft .NET framework lacks a native tarball handler, which is required for unpacking SEC tarball. This issue was resolved by using a third party .NET class [61].

Checking for available updates is performed on the basis of a configurable period of time, every one day by default, as well as on SECwin service start and on user interface launch if not disabled by user. The update process is performed by launching new process instance of SECwin in console mode, to be able to control the service running state without interruption of the update process. End users can disable the automated update and update manually using SECwin CLI, to avoid unexpected service interruption.

Updating SECwin package is performed under two conditions, depending on the file extension of latest release, which is retrieved from the update server. If an update only affects the SECwin portable executable, it is published as the first file in the release assets list. Only the PE file will be downloaded, and then installed via console command, as demonstrated in section 3.3.2. This method saves bandwidth and performs a fast update. The other variant is when many files are affected, such as including a newer Perl release, or adding more Cygwin binaries. A full SECwin package upgrade requires downloading the latest release of MSI package, which is installed silently using Windows native installer *msiexec*.

```
WebClient webClient = new WebClient();
webClient.Encoding = Encoding.UTF8;
webClient.Headers.Add("User-Agent", "Mozilla/5.0 (Windows NT
6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)");
Log.ToDebugger("Checking for update at: " + this._gitURL);
String JSONstr = webClient.DownloadString(this._gitURL);
Log.ToDebugger("WebClient response:\n" +
                    webClient.ResponseHeaders.ToString());
JavaScriptSerializer jss = new JavaScriptSerializer();
GitRelease gitrelease = jss.Deserialize<GitRelease>(JSONstr);
UpdateCheckResultEventArgs UpdateResultArgs = new
                UpdateCheckResultEventArgs(this._AppAlias,
                              this._currentVersion, gitrelease);
if (UpdateCheckResult != null)
    UpdateCheckResult(this, UpdateResultArgs);
```

Figure 16. SECwin code: Update check

The code snippet in Figure 16 demonstrates a call to github.com API domain. Github API is based on REST, returning responds in JSON markup. The returned JSON string is serialized to a custom objects. Object composition is used for serializing release structure. The object *GitRelease* contains other custom objects, such as *ReleaseAssets* - a list of assets which represents full information about files in the release.

3.3.12. Roadmap

The current roadmap for SECwin package is to add an aditional tab, which provides a categorized collection of SEC rulesets, retrieved from online repository. That would allow users to download ready rulesets.

An under investigation feature is including NXLog-ce installer, its configuration file and script to enable critical audit logs to SECwin MSI package. Adding auto-update feature for NXLog-ce, comes with a challenges. NXLog-ce does not use github or any equivilant hosted API to retrieve latest version information. Checking for update availability can be achieved by parsing the download page at nxlog.org website, which is considered an unreliable solution .

Adding the ability to simulate SEC signals via rulesets, when other Perl distribution is used instead of Cygwin Perl, is current work in progress.

During the design stage of SECwin, the author has missed the feature of running multiple instances of SEC simultaneously. This feature was not found to be a launch stopper, thus it is reserved for future release as it requires few design changes.

3.4. Log collection tool configuration and infrastructure

A log collection tool is required to read logs from several sources, transfer logs from client to server as required and normalize logs formats.

3.4.1. Log normalization

Log entries shall be normalized to facilitate event information extraction. The author has adopted IETF syslog [26], for its wide usage, utilization of layered architecture, which allows several transmit protocols, providing vendor-specific extensions and detailed description of minimum requirements for messages transporting. While performing log normalization via NXLog-ce is very simple, it becomes a challenge in case of normalizing unknown raw event format by the log normalizer. This case requires manual normalization using regular expressions side by side with other native NXLog-ce format parsers, For example, to determine the start and end indexes of an event in markup language. A practical example of this method is normalizing *Symantec Management Agent* logs. Events of Symantec EPM agent are in custom xml format, unlike the traditional way, xml attributes are used instead of nodes.

```
<event date='03/27/2016 08:00:09.8860000 +03:00' severity='4'
hostName='MG75LAB' source='NetworkMonitor'
module='AeXNetMon.dll' process='AeXNSAgent.exe' pid='4432'
thread='2304' tickCount='498743886' >
  <![CDATA[Server up [0x00010001]:
https://mg75lab:443:{6E97D571-F3C1-4ec4-AEDE-DD9719B02277}]]>
</event>
```

Figure 17. Symantec Management Agent log sample

An xml log entry sample is shown in Figure 17 from Symantec Management Agent logs. Because the event field-value pairs are in custom format, the log entry in such case usually passes by four stages for normalization as following:

1. Enabling multiline parser and defining header and end lines.

```
<Extension multiline>
    Module   xm_multiline
    HeaderLine  /^<event.*/
    EndLine /^<\/event>/
</Extension>
```

Figure 18. NXLog-ce configuration: multiline defining

2. Parsing input as xml, using *xm_xml* module and drop mismatches, such as line breaks.

```
<Input AexAgent_in>
    Module  im_file
    File  "c:\ProgramData\Symantec\Agent\Logs\Agent.log"
    SavePos FALSE
    ReadFromLast TRUE
    InputType   multiline
    Exec    if ($raw_event !~ /^<event/) { drop(); } \
            else { parse_xml(); }
</Input>
```

Figure 19. NXLog-ce configuration: parsing multilined xml input

3. Use regular expression to extract fields' values

```
<Processor parse_AexAgent_IETF>
    Module  pm_null
    Exec    if $raw_event =~ /date=\'(\S+\s\S+)\s\S+\'
severity=\'([0-9]{1,2})\' hostName=\'(\S+)\' source=\'(.*)\'
module=\'(.*)\' process=\'(\S+)\' pid=\'(\d+)\'
thread=\'(\d+)\'
tickCount=\'(\d+)\'\s?>\s+<!\[CDATA\[((.*\R?)+)\]\]/ \
....
    Exec    to_syslog_ietf();
</Processor>
```

Figure 20. NXLog-ce configuration: SMA log entry parsing

4. Output to IETF syslog format using native normalizer as shown in Figure 20 using *to_syslog_ietf()*.

3.4.2. Distributed logging infrastructure

On each node within an environment, NXLog-ce is installed to collect events from various sources. These logs are normalized to IETF syslog and saved on disk to serve as inputs for SEC, which correlates events and triggers actions locally. The distributed logging architecture represents a part of the proposed implementation by forwarding critical and synthetic events to a centralized log server.

3.4.3. Centralized logging infrastructure

In most cases, when dealing with logs on a corporation level, centralized log collection is required for administration, forensics, reporting and analysis purposes. This study focuses on event correlation, thus, the main purpose of establishing a centralized logging infrastructure is to achieve event correlation within the entire network scope. A typical centralized infrastructure would be a client to server relation, where logs are transferred from clients to log server. Log entries contain confidential information, thus, transporting logs must be secured. Secure transportation of syslog formatted events is governed by RFC5425 [62]. Transporting events through unsecured connection exposes the environment to several types of attacks as described in RFC5425 section two. Types of attacks on unsecured transportation of logs are:

- Masquerade: an unauthorized transport sender may send messages to a legitimate transport receiver, or an unauthorized transport receiver may try to deceive a legitimate transport sender into sending syslog messages to it.
- Modification: an attacker between the transport sender and the transport receiver may modify an in-transit syslog message and then forward the message to the transport receiver. Such modification may make the transport receiver misunderstand the message or cause it to behave in undesirable ways.
- Disclosure: an unauthorized entity may examine the contents of the syslog messages, gaining unauthorized access to the information. Some data in syslog messages is sensitive and may be useful to an attacker, such as an SID or session hash of an authorized administrator or user.
- Message stream modification: an attacker may delete one or more syslog messages from a series of messages, replay a message, or alter the delivery sequence. The syslog protocol itself is not based on message order. However, an event in a syslog message may relate semantically to events in other messages, so message ordering may be important to understanding a sequence of events.

- Denial of Service: an attacker might cause log servers to stop performing their tasks of collecting log messages.
- Traffic Analysis: an attacker may perform espionage on the traffic and get detailed infrastructure information.

To secure log transportation in compliance with RFC5425 section three [63], the proposed implementation relies on a self-signed certificate authority, an optional but recommended intermediate CA, server certificates and client-side certificates. The use of client-side certificates guarantees the authenticity of log entries from clients as the server requires trusted client-certificate to establish the connection prior the acceptance of any log entries. Although NXLog-ce *im_ssl* module supports disabling the requirement for client-side certificate, it is highly recommended not to disable *RequireCert* option, as it exposes the log server to logs exploitation.

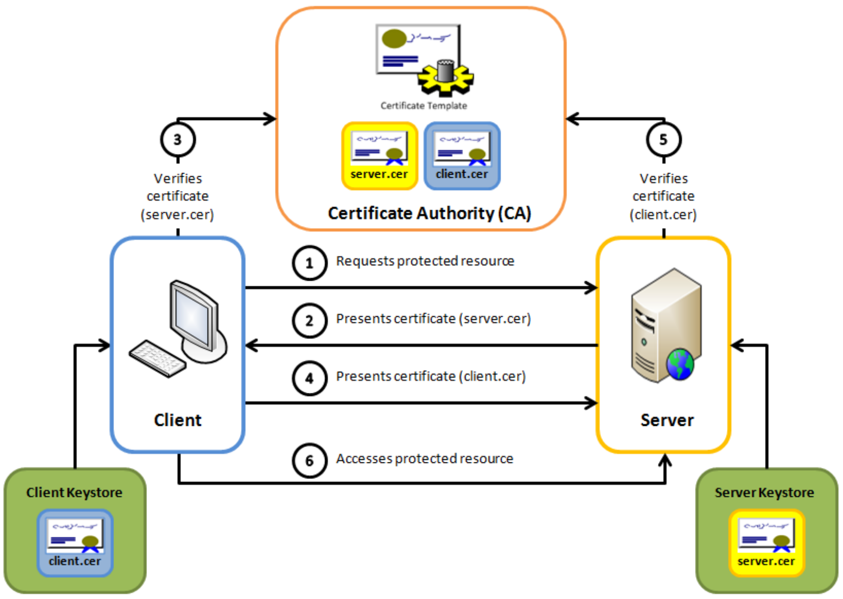

Certificate Template
server.cer
client.cer
Certificate Authority (CA)
3
Verifies certificate (server.cer)
5
Verifies certificate (client.cer)
1 Requests protected resource
2 Presents certificate (server.cer)
4 Presents certificate (client.cer)
6 Accesses protected resource
Client
Server
Client Keystore
client.cer
Server Keystore
server.cer

Figure 21. Mutual SSL certificate based authentication

Figure 21 illustrates a mutual SSL certificate based authentication. During the authentication process both, server and client, provide digital certificate which is validated on the other node against the local store as configured in NXLog-ce configuration file. While *RequireCert* option is enabled on the log server within NXLog-ce configuration, any client attempts to authenticate without a certificate or with an invalid certificate its authentication is denied.

Building certificates chain for SSL mutual authentication can be done using OpenSSL for Windows 0.9.8h [64]. To create a certificate chain, the use of configuration files in appendices 2.1, 2.2 are required for identifying certificate requesting, creation and signing, based on the following steps:

1. Prepare files and directory structure

```
>md ca\private ca\certs ca\newcerts ca\crl
>type nul > ca\index.txt
>type nul > ca\rand
>echo 01 > ca\serial
>md intermediate\private intermediate\certs
intermediate\newcerts intermediate\crl
>type nul > intermediate\index.txt
>type nul > intermediate\rand
>echo 01 > intermediate\serial
```

Figure 22. OpenSSL: Structure preparation

2. Creating self-signed Certificate Authority

```
>openssl genrsa -aes256 -out ca\private\ca.key 8192
>openssl req -config ca\ca-openssl.cnf -key
ca\private\ca.key -new -x509 -days 7300 -sha256 -extensions
v3_ca -out ca\ca.pem
```

Figure 23. OpenSSL: Creating Certificate Authority

3. Creating Intermediate certificate (optional)

```
>openssl genrsa -aes256 -out
intermediate\private\intermediate.key 4096
>openssl req -config intermediate\intmdt-openssl.cnf -new -
sha256 -key intermediate\private\intermediate.key -out
intermediate\intermediate.csr
>openssl ca -config ca\ca-openssl.cnf -extensions
v3_intermediate_ca -days 3650 -notext -md sha256 -in
intermediate\intermediate.csr -out
intermediate\intermediate.pem
>copy /b intermediate/intermediate.pem + ca/ca.pem
intermediate/chain.pem
```

Figure 24. OpenSSL: Creating Intermediate Certificate

It is highly recommended to create an intermediate CA for each group of clients.

4. Certificate Revocation List (mind *crlDistributionPoints* value in config file)

```
>openssl ca -config ca\ca-openssl.cnf -gencrl -out
ca\crl\ca-crl.pem
>openssl ca -config intermediate\intmdt-openssl.cnf -gencrl
-out intermediate\crl\intermediate-crl.pem
```

Figure 25. OpenSSL: Creating Revocation List

- Check *crlDistributionPoints* value in config file
- Check *default_crl_days* value, CRL must be renewed periodically
- Each Intermediate CA must have its own CRL and configuration

5. Creating a server certificate

```
>openssl genrsa -aes256 -out certs/server/FQDN001.key 2048
>openssl req -config certs/intmdt-openssl.cnf -key
certs/server/FQDN001.key -new -sha256 -out
certs/server/FQDN001.csr
>openssl ca -config certs/intmdt-openssl.cnf -extensions
server_cert -days 365 -notext -md sha256 -in
certs/server/FQDN001.csr -out certs/server/FQDN001.pem
```

Figure 26. OpenSSL: Creating server certificate

6. Creating a client certificate

```
>openssl genrsa -aes256 -out certs/client/client001.key 2048
>openssl req -config certs/intmdt-openssl.cnf -key
certs/client/client001.key -new -sha256 -out
certs/client/client001.csr
>openssl ca -config certs/intmdt-openssl.cnf -extensions
client_cert -days 365 -notext -md sha256 -in
certs/client/client001.csr -out certs/client/client001.pem
```

Figure 27. OpenSSL: Creating client certificate

In the steps provided for creating the required digital certificates chain, Certificate Revocation List creation in Figure 25, is critical. In case one or more of the certificates private keys, in particular client certificates keys, are exposed to an attacker. To mitigate the risk of logs exploitation by any of the means described in section two of RFC5435 compliance, the exposed clients' certificates or the intermediate certificate must be revoked immediately. Clients' certificates are in higher risk for exposure for their wider distribution on each logs forwarding node. Figure 28 provides the required OpenSSL commands for certificates revocation. Revocation shall be followed by replacing of CRL file on log servers to overcome the lack of CRL URI validation and total lack of OCSP protocol support by NXLog-ce.

- Revoking a certificate and create CRL

```
>openssl ca -config intermediate/intmdt-openssl.cnf -
revoke intermediate/newcerts/01.pem
>openssl ca -config intermediate/intmdt-openssl.cnf –
gencrl -out intermediate/crl/intermediate-crl.pem
```

Figure 28. OpenSSL: Revoking a certificate

During the validation of client-side certificate, typically a server should check the certificate’s CRL / OCSP values then perform a query according to the used protocol to ensure the validity of the certificate. While CRL is an old method for digital certificates revocation inquiry, at this point, speaking of NXLog-ce 2.0.928, it does not support OCSP protocol, which adds an extra burden to periodically update CRL on each node.

3.5. SEC rules

To facilitate the creation and editing of SEC rules, the author built a syntax-highlighter configuration file for *Notepad++*, packed within SECwin MSI package and also available on github SECwin project page.

3.5.1. Simple Event Correlator rules and best practices

Simple Event Correlator is a rule-based [36], CLI correlation engine written in Perl. SEC loads its configuration by command parameters or from a text file, and loads its correlation rules from one or more text files, in format of key-value pairs. Each file can contain one or more rules, where rulesets from several files are applied virtually in parallel [65]. SEC is a single threaded application, therefore, several instances can run in parallel within separated domains to leverage multi-core CPUs [66] and achieve higher scalability. Such scale has been practically proven to process hundred thousand event log per second [67].

SEC best practices [68] paper was published in 2015 by the author of SEC. The paper provides methods to leverage the full power of the correlation engine with minimal computing resources utilization. The paper highlights the following methods:

- Joining rules into event correlation schemes

Using contexts, synthetic events and other data sharing measures, leverage joining several rules together into more powerful event correlation scheme. Rules can create, modify or delete contexts, while other rules validate those contexts side by side to event matching. Another way is generating synthetic events by ruleset which triggers other set of rules.

- Advanced event matching using Perl function

Using regular expressions allows flexible parsing of event, but it has drawbacks, such as, intensive resources utilization and the lack of mathematical operations capability. SEC allows the use of Perl function as pattern type, this feature allows a very flexible way to match events and leverage Perl modules, to perform evaluation process such as defining the network of an IP address.

- Using named match variables and match caching

SEC supports named match variables and match caching, to minimize resources utilization by using variable maps and cached pattern types. For example one rule can parse an event using a Perl function and then creates a hash table of field-value pairs. The hash table can be used by other rules for matching. This method avoids redundant parsing of events.

- Arranging rulesets hierarchically

Normally, each event from all input sources is matched against all rules, which exhausts computing resources when using big sets of rules. Using hierarchically arranged rulesets allows triggering desired rulesets only for specific inputs. SEC best practices guide addresses this issue in the following ways:

a. By command parameter *–intcontext*, this creates a temporary context, which reflects the source of the event. Thus, if some rulesets are designed to match events from specific source, they use the temporary context, which is validated prior to pattern.
b. By using *Jump* rule type, to submit input events to specific rulesets for further processing, where desired rulesets can be configured to accept input from *Jump* rules only.

3.5.2. SEC rules for MS Windows event logs

An event correlation library can be built with the aid of an official description of security events [69] from Microsoft support knowledge base, or more detailed papers, which target detecting security incidents using MS Windows event logs [70]. Also hardening guides for MS Windows with the aid of event logs [71], represents a source for administrators depending on the corporate infrastructure. Depending on the corporate security policies, system administrators can deploy distributed event correlation rules which perform local actions and forward critical and synthetic events to a centralized server to trigger wider action and for wider security audit.

Many security events are disabled by default, however it depends on the distribution, as MS Windows server family has some audit logs enabled by default unlike workstation family, for example *Account Logon* and *Account Management*. Enabling advanced audit logs is mandatory for giving an insight of occurring incidents.

Enabling advanced audit logs is done via group policy in a network managed by a domain controller or by *Local Security Policy* console or *auditpol* command for standalone computers. Microsoft provides a guide to monitor signs of compromise [72], which discusses in details, which audit policies to enable and means of achieving that in different way. To validate currently applied security audit settings, *auditpol /get /category:** command provides more accurate results [73] than *Local Security Policy* console.

A more advanced events logging can be forced by using *Sysmon* [12] tool from *Sysinternals* suite. The tool allows logging more details about processes creation, in addition to other suspicious activities on the system, such as, changes in files creation timestamps, loading of drivers, open of raw read of disks and other malicious actions. *Sysmon* tool is installed as a system driver, to generate events during the operating system boot process which allows detection of kernel-mode malware.

Another essential tool from *Sysinternals*, is *Sigcheck*, which has the ability to perform a virus scan by virustotal.com using command line. The output is correlated to previous events and accordingly and action is triggered.

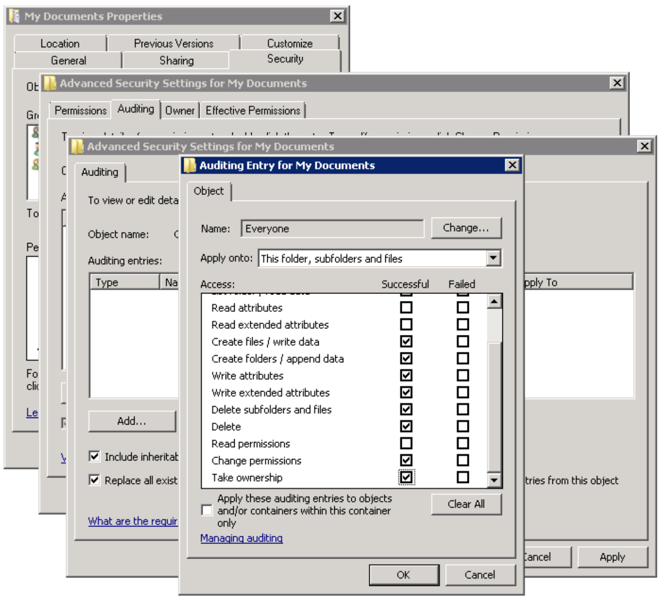
- Detecting malware infection

Usually sophisticated malware gets repacked to change its file signature and to avoid detection by antimalware applications. Detecting malware infection activity is possible with thye aid of *Sysmon* and *Sigcheck* tools from *Sysinternal* suite. Usually, when a malware launches for the first time it copies itself to a new location in system folders, then attempts to register the new file to auto-run on system reboot. SEC ruleset in appendix 3.1 detects such malicious activity when a launched process by a valid system application, such as *Explorer*, creates an additional process. That action is a normal case whenever a user launches an application, but when the launched application starts an additional process, that additional action can be considered an abnormal activity. If both second and third processes hash matches, or if a file is created inside system folders, then files are checked using *Sigcheck* tool for signature and hash with aid of virustotal. If the files are not signed or are signed with an invalid certificate, or their hashes match with a malware in virustotal repository, their processes are killed and an alert is generated. If the log server receives more than three similar events from different nodes, the network connection must be disabled for those nodes to prevent further spread of the malware.

- Detecting ransomware (crypto-locker) activities.

Adopting the same concept for detecting malware infection, ransomware-like suspicious activities can be detected. Ransomware encrypts files on a system then requests a ransom to provide the decryption key. Original files are either deleted or overwritten with the encrypted version. To detect such malicious action, *Object Access – Audit File System* success logging, must be enabled. On the folder to be monitored, auditing must be enabled as well, by folder properties, Security tab - Advanced, Audit tab - Edit, Add *Everyone* account, then enable successful access as shown in Figure 29.

Once these configurations are applied on a system, all write activities performed on the monitored folder are logged in Windows Security Audit event logs. Information logged includes, what object has been changed, in which way, by who (user) and which process. Such events are correlated for reoccurrence and any suspicious process performs changes on those files is killed. If the process is signed and trusted, an alert is generated and optionally, a handle can be opened to all objects inside the monitored folder recursively, to prevent further changes till a human interacts with the system to approve the process. Such event correlation ruleset is provided as an example in appendix 3.



Name: Everyone

Apply onto: This folder, subfolders and files

Access:	Successful	Failed
Read attributes	☐	☐
Read extended attributes	☐	☐
Create files / write data	☑	☐
Create folders / append data	☑	☐
Write attributes	☑	☐
Write extended attributes	☑	☐
Delete subfolders and files	☑	☐
Delete	☑	☐
Read permissions	☐	☐
Change permissions	☑	☐
Take ownership	☑	☐

☐ Apply these auditing entries to objects and/or containers within this container only

Figure 29. Object audit for file changes

- Detecting system compromise and suspicious actions

While Microsoft provides list of events to monitor [74], further actions which can be considered suspicious, shall be monitored as well. Appendix 3 provides some examples such as detecting access to a hidden network share, detecting network login using a local administrator account and Detecting privilege escalation of process then evaluate the process authenticity.

An example of suspicious action is a network login to a workstation followed by several failed attempts to access files, which indicates a possibility of credential compromise. While attacker might be attempting to access unauthorized local files.

- Mitigating spread of malware

Whenever a malicious action is detected locally on a machine, the local correlation engine handles the threat by killing suspected process. SEC generates synthetic events which are forwarded by NXLog-ce to log server. The log server correlates all received events for unusual traffic or failed login attempts on other nodes from the infected machine then triggers an action by isolating the infected node. Further network wide action can be performed, such as, taking an image of the operating system for further analysis.

3.5.3. SEC rules for Symantec Endpoint Management

Symantec Endpoint Management [75] is commercial software with several products provided as suite for endpoints inventory, software installation, patch management, OS deployment etc. the platform which acts as the core component is a free software which can still be used to manage endpoints. Like other endpoint management software it depends on client side application. Because of the critical operations handled by the client side such as installing updates, security patches, software etc. proactive detection of any interruption or faults occurring provides a value to system administrators.

- Detecting failed client to server communication

Client agent logs are monitored for failed communication and crashes, if a crash is detected its memory dump is uploaded for further investigation, and if a failed communication or registration is detected a notification is sent to administrator.

- Detecting failed registration to task service

Clients are required to register to their site task server, if a client is not registered fail reasons are send to the main server. Forwarded events are correlated and evaluated against the site task server; if many clients are affected within specific site the main server resets the site task server services to force connections.

- Quality Assurance Test Automation

During the Quality Assurance cycle of releases prior to general availability, thousands of test automation tasks are performed. Integrating event correlation to the process provides more efficiency.

3.6. Final implementation layouts

The final deployment on a standalone MS Windows machine is very simple. First step is installing NXlog-ce and SECwin MSI. Second step is deploying SEC rulesets files and NXLog-ce configuration file then run required commands to set audit policies. A

VB script or batch script can be used to perform all these actions as per the provided example in appendix 4.1.

The deployment within a network requires more steps for securing the logs transportation to the log server by creating required certificates chain, which was covered precisely in section 3.4.3. Deploying software to workstations can be done via group policy [76], or end point management software if present.

Each node within a network shall receive software installation policy or script for installing SECwin MSI, NXLog-ce MSI and selective Sysinternal suite tools (Sysmon, Sigcheck etc.). Second step is deploying NXLog-ce configuration file and certificates, SEC rulesets and SECwin configuration registry file.

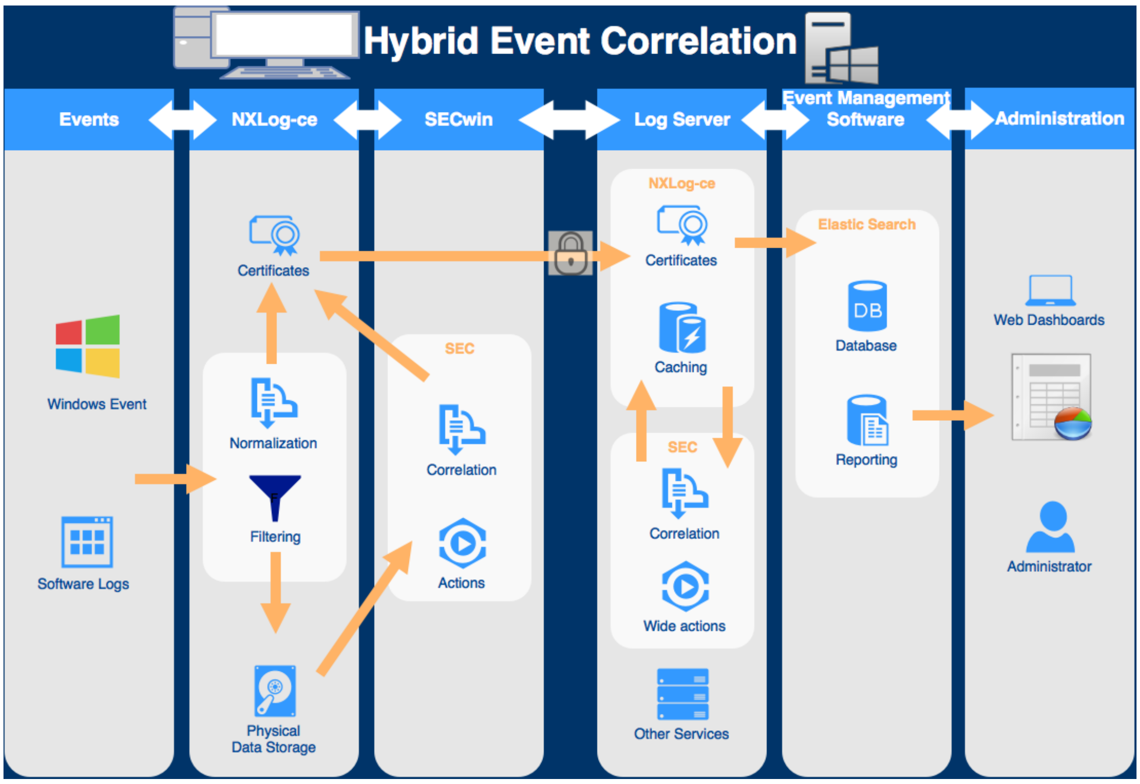


Figure 30. Hybrid event correlation architecture layout

Figure 30 illustrates the proposed layout implementation of a hybrid event correlation architecture. On workstations, NXLog-ce captures events from various sources then normalizes to Syslog format. NXLog-ce saves all normalized events to disk and filters events to forwards critical ones to log server. SEC reads locally stored normalized events as input then correlates events according to local rulesets, and then triggers actions locally to mitigate threats. Correlated events and triggered actions by SEC generate synthetic events stored locally on disk. NXLog-ce reads SEC generated event then forwards them to log server. The log server correlates all incoming events from all workstations, and then triggers wider actions. Optionally event management software can be used, where the log server can feed it with all captured events for storage and generating reports.

3.7. Practical implementation use case

The layout architecture proposed in section 3.6 has been practically applied for Symantec EPM testing automation. Example implementation architecture is multi-tier Symantec EPM servers hierarchy consists of a parent and 3 children servers. Each child server has two site servers and eight clients. All nodes are joined to the same domain controller. SECwin, NXLog-ce and their configuration files were installed on all nodes using software delivery policy from parent server. Logs normalization is performed on clients to balance resources consumption. Logs are transported between nodes in binary mode to allow filtering on destination based on fields' values without reparsing. Received events are saved on different files based on node name and Message ID.

Several policies and tasks are created programmatically on the parent server for test automation purposes, such as, software and hardware inventory policies, software patches policies, and then replicated down the hierarchy to children server to be applied on their clients. The correlation engine residing on the endpoint clients correlates Windows event logs and Symantec EPM client logs for critical incidents then trigger local action such as, restarting services, killing hanging process etc. Clients' critical events are transported to their server, which correlates all received events side by side with its events including IIS logs, then accordingly triggers wider actions, such as refreshing its hosted software packages snapshots, when many clients fail to download them or redirects clients to a different site server if theirs is faulty. Child servers transport critical, actual and synthetic events to the parent server, which correlates all events at its end. If faults are noticed, the failed operations are repeated. All events received by the parent server are passed to ElasticSearch for storage and generating reports.

During testing automation, a lot of failed operations are expected, which produce enormous amount of events, especially when log level is set to verbose for debugging purposes. The applied implementation used the child servers as correlation gateways to balance resources load.

The applied implementation adopted the hybrid event correlation approach with 34 correlation engines in total, and has proven success for processing thousands of events per second, distributed on all nodes with minimal resource consumption.

4. Conclusion

The proposed implementation promotes free software and does not require any additional hardware or dedicated resources. Home users and small to medium sized corporates with limited budgets can benefit from the proposed implementation.

Simple Event Correlator was proven to work reliably on Microsoft Windows operating system using Cygwin Perl and SECwin. However it is very important to follow SEC best practices especially on large scales.

Using event correlation hybrid approach represents a robust and reliable solution for detecting various types of incidents, whether these incidents are security related or critical alerts. The distributed part shall be capable of handling local incidents, while the centralized part can correlate wider set of incidents from various sources and mitigate threats before breaches occur. On the level of home users, using an event correlation to detect malicious actions adds an additional security dimension. Usually regular users do not regularly examine the logs on their computers, and mostly would not have the skills to extract critical information. Thus, proactive event correlation can be considered as critical as antimalware solutions for home users and small to medium corporates.

While this study presented an implementation layout for the proposed solution, it is by no mean the only way such solution can be implemented. Depending on the activity of a corporate, and its security policies, free and open source log management solutions like Elastic and LOGalize can be integrated to build a SIEM solution with powerful proactive correlation capabilities with no cost. The applied implementation documented in section 3.7 has used event correlation gateways to balance the load, and successfully processed thousands of events per second with minimal resources consumption.

It is required to spread the awareness of undetectable threats, like repacked and new malware, and spot the light on the concept of using proactive event correlation for enhancing security, and various IT operations.

This study serves as a gateway in the field of event correlation and attack pattern recognition on Microsoft Windows platform. Further SEC rulesets can be built for malicious actions detection, to expand the current published SEC rulesets.

Bibliography

[1] StatCounter, "Top 7 Desktop Oss from July 2008 to Dec 2015," [Online]. Available: http://gs.statcounter.com/#desktop-os-ww-monthly-200807-201602-bar. [Accessed 2016].

[2] J. Shenk, "Ninth Log Management Survey Report," *SANS survey : InfoSec Reading Room,* October 2014.

[3] R. Vaarandi, "SEC – a Lightweight Event Correlation Tool," in *IEEE/IFIP Network Operations and Management Symposium, pp. 907-910,* 2002.

[4] The Perl Programming Language, "Download Perl Distribution," Perl.org, [Online]. Available: https://www.perl.org/get.html#win32. [Accessed 2016].

[5] B. Botyanszki, "nxlog-community-edition-reference-manual-v20928," NXLog.oeg, December 2009. [Online]. Available: https://nxlog.org/documentation/nxlog-community-edition-reference-manual-v20928#pm_evcorr. [Accessed 2016].

[6] T. Beverly, "Windows version of SEC," SEC mailing list, April 2008. [Online]. Available: https://sourceforge.net/p/simple-evcorr/mailman/message/19156366/. [Accessed 2015].

[7] Github API, "API SECwin releases," [Online]. Available: https://api.github.com/repos/minagerges/SECwin/releases. [Accessed April 2016].

[8] Sourceforge, "SECwin, download statistic - All files," [Online]. Available: https://sourceforge.net/projects/secwin/files/stats/map?dates=2015-06-

18%20to%202016-04-29. [Accessed April 2016].

[9] OWASP, "Logging Cheat Sheet," January 2016. [Online]. Available: https://www.owasp.org/index.php/Logging_Cheat_Sheet. [Accessed 2016].

[10] Microsoft, "How to use the event log management script tool," 2015. [Online]. Available: https://support.microsoft.com/en-us/kb/318763. [Accessed 2016].

[11] TechNet, "Event Log," Microsoft, October 2008. [Online]. Available: https://technet.microsoft.com/en-us/library/cc722385%28v=ws.10%29.aspx. [Accessed 2016].

[12] TechNet, "Sysmon," Microsoft, [Online]. Available: https://technet.microsoft.com/en-ie/sysinternals/sysmon. [Accessed 2016].

[13] TechNet, "Eventcreate," Microsoft, [Online]. Available: https://technet.microsoft.com/en-us/library/bb490899.aspx. [Accessed 2016].

[14] A. Sapegin, D. Jaeger, A. Azodi, M. Gawron, F. Cheng and C. Meinel, "Hierarchical object log format for normalisation of security events," in *9th International Conference on Information Assurance and Security (IAS)*, 2013.

[15] P. Sharma, S. Yadav and B. Brahmdutt, "A Review Study of Server Log Formats for Efficient Web Mining," in *International Conference on Green Computing and Internet of Things (ICGCIoT)*, 2015.

[16] J. Ya, T. Liu, H. Zhang, J. Shi and L. Guo, "An automatic approach to extract the formats of network and security log messages," in *Military Communications Conference*, 2015.

[17] R. Vaarandi and M. Pihelgas, "LogCluster - A Data Clustering and Pattern Mining Algorithm for Event Logs," in *International Conference on Network and Service Management*, 2015.

[18] J. H. a. M. Kamber, in *Data mining concept and technology*, 2007, p. 3.

[19] J. Lv, "Research on the application of web log mining," *Journal of Chonqqing Normal University,* vol. 12, no. 23, pp. 39-44, 2006.

[20] J. Kezhong and W. Chengwen, "An improved algorithm with key attributes constraints for mining interesting association rules in network log," in *Business Management and Electronic Information (BMEI)*, 2011.

[21] M. P. Yadav, P. K. Keserwani and S. G. Samaddar, "An efficient web mining algorithm for Web Log analysis: E-Web Miner," in *Recent Advances in Information Technology (RAIT)*, 2012.

[22] "Common Event Expression," Mitre, 2014. [Online]. Available: https://cee.mitre.org. [Accessed 2016].

[23] "GELF - Graylog 2.0.0 documentation," Graylog, [Online]. Available: http://docs.graylog.org/en/latest/pages/gelf.html. [Accessed 2016].

[24] A. Sapegin, D. Jaeger, A. Azodi, M. Gawron, F. Cheng and C. Meinel, "Hierarchical object log format for normalisation of security events," in *9th International Conference on Information Assurance and Security (lAS)*, 2013.

[25] F. Cheng, A. Azodi, D. Jaeger and C. Meinel, "Pushing the Limits in Event Normalisation to," in *International Conference on Advanced Cloud and Big Data*,

2013.

[26] R. Gerhards, "RFC5424 - The Syslog protocol," March 2009. [Online]. Available: https://tools.ietf.org/html/rfc5424. [Accessed 2016].

[27] MSDN: Developer technologies, "Event Log Format (Windows)," [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/bb309026(v=vs.85).aspx. [Accessed 2016].

[28] "ElasticSearch | Elastic," Elastic corp., [Online]. Available: https://www.elastic.co/products/elasticsearch. [Accessed 2016].

[29] NXLog.co, "NXLog community edition," [Online]. Available: http://nxlog.org/products/nxlog-community-edition. [Accessed 2016].

[30] Elastic, "beats | Elastic," [Online]. Available: https://www.elastic.co/products/beats. [Accessed 2016].

[31] Elastic, "Logstash | Elastic," [Online]. Available: https://www.elastic.co/products/logstash. [Accessed 2016].

[32] intersectalliance, "Snare agent for Windows," [Online]. Available: https://www.intersectalliance.com/our-product/snare-agent/operating-system-agents/snare-agent-for-windows/. [Accessed 2016].

[33] Microsoft MSDN, "Configure computers to forward and collect events," February 2015. [Online]. Available: https://msdn.microsoft.com/en-us/library/cc748890.aspx. [Accessed 2016].

[34] "State of Cybersecurity: Implications for 2015," An ISACA and RSA Conference Survey - CyberSecurity Nexus, 2015.

[35] A. Sah, "A New Architecture for Managing Enterprise Log Data," in *LISA*, 2002.

[36] G. Jakobson and M. Weissman, "Real-time telecommunication network management: Extending event correlation with temporal constraints," in *International Symposium on Integrated Network Management, pp. 290*, 1995.

[37] N. Dwivedi and A. Tripathi, "Event Correlation for Intrusion Detection Systems," in *International Conference on Computational Intelligence & Communication Technology*, 2015.

[38] S. Fu and C.-Z. Xu, "Quantifyingeventcorrelationsforproactivefailuremanagementinnetworked," *J.ParallelDistrib.Comput.*, no. 70, 2010.

[39] A. Muller, "Survey of Existing Event Correlation Approaches," in *Event Correlation Engine, MSc. Thesis*, Zurich, Swiss Federal Institute of Technology, 2009.

[40] B. Gruschke, "Integrated Event Management: Event Correlation using Dependency Graphs," in *9th IFIP/IEEEInternational Workshop on Distributed Systems: Operations and Management (DSOM)*, 1998.

[41] S. A. Yemin, S. Kliger, E. Mozes, Y. Yemini and D. Ohsie, "High speed and robust event correlation," *IEEE Communications Magazine 34(5)*, pp. 82-90, May 1996.

[42] M. Steinder and A. S. Sethi, "End-to-end Service Failure Diagnosis Using Belief

Networks," in *IEEE/IFIP Network Operations and Management Symposium*, 2002.

[43] H. Wietgrefe, K.-D. Tuchs, K. Jobmann, G. Carls, P. Froehlich, W. Nejdl and S. Steinfeld, "Using Neural Networks for Alarm Correlation in Cellular Phone Networks," in *International Workshop on Applications of Neural Networks in Telecommunications*, 1997.

[44] R. N. Cronk, P. H. Callahan and L. Bernstein, Rule-based expert systems for network management and operations: an introduction, IEEE, 1988.

[45] R. Davis, H. Shrobe, W. Hamscher, K. Wieckert, M. Shirley and S. Polit, "Diagnosis based on description of structure and function," in *American Association for Artificial Intelligence*, 1982.

[46] A. Hanemann and M. Sailer, "A framework for service quality assurance using event correlation techniques," in *AICT/SAPIR/ELETE*, 2005.

[47] J. Myers, in *A Dynamically Configurable Log-based Distributed Security Event Detection Methodology using Simple Event Correlator, MSc. Thesis*, US Air Force Institute of Technology, 2010.

[48] J. Myers, G. R. Michael and R. F. Mills, "Log-Based Distributed Security Event Detection Using Simple Event Correlator," in *44th Hawaii International Conference on System Sciences*, 2010.

[49] M. Kont, "Comparative analysis of open-source log collection and correlation tools," in *Event Management and active defense framework for small companies, MSc Thesis*, Tallinn University of Technology, 2014, pp. 24 - 44.

[50] "Log Management | Log Analysis | Splunk," Splunk Inc., [Online]. Available:

https://www.splunk.com/en_us/solutions/solution-areas/log-management.html. [Accessed 2016].

[51] ZURIEL Kft., "LOGalyze - Open Source Log Management Tool, SIEM, Log Analyzer," ZURIEL Kft., [Online]. Available: http://www.logalyze.com/. [Accessed 2016].

[52] EsperTech, "EsperTech - Esper," EsperTech, 2006. [Online]. Available: http://www.espertech.com/esper/. [Accessed 2016].

[53] TechNet, "Event Logs," Microsoft, [Online]. Available: https://technet.microsoft.com/en-us/library/cc722404.aspx. [Accessed 2016].

[54] Roth Consulting, "Official Win32::Daemon home page," 12 2001. [Online]. Available: http://www.roth.net/perl/Daemon/. [Accessed 2016].

[55] R. Vaarandi, "Simple Event Correlator faq," November 2013. [Online]. Available: http://simple-evcorr.sourceforge.net/FAQ.html#1. [Accessed 2016].

[56] M. Gerges, "SECwin: Simple Event Correlator Windows Integration," June 2015. [Online]. Available: https://sourceforge.net/projects/secwin/. [Accessed 2016].

[57] A. Perl, "Active Perl," ActiveState, 2015. [Online]. Available: http://www.activestate.com/activeperl. [Accessed 2016].

[58] StrawberryPerl, "The Perl for MS Windows, free of charge!," StrawberryPerl.com, 2015. [Online]. Available: http://strawberryperl.com/. [Accessed 2016].

[59] "Cygwin," [Online]. Available: https://www.cygwin.com/. [Accessed 2015].

[60] MSDN, "GenerateConsoleCtrlEvent function," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ms683155(v=vs.85).aspx. [Accessed 2016].

[61] Cheeso, "tar .NET class," Cheeso - CodePlex, October 2011. [Online]. Available: http://cheesoexamples.codeplex.com/. [Accessed 2015].

[62] e. a. Miao, "tools.ietf.org," IETF org., March 2009. [Online]. Available: http://tools.ietf.org/html/rfc5425. [Accessed 2016].

[63] e. a. Miao, "tools.ietf.org," IETF org., March 2009. [Online]. Available: http://tools.ietf.org/html/rfc5425#section-3. [Accessed 2016].

[64] GnuWin32, "OpenSSL for Windows," OpenSSL.org, December 2008. [Online]. Available: http://gnuwin32.sourceforge.net/packages/openssl.htm. [Accessed 2016].

[65] R. Vaarandi, Tools and Techniques for Event Log Analaysis, PhD. Thesis, Tallinn University of Technology, 2005.

[66] F. Cheng, A. Azodi, D. Jaeger and C. Meinel, "Security Event Correlation Supported by Multi-Core Architecture," in *IT Convergence and Security (ICITCS)*, 2013.

[67] D. Lang, "Building a 100K log/sec logging infrastructure," in *USENIX, Large Installation System Administration Conference*, 2012.

[68] R. Vaarandi, "Simple Event Correlator - Best Practices for Creating Scalable Configurations," in *IEEE CogSIMA Conference*, 2015.

[69] Miscrosoft Support, "Description of security events in Windows 7 and in Windows Server 2008 R2," February 2011. [Online]. Available: https://support.microsoft.com/en-ie/kb/977519. [Accessed 2016].

[70] R. Anthony, "Detecting Security Incidents Using Windows," SANS Institute - InfoSec Reading Room, June 2013.

[71] Information Assurance Directorate, "Spotting the Adversary with Windows," National Security Agency of Unitied States of America, 2013.

[72] Technet - Microsoft, "Monitoring Active Directory for Signs of Compromise," Microsoft, July 2013. [Online]. Available: https://technet.microsoft.com/en-us/library/dn487458.aspx. [Accessed 2016].

[73] Support - Microsoft, "AuditPol and Local Security Policy results may differ," May 2014. [Online]. Available: https://support.microsoft.com/en-ie/kb/2573113. [Accessed 2016].

[74] Technet - Microsoft, "Appendix L: Events to Monitor," July 2013. [Online]. Available: https://technet.microsoft.com/en-us/library/dn535498.aspx. [Accessed 2016].

[75] Symantec corporation, "Endpoint Management," Symantec corporation, 1995. [Online]. Available: https://www.symantec.com/products/threat-protection/endpoint-management. [Accessed 2016].

[76] Support - Microsoft, "How to use Group Policy to remotely install Software," December 2013. [Online]. Available: https://support.microsoft.com/en-us/kb/816102. [Accessed 2016].

Appendices

Appendix 1 – NXLog-ce configuration files

1.1. Header and extensions (modules)

```
define ROOT C:\Program Files (x86)\nxlog
Moduledir %ROOT%\modules
CacheDir %ROOT%\data
Pidfile %ROOT%\data\nxlog.pid
SpoolDir %ROOT%\data
LogFile %ROOT%\data\nxlog.log
## Modules ##
# Open WinEventLog
<Extension _syslog>
    Module      xm_syslog
</Extension>
# XML parsing
<Extension xml>
    Module      xm_xml
</Extension>
# MultiLine rowLog declaration (SMP & AexAgent)
<Extension multiline>
    Module  xm_multiline
    HeaderLine  /^<event.*/
    EndLine /^<\/event>/
</Extension>
# For rotating internal log file
<Extension fileop>
    Module  xm_fileop
    # Rotate log file every hour if larger than 1Mb
   <Schedule>
        Every 1 hour
        Exec if (file_size('%ROOT%\data\nxlog.log') >= 1M) { \
                         file_cycle('%ROOT%\data\nxlog.log', 2);
             }
    </Schedule>
</Extension>
```

1.2 Log Server (inputs / outputs)

```
<Input SSLin>
    Module  im_ssl
    Host    0.0.0.0
    Port    23456
    CAFile  C:\Program Files (x86)\nxlog\cert\ca.crt
    CertFile    C:\Program Files (x86)\nxlog\cert\server.crt
    CertKeyFile C:\Program Files (x86)\nxlog\cert\server.key
    CRLFile     C:\Program Files (x86)\nxlog\cert\CRL.crt
    KeyPass passphrase
    InputType   Binary
</Input>

<Output SSL_toDisk>
    Module  om_file
    File    'D:\ReceivedLogs\data\' + "$Hostname_$MessageID.txt"
    Exec  if SSL_toDisk->file_size() > 10M { \
            $newfile = 'D:\ReceivedLogs\Rotated'+"$Hostname_$MessageID.txt";\
            SSL_toDisk->rotate_to($newfile); \
          }
</Output>
```

1.3 Clients logs collection and normalization (inputs / outputs & Routes)

```
<Input WinEventLog_in>
    Module      im_msvistalog
    Exec    to_syslog_ietf();
</Input>

<Input AexAgent_in>
    Module  im_file
    File    "c:\ProgramData\Symantec\Symantec Agent\Logs\Agent.log"
    SavePos FALSE
    ReadFromLast TRUE
    InputType   multiline
    Exec    if ($raw_event !~ /^<event/) { drop(); } \
            else { parse_xml(); }
</Input>

<Input SMP_in>
    Module  im_file
    File    'C:\ProgramData\Symantec\SMP\Logs\a.log'
    SavePos FALSE
    ReadFromLast TRUE
    InputType   multiline
    Exec    if ($raw_event !~ /^<event/) { drop(); } \
            else { parse_xml(); }
</Input>

# IETF, BSD compliance
<Processor parse_SMP_IETF>
    Module  pm_null
    Exec    if ($raw_event =~ /date="(\S+\s\S+)\s\S+" severity="([0-9]{1,2})"
hostName="(\S+)" source="(.*)" module="(.*)" process="(\S+)" pid="(\d+)"
thread="(\d+)" tickCount="(\S+)"><!\[CDATA\[((.*\R?)+)\]\]/) \
         { \
          $EventTime = strptime($1, '%m/%d/%Y%t%H:%M:%S'); \
          $Hostname = $3; \
          $SourceName = $6; \
          $ProcessID = $7; \
          $MessageID = "SMP"; \
          if $2 == "1" { $Severity = "Error"; $AexSeverity = "Error"; } \
          else if $2 == "2" { $Severity = "Warning"; $AexSeverity =
"Warning"; } \
          else if $2 == "4" { $Severity = "Informational"; $AexSeverity =
"Informational"; } \
           else if $2 == "8" { $Severity = "Notice"; $AexSeverity = "Trace";
} \
           else if $2 == "16" { $Severity = "Debug"; $AexSeverity =
"Verbose"; } \
           if ($2 == "1" OR $2 == "2") { $SyslogFacilityValue = 14; } \
           else { $SyslogFacilityValue = 13; } \
                  $Source = $4; \
                  $Thread = $8; \
                  $TickCount = $9; \
                  $Message = $10; \
                }  \
            else { $Message = "No Regex match!"; $STATUS = "A7A neek";}
    Exec    delete($EventReceivedTime); \
            delete($SourceModuleType);
    Exec    to_syslog_ietf();
</Processor>

<Output SSL_out>
    Module  om_ssl
    Host    10.31.24.133
    Port    23456
    CAFile  C:\Program Files (x86)\nxlog\cert\ca.crt
    CertFile    C:\Program Files (x86)\nxlog\cert\client.crt
    CertKeyFile C:\Program Files (x86)\nxlog\cert\client.key
    CRLFile      C:\Program Files (x86)\nxlog\cert\CRL.crt
    KeyPass passphrase
    OutputType  Binary
</Output>

<Route WinEventLog>
    Path    WinEventLog_in => WinEventLog_out, TCP_out, SSL_out
</Route>
<Route SMP>
    Path    SMP_in => parse_SMP_IETF => SMP_out, SSL_out
</Route>

<Route Agent>
    Path    AexAgent_in => parse_AexAgent_IETF => AexAgent_out, SSL_out
</Route>
```

Appendix 2 - OpenSSL configuration files

Appendix 2.1 - OpenSSL Certificate Authority Configuration file

```
# OpenSSL root CA configuration file.
[ ca ]
default_ca          = CA_default

[ CA_default ]
dir                         = d:/certs/ca
certs                       = $dir/certs
crl_dir                     = $dir/crl
new_certs_dir               = $dir/newcerts
database            = $dir/index.txt
serial                      = $dir/serial
RANDFILE                    = $dir/private/.rand
private_key                 = $dir/private/ca.key
certificate                 = $dir/ca.pem
crlnumber                   = $dir/crlnumber
crl                         = $dir/crl/ca-crl.pem
crl_extensions              = crl_ext
default_crl_days  = 365
default_md                  = sha256
name_opt                    = ca_default
cert_opt            = ca_default
default_days                = 3650
preserve            = no
policy                      = policy_match

[ policy_match ]
countryName                  = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress          = optional

[ req ]
default_bits                = 4096
distinguished_name          = req_distinguished_name
string_mask                 = utf8only
default_md                  = sha256
x509_extensions  = v3_ca

[ req_distinguished_name ]
countryName                         = Country Name (2 letter code)
stateOrProvinceName                 = State or Province Name
localityName                        = Locality Name
0.organizationName                  = Organization Name
organizationalUnitName              = Organizational Unit Name
commonName                          = Common Name
emailAddress                        = Email Address

countryName_default                         = EE
stateOrProvinceName_default                 = Harjumaa
localityName_default                        =
0.organizationName_default                  =
organizationalUnitName_default     =
emailAddress_default                        =
[ v3_ca ]
subjectKeyIdentifier         = hash
authorityKeyIdentifier       = keyid:always,issuer
basicConstraints  = critical, CA:true
keyUsage                     = critical, digitalSignature, cRLSign, keyCertSign

[ v3_intermediate_ca ]
subjectKeyIdentifier         = hash
authorityKeyIdentifier       = keyid:always,issuer
basicConstraints  = critical, CA:true, pathlen:0
keyUsage                     = critical, digitalSignature, cRLSign, keyCertSign
crlDistributionPoints        = URI:http://www.minagerges.com/certs/ca-01.crl
authorityInfoAccess          = OCSP;URI:http://www.minagerges.com/certs/ocsp/ca-01/

[ crl_ext ]
authorityKeyIdentifier       = keyid:always
```

Appendix 2.2 - OpenSSL Intermediate Certificate Authority Configuration file

```
# OpenSSL root CA configuration file.
[ ca ]
default_ca          = Intermediate_CA_default

[ Intermediate_CA_default ]
dir                          = d:/certs/intermediate
certs                        = $dir/certs
crl_dir                      = $dir/crl
new_certs_dir                = $dir/newcerts
database            = $dir/index.txt
serial                       = $dir/serial
RANDFILE                     = $dir/private/.rand
private_key                  = $dir/private/intermediate.key
certificate                  = $dir/intermediate.pem
crlnumber                    = $dir/crlnumber
crl                          = $dir/crl/intermediate-crl.pem
crl_extensions               = crl_ext
default_crl_days  = 365
default_md                   = sha256
name_opt                     = intermediate_ca_default
cert_opt            = intermediate_ca_default
default_days                 = 3650
preserve            = no
policy                       = policy_no_match

[ policy_no_match ]
countryName                   = optional
stateOrProvinceName     = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = optional
emailAddress          = optional

[ req ]
default_bits                 = 4096
distinguished_name           = req_distinguished_name
```

```
string_mask             = utf8only
default_md              = sha256
x509_extensions  = v3_ca

[ req_distinguished_name ]
countryName                     = Country Name (2 letter code)
stateOrProvinceName             = State or Province Name
localityName                    = Locality Name
0.organizationName              = Organization Name
organizationalUnitName          = Organizational Unit Name
commonName                      = Common Name
emailAddress                    = Email Address

countryName_default                     = EE
stateOrProvinceName_default             = Harjumaa
localityName_default                    =
0.organizationName_default              =
organizationalUnitName_default  =
emailAddress_default                    =

[ v3_ca ]
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid:always,issuer
basicConstraints  = critical, CA:true
keyUsage                = critical, digitalSignature, cRLSign, keyCertSign

[ server_cert ]
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid,issuer:always
basicConstraints  = CA:FALSE
keyUsage                = critical, digitalSignature, keyEncipherment
extendedKeyUsage        = serverAuth
crlDistributionPoints   = URI:http://www.minagerges.com/certs/intermediate-01.crl
authorityInfoAccess     = OCSP;URI:http://www.minagerges.com/certs/ocsp/intermediate-ca-01/

[ usr_cert ]
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid,issuer
basicConstraints  = CA:FALSE
keyUsage                = critical, nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage        = clientAuth, emailProtection
crlDistributionPoints   = URI:http://www.minagerges.com/certs/intermediate-01.crl
authorityInfoAccess     = OCSP;URI:http://www.minagerges.com/certs/ocsp/intermediate-ca-01/

[ crl_ext ]
authorityKeyIdentifier  = keyid:always
```

Appendix 3 – SEC rules (POC)

Appendix 3.1 – Required Hierarchal rulesets for Windows Events

File: WinEvt_Dispatcher.sec

```
rem=Windown Event Logs Dispatcher

rem= Requires accepting Sysmon.exe, SigCheck.exe and Virus total EULA,
USING SAME ACCOUNT RUNNING SECWIN (SYSTEM ACCOUNT)
rem= Install Sysmon cmd: Sysmon.exe -i -l -n -accepteula
type=jump
ptype=perlfunc
pattern=sub {    my(%hash); my($line) = $_[0]; \
                 if ( $line !~ /Channel="Microsoft-Windows-Sysmon/) {
return 0; } \
                 @array = $line =~ /\[NXLOG@\d+\s(.*)\](.*)/; $input =
$array[0]; \
                 @pairs = split('" ', $input); \
                 foreach my $item(@pairs) { \
                   my ($i,$j)= split('="', $item); \
                   $hash{$i} = $j;} \
                 $hash{"Message"} = $array[1]; \
                 return \%hash;\
             }
varmap=Sysmon_ht
desc=Parse Sysmon events to hashtable
cfset=SYSMON

rem=SecurityAudit
type=jump
ptype=perlfunc3
pattern=sub {    my(%hash); my($line) = $_[0]; \
                 if ( $line !~ /Channel="Security"/ ) { return 0; } \
                 if ( $line !~ /EventID="4663"/ ) { return 0; } \
                 $line  = $_[0].$_[2]; \
                 @array = $line =~ /\[NXLOG@\d+\s(.*)\](.*)/; $input =
$array[0]; \
                 @pairs = split('" ', $input); \
                 foreach my $item(@pairs) { \
                   my ($i,$j)= split('="', $item); \
                   $hash{$i} = $j;} \
                 $hash{"Message"} = $array[1]; \
                 return \%hash;\
             }
varmap=SecurityAudit_ht
desc=Parse SecurityAudit 4663 events to hashtable
cfset=SECURITYAUDIT

type=jump
ptype=perlfunc19
pattern=sub {    my(%hash); my($line) = $_[0]; \
                 if ( $line !~ /Channel="Security"/ ) { return 0; } \
                 if ( $line !~ /EventID="4656"/ ) { return 0; } \
                 $line  = $_[0].$_[18]; \
                 @array = $line =~ /\[NXLOG@\d+\s(.*)\](.*)/; $input =
$array[0]; \
                 @pairs = split('" ', $input); \
                 foreach my $item(@pairs) { \
                  my ($i,$j)= split('="', $item); \
                  $hash{$i} = $j;} \
                $hash{"Message"} = $array[1]; \
                return \%hash;\
            }
varmap=SecurityAudit_ht
desc=Parse SecurityAudit 4656 events to hashtable
cfset=SECURITYAUDIT

rem=Other SecurityAudit
type=jump
ptype=perlfunc
pattern=sub {   my(%hash); my($line) = $_[0]; \
                if ( $line !~ /Channel="Security"/ ) { return 0; } \
                @array = $line =~ /\[NXLOG@\d+\s(.*)\](.*)/; $input =
$array[0]; \
                @pairs = split('" ', $input); \
                foreach my $item(@pairs) { \
                  my ($i,$j)= split('="', $item); \
                  $hash{$i} = $j;} \
                $hash{"Message"} = $array[1]; \
                return \%hash;\
            }
varmap=SecurityAudit_ht
desc=Parse SecurityAudit Other events to hashtable
cfset=SECURITYAUDIT


rem=Application
type=jump
ptype=perlfunc
pattern=sub {   my(%hash); my($line) = $_[0]; \
                if ( $line !~ /Channel="Application"/) { return 0; } \
                @array = $line =~ /\[NXLOG@\d+\s(.*)\](.*)/; $input =
$array[0]; \
                @pairs = split('" ', $input); \
                foreach my $item(@pairs) { \
                  my ($i,$j)= split('="', $item); \
                  $hash{$i} = $j;} \
                $hash{"Message"} = $array[1]; \
                return \%hash;\
            }
varmap=Application_ht
desc=Parse Application events to hashtable
cfset=APPLICATION

rem=Other, mind multiline events and perlfuncN
type=jump
ptype=perlfunc
pattern=sub {   my(%hash); my($line) = $_[0]; \
                if ( $line !~ /Channel=".*"/) { return 0; } \
                @array = $line =~ /\[NXLOG@\d+\s(.*)\](.*)/; $input =
$array[0]; \
                @pairs = split('" ', $input); \
                foreach my $item(@pairs) { \
                  my ($i,$j)= split('="', $item); \
                  $hash{$i} = $j;} \
                $hash{"Message"} = $array[1]; \
                return \%hash;\
            }
```

```
varmap=WinEvtOther_ht
desc=Parse other Win events to hashtable
cfset=WINEVTOTHER
```

File: Sysmon.sec

```
type=Options
procallin=no
joincfset=SYSMON


type=single
ptype=cached
pattern=Sysmon_ht
context=Sysmon_ht:>( sub { return $_[0]->{"EventID"} eq "1" ; } )
desc=Process Created by [$+{User}] image:$+{Image}
parentImage:$+{ParentImage}
action=logonly; \
        create PROCESS_$+{Hashes} 1 ;\
        event
ProcessStarted_Image=[$+{Image}]_Hash=[$+{Hashes}]_Parent=[$+{ParentIm
age}]  $+{_inputsrc}

type=single
ptype=cached
pattern=Sysmon_ht
context=Sysmon_ht:>( sub { return $_[0]->{"EventID"} eq "2" ; } )
desc=Process Changed file creation tTimeStamp image:$+{Image}
parentImage:$+{ParentImage} Hash:[$+{Hashes}]
action=logonly; \
        create PROCESS_$+{Hashes} 1 ;\
        event ProcessChanged-File-
tTimeStamp_Image=[$+{Image}]_Hash=[$+{Hashes}]_FILE=[$+{File}]

type=single
ptype=cached
pattern=Sysmon_ht
context=Sysmon_ht:>( sub { return $_[0]->{"EventID"} eq "3" && $_[0]-
>{"ProcessId"} ne "4" ; } )
desc=NetworkConnectionActivity by [$+{User}] image:$+{Image}
SRCIP=[$+{SourceIp}] DSTIP=[$+{DestinationIp}]
DSTport=[$+{DestinationPort}] DSTportName=[$+{DestinationPortName}]
action=event
NetworkConnectionActivity_Image=[$+{Image}]_IsInitiated=[$+{Initiated}
]_Protocol=[$+{Protocol}]_SRCIP=[$+{SourceIp}]_DSTIP=[$+{DestinationIp
}]_DSTport=[$+{DestinationPort}]_DSTportName=[$+{DestinationPortName}]

type=single
ptype=cached
pattern=Sysmon_ht
context=Sysmon_ht:>( sub { return $_[0]->{"EventID"} eq "4"; } )
desc= <<URGENT>> Sysmon service state changed State:[$+{State}]
action=logonly; create SYSMON_SERVICE_STATE 2; event
SysmonService_StatusChanged_$+{State}

type=single
ptype=cached
pattern=Sysmon_ht
context=Sysmon_ht:>( sub { return $_[0]->{"EventID"} eq "5" ; } )
```

```
desc=Process Terminated image:[$+{Image}] processID:[$+{ProcessId}]
UtcTime:[$+{UtcTime}]
action=logonly;

type=single
ptype=cached
pattern=Sysmon_ht
context=Sysmon_ht:>( sub { return $_[0]->{"EventID"} eq "6" ; } )
desc=Driver loaded ImageLoaded:$+{ImageLoaded} Signed:$+{Signed}
Signature:$+{Signature} HashType$+{HashType} Hash:$+{Hash}
action=logonly; create DriverLoaded_$+{Hash} 1; event
Driver_Loaded_Signed=[$+{Signed}]_ImageLoaded=[$+{ImageLoaded}]_Hash=[
$+{Hash}]

type=single
ptype=cached
pattern=Sysmon_ht
context=Sysmon_ht:>( sub { return $_[0]->{"EventID"} eq "7" ; } )
desc=Image Loaded Image:[$+{Image}] Hash:[$+{Hashes}]
ImageLoaded:[$+{ImageLoaded}] Signed;[$+{Signed}]
action=create PROCESS_$+{Hashes} 1 ;\
        event
ImageLoaded_Image=[$+{Image}]_Hash=[$+{Hashes}]_ImageLoaded=[$+{ImageL
oaded}]_Signed=[$+{Signed}]

type=single
ptype=cached
pattern=Sysmon_ht
context=Sysmon_ht:>( sub { return $_[0]->{"EventID"} eq "8" ; } )
desc=<<Suspeciouse activity>> Created Remote Thread
SourceImage:$+{SourceImage} id:$+{SourceProcessId}
TargetImage:$+{TargetImage} id:$+{TargetProcessId}
ThreadID:$+{NewThreadId} Function:$+{StartFunction}
action=logonly;

type=single
ptype=cached
pattern=Sysmon_ht
context=Sysmon_ht:>( sub { return $_[0]->{"EventID"} eq "9" ; } )
desc=Raw Disk Access Read image:[$+{Image}] PID=[$+{ProcessId}]
Device=[$+{Device}]
action=event
RawAccessRead_Image=[$+{Image}]_PID=[$+{ProcessId}]_Device=[$+{Device}
]

type=single
ptype=cached
pattern=Sysmon_ht
context=Sysmon_ht:>( sub { return $_[0]->{"EventID"} eq "255" ; } )
desc=<<Critical>> Sysmon ERROR id:$+{ID} Description:[$+{Description}]
action=logonly;
```

File: WinEvt_SecurityAudit.sec

```
type=Options
procallin=no
joincfset=SECURITYAUDIT
```

```
rem=Detailed event information
https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event
.aspx?eventID=0000

type=single
ptype=cached
pattern=SecurityAudit_ht
context=SecurityAudit_ht:>( sub { return $_[0]->{"EventID"} eq "1006"
; } )
desc=<<EXTREMELY URGENT>> Malware Detected by Win_Defender
action=logonly; event WinDefenderMalwareDetected

type=single
ptype=cached
pattern=SecurityAudit_ht
context=SecurityAudit_ht:>( sub { return $_[0]->{"EventID"} eq "1008"
; } )
desc=<<EXTREMELY URGENT>> anActionon Malware failed by Win_Defender
action=logonly; event WinDefenderActionFailed

type=single
ptype=cached
pattern=SecurityAudit_ht
context=SecurityAudit_ht:>( sub { return $_[0]->{"EventID"} eq "1102"
; } )
desc=<<EXTREMELY URGENT>> EventLog cleared by User:[$+{SecurityID}]
action=logonly; event WINEVT_AUDIT_LOG_CLEARED_User=[$+{SecurityID}]

type=single
ptype=cached
pattern=SecurityAudit_ht
context=SecurityAudit_ht:>( sub { return $_[0]->{"EventID"} eq "1104"
; } )
desc=<<EXTREMELY URGENT>> EventLog is full
action=logonly; event WINEVT_AUDIT_LOG_Full

type=single
ptype=cached
pattern=SecurityAudit_ht
context=SecurityAudit_ht:>( sub { return $_[0]->{"EventID"} eq "4624"
; } )
desc=An account was successfully logged on
action=logonly; create SuccessfulLogon_$+{LogonID} ; event
SuccessfulLogon_User=[$+{SecurityID}]_LogonID=[$+{LogonID}]_LogonType=
[$+{LogonType}]_ProcessName=[$+{ProcessName}]_SourceIP=[$+{SourceNetwo
rkAddress}]

type=single
ptype=cached
pattern=SecurityAudit_ht
context=SecurityAudit_ht:>( sub { return $_[0]->{"EventID"} eq "4648"
; } )
desc=A logon was attempted using explicit credentials by
User:$+{SecurityID}
action=none; Create EXPLICIT_LOGON_ID_$+{LogonID} ; event
ExplicitLogon_Process=[$+{ProcessName}]_User=[$+{SecurityID}]_Credenti
als=[{$+{AccountDomain}\$+{AccountName}}]_TargetServer=[$+{TargetServe
rName}]

type=single
```

```
ptype=cached
pattern=SecurityAudit_ht
context=SecurityAudit_ht:>( sub { return $_[0]->{"EventID"} eq "4656"
; } )
desc=A handle to an object was requested:
ProcessName:[$+{ProcessName}] ObjectName:[$+{ObjectName}]
User:[$+{SecurityID}] Message:[$+{Message}]
action=logonly;

rem=correlated with 4663 to define object name based on HandleID
>>>>NOT DONE YET
type=single
ptype=cached
pattern=SecurityAudit_ht
context=SecurityAudit_ht:>( sub { return $_[0]->{"EventID"} eq "4660"
; } )
desc=An object was deleted object: ProcessName:[$+{ProcessName}]
ObjectName:[$+{ObjectName}] User:[$+{SecurityID}]
Message:[$+{Message}]
action=logonly; create ObjectDeleted_$+{HandleID} ; event
ObjectDeleted_HandleID=[$+{HandleID}]_ProcessName=[$+{ProcessName}]_Us
er=[$+{SecurityID}]

type=single
ptype=cached
pattern=SecurityAudit_ht
context=SecurityAudit_ht:>( sub { return $_[0]->{"EventID"} eq "4663"
; } )
desc=An attempt was made to access an object:
ProcessName:[$+{ProcessName}] ObjectName:[$+{ObjectName}]
User:[$+{SecurityID}] Message:[$+{Message}]
action=logonly; create ObjectAccess_$+{HandleID}; event
ObjectAccess_HandleID=[$+{HandleID}]_ProcessName=[$+{ProcessName}]_Obj
ectName=[$+{ObjectName}]_User=[$+{SecurityID}]
rem=event
AccessesObject_4463_ProcessName=[$+{ProcessName}]_ObjectName=[$+{Objec
tName}]_User=$+{SecurityID}_Message=[$+{Message}]
continue=takenext

type=single
ptype=cached
pattern=SecurityAudit_ht
context=SecurityAudit_ht:>( sub { return ($_[0]->{"EventID"} eq "4663"
&& $_[0]->{"Message"} =~ /Accesses\:\tWriteData.*AddFile/ ) ; } )
desc= <<WRITEDATA>> An attempt was made to access an object:
ProcessName:[$+{ProcessName}] ObjectName:[$+{ObjectName}]
User:[$+{SecurityID}] Message:[$+{Message}]
action=logonly; event
ObjectAccess_WriteData_HandleID=[$+{HandleID}]_ProcessName=[$+{Process
Name}]_ObjectName=[$+{ObjectName}]_User=[$+{SecurityID}] ;\
        delete SCANNED_$+{ObjectName}

rem=Correlated with 4624:An account was successfully logged on
type=single
ptype=cached
pattern=SecurityAudit_ht
context=SecurityAudit_ht:>( sub { return $_[0]->{"EventID"} eq "4672"
; } )
desc=<<Privileges Escalation>> Special privileges assigned to new
logon: User:[$+{SecurityID}] Message:[$+{Message}]
```

```
action=logonly; create PrivilegesEscalation_$+{LogonID} ; event
PrivilegesEscalation_User=[$+{SecurityID}]_LogonID=[$+{LogonID}]

type=single
ptype=cached
pattern=SecurityAudit_ht
context=SecurityAudit_ht:>( sub { return $_[0]->{"EventID"} eq "4719"
; } )
desc=Audit Policy Changed: Category:$+{Category}
Subcategory:$+{Subcategory} Changes:$+{Changes}
action=logonly;

type=single
ptype=cached
pattern=SecurityAudit_ht
context=SecurityAudit_ht:>( sub { return ($_[0]->{"EventID"} eq "5140"
&& $_[0]->{"EventID"} !~ /\\IPC\$/ ); } )
desc=<<SUSPICIOUSE ACTIVITY>> Network share has been accessed
action=logonly; Create
NetworkShareAccessed_SourceIP_$+{SourceAddress} ; event
NetworkShareAccessed_SourceIP=[$+{SourceAddress}]_User=[$+{SecurityID}
]_Share=[$+{ShareName}]
```

File: WinEvt_Application.sec

```
type=Options
procallin=no
joincfset=APPLICATION

type=single
ptype=cached
pattern=Application_ht
context=Application_ht:>( sub { return $_[0]->{"EventType"} eq "Error"
; } )
desc=WinEvt Application Error EventID:$+{EventID} Source:$+{Source}
Message:$+{Message}
action=logonly;

type=single
ptype=cached
pattern=Application_ht
context=Application_ht:>( sub { return $_[0]->{"EventType"} eq
"Warning" ; } )
desc=WinEvt Application Error EventID:$+{EventID} Source:$+{Source}
Message:$+{Message}
action=logonly;
```

File: WinEvt_Other.sec

```
type=Options
procallin=no
joincfset=WINEVTOTHER

type=single
ptype=cached
pattern=WinEvtOther_ht
```

```
context=WinEvtOther_ht:>( sub { return $_[0]->{"EventType"} eq "Error"
; } )
desc=<<Critical>> WinEvt Error Channel:[$+{Channel}]
EventID:$+{EventID} Source:[$+{Source}] Message:[$+{Message}]
action=logonly;

type=single
ptype=cached
pattern=WinEvtOther_ht
context=WinEvtOther_ht:>( sub { return $_[0]->{"EventType"} eq
"Warning" ; } )
desc=<<Warning>> WinEvt Warning Channel:[$+{Channel}]
EventID:$+{EventID} Source:[$+{Source}] Message:[$+{Message}]
action=logonly;
```

Appendix 3.2 – Detecting malware and ransomware activities

File: Malware.sec

```
rem=process started another instance of itself
type=Single
ptype=RegExp
pattern=ProcessStarted_Image=\[(.*)\]_Hash=\[(.*)\]_Parent=\[(.*)\]
context=PROCESS_SHA1=62952E85C608B68EDE624F1F1C3AB6BB0CD6B092 ||
PROCESS_SHA1=4CB74DC73887907F1EFD67CF6690DD037D88F95D
desc=TRUSTED Process $1 started by $3 [HASH:$2]
rem= TRUSTED processes hashes to avoid dead loop (Perl & SigCheck)
action=none

type=Single
ptype=RegExp
pattern=ProcessStarted_Image=\[(.*)\]_Hash=\[(.*)\]_Parent=\[(.*)\]
context=PROCESS_$2 &&
!PROCESS_SHA1=62952E85C608B68EDE624F1F1C3AB6BB0CD6B092 &&
!PROCESS_SHA1=4CB74DC73887907F1EFD67CF6690DD037D88F95D
desc=Process $1 started by $3 [HASH:$2]
continue=takenext
action=create HASH_$2_IMAGE_$1 60

type=Single
ptype=RegExp
pattern=ProcessStarted_Image=\[(.*)\]_Hash=\[(.*)\]_Parent=\[(.*)\]
context=HASH_$2_IMAGE_$3 && !SCANNED_$2
desc=<<Maliciouse Activity>> Process $3 forked itself as $1 [Hash:$2]
action=logonly; cspawn FileSigChecked sigcheck -q -c -vs "$1"; create
SCANNED_$2 86400

type=jump
ptype=perlfunc2
context=FileSigChecked
pattern=sub {   if ( $_[0] =~ /^.Path,Verified,/) { return 0; } \
                my(%hash); @fields = split(',', $_[0]); @values =
split(',', $_[1]);\
                @hash{@fields} = @values;\
                foreach $key (keys \%hash) { $keyns = $key; $keyns=~s/
/_/g; $hash{$keyns}=$hash{$key}; } \
                return \%hash; }
varmap=SigCheck_VThashtable
desc=Parse SigCheck csv output to hashtable
cfset=sigcheck
```

```
rem=Warn about semi trusted processes
type=single
ptype=RegExp
pattern=ProcessStarted_Image=\[(.*)\]_Hash=\[(.*)\]_Parent=\[(.*)\]
desc=THIS MUST BE A TRUST PROCESS!!!: Process $1 started by $3
[HASH:$2]
rem=must return csv
action=logonly


type=Single
ptype=RegExp
pattern=ImageLoaded_Image=\[(.*)\]_Hash=\[(.*)\]_ImageLoaded=\[(.*)\]_
Signed=false
context=HASH_$2_IMAGE_$3 && !SCANNED_$2
desc=<<Suspicious Activity>> Unsigned library [$3] LoadedBy [$1]
[Hash:$2]
action=logonly; cspawn FileSigChecked sigcheck -q -c -vs "$1"; create
SCANNED_$2 86400

type=Single
ptype=RegExp
pattern=RawAccessRead_Image=\[(.*)\]_PID=\[(.*)\]_Device=\[(.*)\]
context=!SCANNED_$1
desc=<<Suspicious Activity>> RAW DISK ACCESS READ from process [$1]
PID:[$2] Device:[$3]
action=logonly; cspawn FileSigChecked sigcheck -q -c -vs "$1"; create
SCANNED_$1 86400

rem=RansomWare, Change count and time periods
type=SingleWithThreshold
ptype=RegExp
pattern=AccessesObject_WriteData_4463_ProcessName=\[(.*)\]_ObjectName=
\[(.*)\]_User=\[(.*)\]
rem=context=AccessesObject_WriteData
desc=<<Suspected RansomWare>> Many DataWrite by process $1 within very
shorttime
action=logonly; cspawn FileSigChecked sigcheck -q -c -vs "$1"; create
SCANNED_$1 86400
thresh=5
window=3

rem= Port Scan
type=SingleWithThreshold
ptype=RegExp
pattern=NetworkConnectionActivity_Image=\[(.*)\]_IsInitiated=\[false\]
_Protocol=\[(.*)\]_SRCIP=\[(.*)\]_DSTIP=\[(.*)\]_DSTport=\[(.*)\]_DSTp
ortName=\[(.*)\]
desc=Five UnInitiated connection from $3
action=event 5_UNINITIATED_CONNECTIONS_PORT_$5_FROM_$3; create
UNINITIATED_CONNECTIONS_SRC_$3
window=60
thresh=5

type=EventGroup
init=create UNINITIATED_CONNECTION_COUNTING
end=delete UNINITIATED_CONNECTION_COUNTING
ptype=RegExp
pattern=5_UNINITIATED_CONNECTIONS_PORT_(\d+)_FROM_(.*)
```

```
context=!UNINITIATED_CONNECTION_$2_COUNTED
count=alias UNINITIATED_CONNECTION_COUNTING
UNINITIATED_CONNECTION_$2_COUNTED
desc=Repeated UnInitiated connection from $2 for 20 distinct ports
within 1 minute
action=logonly
window=60
thresh=20
```

File: SigCheck.sec

```
type=Options
procallin=no
joincfset=SIGCHECK


type=single
ptype=cached
pattern=SigCheck_VThashtable
context=SigCheck_VThashtable:>( sub { return exists($_[0]-
>{"Verified"}) ; } )
desc=SigCheck result: process image is [$+{Verified}] image:[$+{Path}]
VirusTotal_detection:[$+{VT_detection}] VirusTotal_link:[$+{VT_link}]
continue=takenext
action=logonly


type=single
ptype=cached
pattern=SigCheck_VThashtable
context=SigCheck_VThashtable:>( sub {  @array = $_[0]-
>{"VT_detection"} =~ /(\d+)\|(\d+)/ ; return $array[0] > 0 } )
desc=<<Known malware detected by VirusTotal>> process image is
[$+{Verified}] image:[$+{Path}]
VirusTotal_detection:[$+{VT_detection}] VirusTotal_link:[$+{VT_link}]
action=logonly

rem=rerun Sigcheck to get result after 5 minutes
type=single
ptype=cached
pattern=SigCheck_VThashtable
context=SigCheck_VThashtable:>( sub { return $_[0]->{"Verified"} eq
"Submitted" ; } )
desc=>>Submitted [$+{Verified}] file to VirusTotal image:$+{Path}
VirusTotal_detection:$+{VT_detection} VirusTotal_link:$+{VT_link}
action=logonly;  delete SCANNED_$+{Path}; cspawn FileSigChecked ping
127.0.0.1 -n 300 > nul && sigcheck -q -c -vs "$+{Path}"
```

Appendix 3.3 – System compromise

File: Compromise.sec

```
rem=Monitor Sysmon service for stopped status
type=single
ptype=RegExp
pattern=SysmonService_StatusChanged_Stopped
context=SYSMON_SERVICE_STATE
```

```
desc=<<Suspeciouse incident>> Sysmon Service Stopped, will be started
again
action=logonly; cspawn STARTING_SYSMON net start Sysmon
continue=takenext

type=PairWithWindow
ptype=RegExp
pattern=SysmonService_StatusChanged_Stopped
context=SYSMON_SERVICE_STATE
desc=<<EXTREMELY URGENT>> Sysmon service stopped and failed to start
again within 30 seconds
action=logonly;
ptype2=RegExp
pattern2=The Sysmon service was started successfully
context2=STARTING_SYSMON
desc2=Sysmon service was restarted by correlation engine after it was
stopped
action2=logonly
window=30

rem=NetworkShareAccessed
type=single
ptype=RegExp
pattern=NetworkShareAccessed_SourceIP=\[(.*)\]_User=\[(.*\\localadmin)
\]_Share=\[(.*)\]
context=NetworkShareAccessed_SourceIP_$1
desc=<<VERY Suspeciouse incident>> Network share access by local admin
from IP:$1 user:$2 on share:$3
action=logonly;
continue=takenext

type=single
ptype=RegExp
pattern=NetworkShareAccessed_SourceIP=\[(.*)\]_User=\[(.*)\]_Share=\[(
.*\$)\]
context=NetworkShareAccessed_SourceIP_$1
desc=<<VERY Suspeciouse incident>> Network share access to a hidden
share from IP:$1 user:$2 on share:$3
action=logonly;

rem=Privileges Escalation
type=Pair
ptype=RegExp
pattern=SuccessfulLogon_User=\[(.*)\]_LogonID=\[(.*)\]_LogonType=\[(.*
)\]_ProcessName=\[(.*)\]_SourceIP=\[(.*)\]
context=SuccessfulLogon_$2
desc=Successfull Logon for user $1
action=none
ptype2=RegExp
pattern2=PrivilegesEscalation_User=[$+{SecurityID}]_LogonID=[$+{LogonI
D}]
context2=PrivilegesEscalation_$2
desc2=<<Attention required>> Privileges Escalation by user:[$1]
processName:%4 SourceIP:%5
action2=logonly
window=0

rem=Deleted Object
type=Pair
ptype=RegExp
```

```
pattern=ObjectAccess_HandleID=\[(.*)\]_ProcessName=\[(.*)\]_ObjectName
=\[(.*)\]_User=\[(.*)\]
context=ObjectAccess_$1
desc=ObjectAccess object:[$3] ProcessName:[$2] User:[$4]
continue=takenext
action=none
ptype2=RegExp
pattern2=ObjectDeleted_HandleID=\[(.*)\]_ProcessName=\[(.*)\]_User=\[(
.*)\]
context2=ObjectDeleted_$1
desc2=<<Attention required>> Objected was deleted ObjectName:[%3]
ProcessName:[$2] user[$3]
action2=logonly;
window=0
```

Appendix 4 – Installation and Configuration scripts

Appendix 4.1 – Install NXLog-ce, SECwin

```
'Download NXLog-ce
dim xHttpNXmsi: Set xHttpNXmsi = createobject("Microsoft.XMLHTTP")
dim bStrm: Set bStrm = createobject("Adodb.Stream")
xHttpNXmsi.Open "GET",
"http://nxlog.org/system/files/products/files/1/nxlog-ce-
2.9.1504.msi", False
xHttpNXmsi.Send
with bStrm
    .type = 1
    .open
    .write xHttpNXmsi.responseBody
    .savetofile "c:\windows\temp\nxlog-ce-2.9.1504.msi", 2
end with
'Install NXLog-ce
Set WshShell = WScript.CreateObject("WScript.Shell")
sCmd = "msiexec /qn /i c:\windows\temp\nxlog-ce-2.9.1504.msi"
WshShell.Run sCmd , 1, True
'Deploy NXLog-ce config file
dim xHttpNXcfg: Set xHttpNXcfg = createobject("Microsoft.XMLHTTP")
dim bStrm1: Set bStrm1 = createobject("Adodb.Stream")
xHttpNXcfg.Open "GET", "http://10.31.24.123/nxlog-ce_config.txt",
False
xHttpNXcfg.Send
with bStrm1
    .type = 1
    .open
    .write xHttpNXcfg.responseBody
    .savetofile "C:\Program Files (x86)\nxlog\conf\nxlog.conf", 2
end with
'Enable All Audit Log
sCmd = "auditpol /set /category:* /success:enable /failure:enable"
WshShell.Run sCmd , 1, True
'Start NXLog-ce service
sCmd = "net start nxlog"
WshShell.Run sCmd , 1, True

'Download SECwin
dim xHttp: Set xHttp = createobject("Microsoft.XMLHTTP")
dim bStrm: Set bStrm = createobject("Adodb.Stream")
xHttp.Open "GET",
"https://github.com/minagerges/SECwin/releases/download/1.2.1450/SECwi
n-setup-1.2.1450.msi", False
xHttp.Send
with bStrm
    .type = 1 '//binary
    .open
    .write xHttp.responseBody
    .savetofile "c:\windows\temp\SECwin-setup-1.2.1450.msi", 2
end with
'Install SECwin
Set WshShell = WScript.CreateObject("WScript.Shell")
sCmd = "msiexec /qn /i c:\windows\temp\SECwin-setup-1.2.1450.msi"
WshShell.Run sCmd , 1, True
'Deploy SECwin config
dim xHttp1: Set xHttp1 = createobject("Microsoft.XMLHTTP")
dim bStrm1: Set bStrm1 = createobject("Adodb.Stream")
```

```
xHttp1.Open "GET", "http://10.31.24.123/SECwin_config.txt", False
xHttp1.Send
with bStrm1
    .type = 1
    .open
    .write xHttp1.responseBody
    .savetofile "C:\Program Files\SECwin\Config.reg", 2
end with
'Apply SECwin Config
sCmd = ’regedit /s "C:\Program Files\SECwin\Config.reg"’
WshShell.Run sCmd , 1, True
'Start SECwin service
sCmd = "net start SECwin"
WshShell.Run sCmd , 1, True
```