Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Christian Simoes Isnes

# Generating historical network logs for cyber range exercises

Master's thesis in Information Security
Supervisor: Basel Katt
Co-supervisor: Muhammad Mudassar Yamin
June 2023

NTNU
Norwegian University of
Science and Technology

Christian Simoes Isnes

# Generating historical network logs for cyber range exercises

Master's thesis in Information Security
Supervisor: Basel Katt
Co-supervisor: Muhammad Mudassar Yamin
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

# Generating historical network logs for cyber range exercises

Christian Simoes Isnes

# Abstract

Cyber attacks are increasingly more common, and studies show that threats from Advanced Persistence Threats (APTs), are increasing exponentially. Training cybersecurity professionals in realistic environments is crucial to be prepared for handling a real incident. These training environments can go by the name Cyber Range (CR); whose intention is to provide a safe and isolated environment where cyber security personnel can exercise without risk of affecting any related production systems.

A CR consists of an infrastructure of clients and servers emulating a business or organization in a realistic manner. A part of making the scenario realistic is to have realistic network logs for a blue team to analyze. The research contributions of this thesis include investigating methods and techniques for generating historical network data, as this topic is highly lacking in current literature. This thesis presents an approach to a generator for creating network logs for cyber exercises, with the capability of producing logs spanning over a long period of time, generated in a significantly shorter time.

Following the Design Science Research (DSR) methodology, the thesis defines the requirements for a network log generator. Finally, it presents a developed artifact that successfully generates logs with said functionality. The developed artifact utilizes system clock manipulation and simulates user behavior using browser emulation on the system to generate traffic, all while the generator captures all the traffic on the network interface. The proposed artifact was tested in a closed environment with a number of other clients with its results analyzed and also compared to a baseline of normally generated traffic. The artifact can successfully generate network logs as PCAP files for cyber exercises with customizable start and stop times, different work schedules for traffic distributions, and a speed ranging from 10 to 30 times faster than in real-time.

# Sammendrag

Cyberangrep blir stadig mer vanlige, og studier viser at trusler fra Advanced Persistance Threats (APT'er), øker eksponentielt. Trening av cybersikkerhetsprofesjonelle i realistiske miljøer er avgjørende for å være forberedt på å håndtere en virkelig hendelse. Disse treningsmiljøene kan kalles Cyber Ranger (CR); hvor deres hensikt er å tilby et trygt og isolert miljø der cybersikkerhetspersonell kan øve uten fare for å påvirke relaterte produksjonssystemer.

En CR består av en infrastruktur av klienter og servere som etterligner en virksomhet eller organisasjon. En del av å gjøre scenariet realistisk er å ha realistiske nettverkslogger for cybersikkerts-personellet å analysere under øvelser i Cyber Rangen. Denne prosjektet presenterer en løsning til generering av nettverkslogger for cyberøvelser, med evnen til å produsere logger som spenner over en lang periode, generert på betydelig kortere tid.

Ved å følge forskningsmetodikken Design Science Research (DSR), definerer prosjektet kravene til en nettverkslogggenerator. Til slutt presenterer den en utviklet generator for nettverkslogger som genererer logger med nevnte funksjonalitet. Det utviklede generatoren manipulerer systemklokken til å gå raskere enn normalt og simulerer brukeratferd ved hjelp av nettleseremulering på systemet for å generere trafikk, alt mens generatoren fanger all trafikken på nettverksadapteren.

Den foreslåtte generatoren kan generere nettverkslogger som PCAP-filer for cyberøvelser med tilpassbare start- og stopptider, forskjellige arbeidsplaner for trafikkdistribusjoner, og en hastighet på generering som varierer fra 10 til 30 ganger raskere enn sanntid.

# Preface

This master thesis concludes my Master of Science Degree in Information Security at the Norwegian University of Science and Technology (NTNU) in Gjøvik, Norway.

The Norwegian Cyber Range, NCR, provided the topic. The limited existing research in the area of network traffic generators and being able to provide new research on the topic was a large motivational factor for writing this thesis.


### Acknowledgements

I want to formally thank my two supervisors from the NCR, Muhammad Mudassar Yamin and Basel Katt, for their guidance during the writing of the project. Their knowledge and advice provided during the project helped make this thesis possible. I would also like to thank the other lecturers and staff for their time and knowledge shared throughout their courses over the past five years.

Additionally, I want to direct thank my family for their continuous support and encouragement during my five years at NTNU, through my Bachelor's degree in IT Operations and Information Security and my final Master's degree in Information Security.

Lastly, I want to direct thanks to a dear friend Håvard for all the technical discussions and for pulling me up from all the bottomless rabbit holes I kept digging.


Christian Simoes Isnes
Gjøvik, 01/06/2023

# Contents

# Figures

# Tables

# Code Listings

# Acronyms

**API** Application Programming Interface. 18, 26, 33, 40, 54

**CIM** Common Information Model. 46

**CLI** Command-Line Interface. 7, 9, 18, 32, 53

**CR** Cyber Range. 1–3, 5–7, 10, 17, 38

**DDOS** Distributed Denial-Of-Service. 7

**DNS** Domain Name System. 49

**DS** Design Science. 11

**DSR** Design Science Research. 11, 13, 16, 32, 39, 40, 59

**ELK-stack** Elasticsearch, Logstash and Kibana stack. 6, 29, 38

**GUI** Graphical User Interface. 7

**JSON** Javascript Object Notation. 20

**KVM** Kernel-based Virtual Machine. 40

**NAT** Network Address Translation. 40

**NCR** Norwegian Cyber Range. 1, 2, 5, 6, 9, 14, 17, 19, 21, 24, 27, 38, 56, 60

**NIDS** Network Intrusion Detection System. 7, 8

**NPC** Non-Player/Person Character. 7, 8

**NTP** Metwork Transfer Protocol. xi, 25, 26, 32, 33, 40

**PCAP** Packet-CAPture. 9, 27–29, 38, 41, 42, 45, 46, 56, 59

**PKI** Public Key Infrastructure. 24, 56

**REST** Representational State Transfer. 18, 53

**RTT** Return Trip Time. 60

**SIEM** Security Information and Event Management. 6, 27, 38, 56

**SiLK** System for Internet-Level Knowledge. 9

**TLS** Transport Layer Security. xi, 49, 51, 59

**WLMS** Windows Licence Monitoring Software. xi, 39, 40

**XDR** Extended detection and response. 6

# Chapter 1

# Introduction

In recent years, maintaining a high level of cyber security has become increasingly critical as cyber attacks have become more frequent, sophisticated, and damaging. Cyber attack statistics show that the cost of dealing with a cyber security incident increases yearly, and the threats from advanced persistent threats (APTs) are increasing exponentially [1]. A recent analysis by Google's Threat Analysis Group has shown that Russian government-backed attackers targeting users in NATO countries increased over 300% in 2022 compared to 2020 [2]. Threat campaigns from APTs have also shown a strong focus on critical infrastructure, utilities, and public services, as well as the media and information space.

Training and education for cybersecurity professionals must evolve to keep pace with the changing threat landscape of cyber incidents. Cyber Range (CR)s or testbeds are created to increase security personnel's skills.

The American National Institute of Standards and Technology (NIST) agency has a set of definitions for a CR: [3]:

- A CR is an interactive, simulated platform representing a certain organization's networks, tools, and applications. The CR provides a well-defined safe environment where cyber professionals can gain hands-on skills in a secure environment for product development or security-posture testing.
- The CR consists of tools that may include hardware and software or a combination of virtual and actual components. A well-made CR includes traffic generation, which creates true-to-life network emulation and not static preprogrammed events.

Emulating cyber attacks risk disrupting and damaging production systems if not done in a closed environment. A CR ensures this isolation, and the exercise is not at risk of affecting related systems.

## 1.1 Problem description

In the context of work at NCR for training industrial and high educational partners, they came up with the problem of not having historical network logs for

the clients in a CR exercise. A crucial part of providing the personnel with the best possible scenario is to have artificially generated network logs undistinguishable from real-world everyday web activities, as defined by NIST [3]. Scenarios initiated at the CR need logs that imitate the scenario's infrastructure and contain network traffic over a set period. This timeframe may be historical, meaning it contains events that have happened in the past before the exercise at a CR is taking place.

Without a reliable system for generating these historical logs, a CR needs to orchestrate the infrastructure for the exercise in advance and record logs for the wanted events in real-time. If the scenario benefited from logs spanning two months back, it would require the infrastructure to be initiated two months back in time, running 24/7 and recording network logs with user events which will be part of the exercise.

Setting up the infrastructure that extended time in advance to create the necessary logs is not always doable since a CR may have a short time to prepare the environment for an exercise. The NCR have previously deployed nearly 400 machines in an environment in just 37 minutes [4]. Although to make an infrastructure real, you need to have realistic-looking network traffic as well.

This research aims to investigate methods of generating historical network traffic data and develop a network traffic generator that can generate traffic faster than in real-time.

## 1.2 Scope and limitations

The research in this thesis will focus mainly on generating traffic for network logs. For the generated network logs, only some types of traffic will be prioritized, such as web browsing. It is likely that the system could easily be expanded to integrate more protocols for more realism, for instance, sending emails to introduce SMTP traffic and similar.

## 1.3 Research questions

This thesis is building upon the following research questions:

- **Question 1: What is the current state of the art of historic log generation and their limitation**
  Research into the state of the art of historical log generation will help the audience become more aware of the problem and will be approached using a literature study. Additionally, it will help provide a deeper understanding of the limitations of the current generators, which will later be addressed.
- **Question 2: How these limitations from question 1 be addressed**
  Limitations of the topic of historical log generation will be presented at the end of the literature study. In order to address these, they will be approached using the DSR methodology, which will finally present a working historical

network log generator artifact. The artifact will demonstrate a way of addressing these limitations.

## 1.4 Contribution

This thesis will contribute log generation for CRs and other cyber exercises. We will investigate methods and techniques for generating historical network traffic data and build a working system for generating logs that can be integrated into a CR. A system for generating new, unique, and customizable network logs tailored for a scenario will benefit training environments as they can provide a realistic simulation for security personnel to train on.

## 1.5 Outline

This chapter introduced the problem, while the next chapter will provide background information and the current state of the art of network traffic generators. Chapter three will introduce the method used to solve the discussed problem, and chapters four to six will provide a solution to the problem using the discussed methodology. The discussion will be provided in chapter seven, along with evaluation, and the thesis will be concluded in chapter eight along with future work.

## 1.6 Keywords

Cyber Range, network log generator, historical network logs, cyber exercise

# Chapter 2

# Background and related work

## 2.1 Background

A CR can simulate an entire network and be customized in terms of infrastructure to match a household, small company, or larger enterprise. The CR will imitate a practicing company's current live network or the infrastructure of a newly upgraded network. This CR infrastructure can then be attacked, and personnel can practice defensive mechanisms and forensics in a safe environment. Yamin *et al.* state that an exercise in the cyber range involves different groups of people for preparation and execution. The groups are divided into three parties: The white team, the blue team, and the red team. The white team is responsible for creating and administrating the exercise environment. The red team will try to exploit and attack the exercise infrastructure, while the blue team will attempt to defend and prevent the attacks.

Cloudshare has drawn comparisons between a cyber range and a shooting range, as they both serve the same fundamental purpose [6]:

> "Think about how a shooting range allows marksmen to test their skills in a controlled environment. Every military worldwide has shooting ranges to help impart and refine essential skills. Cyber ranges embrace this same idea – test and hone skills when there's no threat to worry about." [6]

The Norwegian Cyber Range (NCR) project was established in 2018 at the Norwegian University of Science and Technology in Gjøvik, Norway. The establishment of the NCR was a part of the Norwegian government's new strategy for strengthening societal cyber security, and it opened its physical facilities in August 2022 [7]. The NCR is under constant development. According to PhD candidate and lecturer Grethe Østby, the NCR will be used internally at the university to conduct exercises as parts of leadership classes and full-scale exercises and analyses for external organizations [8]. Unlike some other CR's used for training, the NCR also provides facilities to include all roles of an organization during a cyber

incident, not just IT professionals. The board of directors, PR department, CISO, and other functions also handle an incident at the NCR [9]. This way, all involved parties in an organization will be better prepared if a real cyber incident happens.

A security exercise for a CR can be divided into five phases: preparation, dry run, execution, evaluation, and repetition. Exercises have been observed to last from a couple of hours to a couple of days The preparation and dry run may, however, take up to several months [4]. Generating months' worth of historical network data for the Security Information and Event Management (SIEM) solution would significantly increase the time in the preparation and dry-run phase. The purpose of the dry run is to perform final preparations before the execution, including quality assessments of the infrastructure and testing vulnerabilities. Therefore, the dry run phase would be ideal for generating necessary logs for the SIEM before the execution.

Uetz *et al.* value the log data as indispensable for detecting network breaches in a timely manner. The blue team, which detects and prevents attacks on the infrastructure, depends on logs to analyze, often done through a SIEM solution. Exactly how these logs are parsed and analyzed varies between the form of SIEM solution a CR has implemented. The purpose of a SIEM solution is to record, warn, monitor, connect, anticipate, and display the security events and information on network-connected systems [11]. As mentioned by Hasbi *et al.*, a widely used SIEM solution is the Elasticsearch, Logstash and Kibana stack (ELK-stack) and Wazuh, with the latter being an open-source security platform for XDR and SIEM protection. This particular SIEM solution is also integrated at the NCR in Norway.

## 2.2 Related work and State-of-the-art

In this chapter, we review relevant related work and the state of the art on generating network logs. The chapter will describe the currently available tools, their capabilities, and other research related to the thesis field. To our knowledge, no publicly available research on generating network logs with historical timestamps exists. One of the reasons for this might be that performing cyber exercises and running CRs are relatively new concepts, and publishing research papers often take time. To some degree, this can make the state of the art not reflect the current research when this thesis was written. However, This chapter will cover research on different components of building a historical network traffic generator. Firstly it will cover research in the area of traffic generation itself, which is a crucial component of the thesis work, and proceed with other research and techniques, specifically on time manipulation.

### 2.2.1 Traffic generation techniques

Generating the traffic can be divided into three main techniques, manual intervention, automatic generation, and user automation.

**Automatic generation**
As for automatic generation, tools can generate realistic traffic up to 10Gbps. They can generate this enormous amount of data in a short amount of time at the cost of computing power/resources. This type of traffic is often used to test Network Intrusion Detection System (NIDS) for their throughput, and although the traffic can be defined as realistic, it is not based on user behavior [12].

**User automation**
User automation can be achieved using Non-Player/Person Character (NPC)-like systems to emulate users and perform actions on a computer. These NPCs can be scripted to perform certain actions on a target system at specific times relevant to the case at the CR [13].

**Manual intervention**
A difficult task in generating realistic traffic is that no genuine users interact with the computer. Although not viable, a solution is manually interacting with the computers in the CR over a set period to generate the desired traffic manually. The client traffic would then need to be logged into network analysis software to store the traffic for future use. This solution could generate the most realistic traffic of its alternatives since it is generated by humans interacting with computers [12].

However, as previously mentioned, the historical logs may span several months. Daily interaction with all the computers in the infrastructure would require significant human resources and is not liable to generate logs over a long period.

### 2.2.2 Tools for generating benign network traffic

The Cisco TReX generator is Open-source and maintained by Cisco. It is flexible and can be used in stateless and stateful configurations. TRex can scale up to 10-30M packets per second (Mpps) and generate traffic from L3 to L7 [14]. It is Command-Line Interface (CLI)-based, can be interacted with through Python, and has a community-built Graphical User Interface (GUI). This traffic generator is advertised as realistic benign traffic but is not to be confused with benign user-behaviour realistic traffic. Cisco TReX is primarily meant to test the throughput of NIDS', and the traffic would likely look like a Distributed Denial-Of-Service (DDOS) attack due to the data's amount, repeatability, and structure. The packets generated by such a tool would easily be identified as artificially generated traffic by a forensics investigator.

A paper released by researchers from the Institute for Infocomm Research in Singapore introduced a web-traffic generator. Their work presents a web traffic generator based on the Markov Model, Dirichlet distribution, and Hybrid distribution and focuses specifically on generating realistically looking web-browsing traffic. The background traffic used in CR must be comparable and not distinguishable from the random traffic generated from everyday web activities, which is what their research was about. [15].

The GHOSTS Framework was created at the Carnegie Mellon University Software Engineering Institute in Pennsylvania [13]. The tool aims to create high realism in cyber warfare exercises by establishing and building realistic NPCs. The framework was created for generating highly realistic observable network traffic. Researchers can utilize the GHOSTS NPCs to simulate interactions such as browsing websites, downloading files, sending and responding to emails, and much more [13].

Uetz *et al.* did a study for adaptable log generation for testbeds. Their study was motivated by the lack of variety in exercise datasets. The solution was based on user emulation on a client and performing adversary emulation with one step of each tactic in the ATT&CK Matrix for Enterprise from MITRE. Their research is publicly available but does, however, only cover real-time emulation, which happens live during an exercise, and not with the purpose of creating logs of historical context [10]. Their developed artifact SOCBED is open source and publicly available on GitHub and has support for generating benign and malicious network events. SOCBED is different from the other presented works in this chapter as it is a full tool for deploying the infrastructure using VirtualBox and Ansible for generating the traffic on the clients.

### 2.2.3 Tools suited for generating traffic with malicious characteristics

GENESIDS is a traffic generator specializing in creating packets for testing NIDS solutions. To generate application layer payloads, it uses snort rules and reverses these rules into actual network packets, and can only generate "malicious-looking" traffic. It will parse a Snort file with a set of rules and generate traffic that imitates malicious traffic matching the rules signature. For instance, a password cracker, Brutus, uses its name in the HTTP header. A Snort-rule to detect this would look for the ASCII pattern "Mozilla/3.0 (Compatible); Brutus/AET" in the HTTP header/UA. If GENESIDS were to create traffic matching this, it would generate an HTTP packet containing this pattern in the header but lacking other data related to the password crack attempt [16]. This technique is well suited for testing and IDS looking for a particular pattern in its signature. However, the packets generated this way will lack other data in an actual attack and therefore be easily distinguishable.

Gjerstad presented a master thesis on "Generating labeled network datasets of APT with the MITRE CALDERA framework". In a thesis section, the author utilized MITRE CALDERA to generate network logfiles containing malicious traffic, imitating APT29 and emulating observed activity. CALDERA is a framework developed by MITRE that can perform adversary simulation and malicious network attacks on-demand and emulate a particular adversary, for instance, APT29[17]. However, this software produces actual malicious traffic on the network using CALDERA. At the same time, this thesis might only need the characteristics of such traffic to supplement the bening traffic in the final logs.

### 2.2.4 Automatically generating network flow data

It is not given that all network traffic generators or network capture programs can generate flow data. Several of them are primarily generating Packet-CAPture (PCAP) files. Flow data is a more compressed output of network traffic, providing a communication summary. It is more beneficial for initial analysis and narrowing down the investigation as it provides a better overview.

Tools are available to generate the network flows from PCAP. One of the methods is using the SiLK Security Suite from CERT to perform the conversion [18]. The System for Internet-Level Knowledge (SiLK) suite is a set of network analysis tools developed by the CERT NetSA (CERT Network Situational Awareness Team), including tools for analyzing and converting flow data. According to the documentation, a tool in the SiLK suite is called *rwptoflow* and will "Generates SiLK Flow records from packet data". It is a CLI-based tool and should be relatively easy to integrate into another tool to automate the creation of NetFlow logs and the more detailed packet capture files.

Another tool is Suricata. Suricata is an open-source network analysis and threat detection software with high performance that can capture raw network data and create flow data based on packet captures[1]. Since the Norwegian Cyber Range (NCR) already uses Suricata for generating flows in real-time on the clients for the SIEM solution, it is likely that Suricata may be the correct tool to solve the problems related to capturing traffic and converting it to flow data.

### 2.2.5 Generating logs with timestamps set in the past, and faster than real-time

No relevant tools or documented techniques were found in generating traffic/logs that are not happening at the present time. Research for this particular topic or problem was conducted on academic and non-academic sources. Researching of Academic sources was primarily conducted through Google Scholar. Search terms used for research include, but are not limited to, the following:

- Historic network log generation
- Network log generation
- Artificial network logs
- Network logs in the past
- Cyber Range network traffic generator
- Cyber Range network log generation clock
- Cyber exercise historic network logs
- Cyber testbed historic log generation

Research on general traffic generation was found and explored, but their functionality only supported traffic generation in real-time on a client. A solution with functionality for setting a start and end date and generating network logs for that particular timeframe was not found.

[1] https://suricata.io/

According to Vykopal *et al.*, a number of CRs are funded and developed by government and military agencies. There are likely to be classified CRs with many technical details regarded as sensitive, thus leading to the information and research for these cyber ranges being non-public [19]. For this reason, solutions to this problem may exist, although they are researched and developed in closed facilities with their details classified. Another possible reason, as mentioned in the introduction to the chapter, is that publishing research papers may take some time and thus are not publicly available at the time of writing this thesis.

### 2.2.6 Discussion

The tools discovered and covered in this chapter are not dedicated to generating the data needed for a cyber range, which has to be automatically generated and realistic with user-like behavior. The found research is mostly papers that solve a specific problem and do not provide their code/algorithms/final product.

A crucial part of deploying a realistic CR infrastructure is to have realistic traffic in the network, both traffic generated in real-time and historical events in the logs. The limitation in the current state-of-the-art includes that none of the explored works can natively generate any traffic in the past or future, only at the present time. A limitation of a selection of tools is the lack of functionality for capturing the network traffic. For instance, the GHOSTS framework can generate traffic using user simulation but lacks functionality for capturing and saving this network traffic. Some work present tools and techniques with functionality that, in combination, can generate realistically-looking network traffic with both benign and malicious events. However, all the explored works lack functionality for generating traffic with customized timestamps which have happened in the past and seems to be a common limitation on all explored generators.

# Chapter 3

# Methodology

The methodology used for this thesis is Design Science Research (DSR). The method is a problem-solving paradigm seeking to create more knowledge by creating innovative artifacts. Several variations exist of DSR, and this project uses the model presented by Hevner *et al.*, a DSR model for information systems [20]. DSR is used to develop and evaluate new designs, artifacts, and systems, focusing on creating and integrating theoretical and practical knowledge to develop solutions to real-world problems. This chapter will provide theoretical knowledge of the DSR methodology and further explain how the DSR methodology is applied to the thesis. According to Hevner *et al.*, DSR can be fundamentally defined as the following:

> The fundamental principle of design science research is that knowledge and understandingof a design problem and its solution are acquired in the building and application of an artifact [20].

A DSR methodology process consists of the following steps:

1. **Awareness of problem:** Definition of the research problem and justification of the solution's value.
2. **Suggestion** Objectives and requirements to generate a suggested solution to the problem.
3. **Development** An artifact is created based on the requirements and objectives.
4. **Demonstation:** Using the created artifact to solve the problem practically.
5. **Evaluation:** Observing the results of the artifact in use and comparing it to the objective.
6. **Communication:** Will conclude the process and communicate the results and contributions.

Iivari describe the Design Science (DS) activity of building information technology artifacts as important for prescriptive research in Information Systems [21]. Hevner *et al.* have provided some rules for conducting DS research, where the most important of these is that the research conducted with the DSR methodology

must produce an "artifact created to address a problem" [20]. The new developed artifact should draw from existing theories, practices and knowledge in order to provide a solution to a defined problem[20].

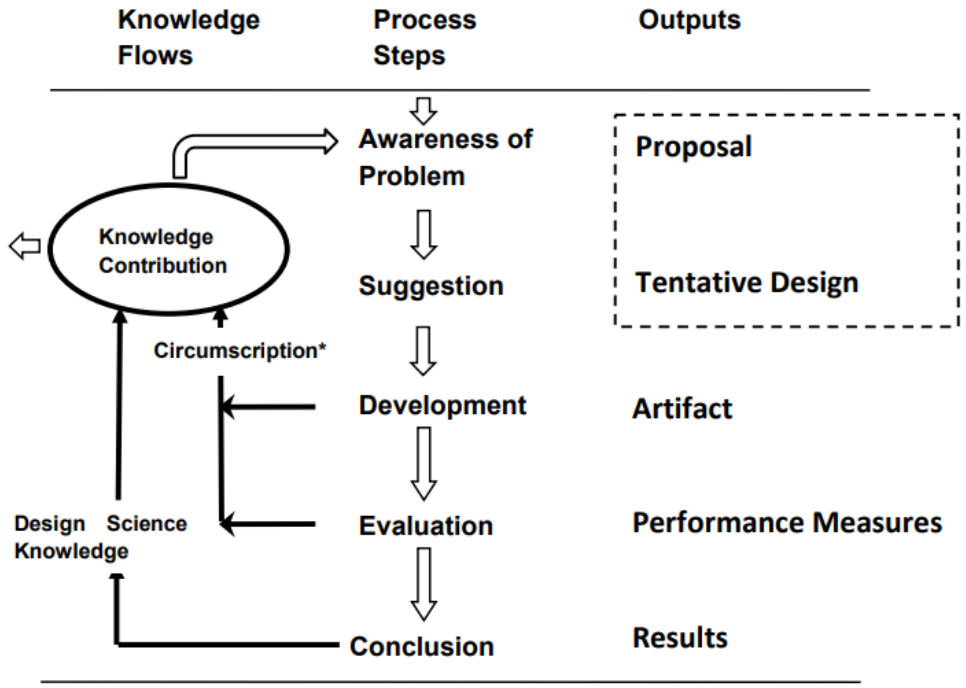



**Figure 3.1:** Design Science Research process model from [20]

The DSR process is followed chronologically, and each activity provides an output used as input in the next stage, as illustrated in Figure 3.1. DSR is a cycle where the result outputted from the development, evaluation, and conclusion phase is used as knowledge for further development of the problem and improved solutions. Development rarely results in a perfect result on the first iteration, and this cycle ensures that the artifact will undergo several iterations for improvement.

## 3.1 Research design in this thesis

The DSR methodology can be tailored to a project based on various needs. Nunamaker Jr *et al.*, Eekels *et al.* and Peffers *et al.* include an extra activity, demonstration, for formally producing results for the evaluation [22–24]. This additional activity, demonstration, has been included in the methodology for this thesis as it is considered an important step in understanding the results presented in the evaluation phase.

### 3.1.1 Activity 1: Awareness of problem

This activity defines the research questions and objectives for the project and justifies the value of a solution that solves the defined problem. This activity requires knowledge about the state of the art and the importance of providing a solution.

A part of becoming more aware of the problem is to study the current state of the art and challenges in the domain of the topic. A literature study in section 2.2 describes the current state of the art for historical traffic generators. The literature study was primarily conducted by reading papers found through Google Scholar using search strings such as "traffic generator", "network log generation", "historical network logs", and more. Studying these papers showed the current state of the art and how network logs are generated for cyber exercises.

Formal papers are often published a while after they are written and may not present the current state of the art, which can provide a false view of the current research state of the topic. No relevant resources were found to cover the specific problem of generating historical network logs, which still indicates the lack of research on the area. The papers provided context for generating network traffic for other purposes. This activity provides knowledge about the problem and outputs a proposal of the problem for the next activity. The activity of becoming aware of the problem was covered in Chapter 1 Introduction, and Chapter 2, Background and Related Work.

### 3.1.2 Activity 2: Suggestion

Within the Suggestion activity, we have split it into two subactivities; definition of requirements and technical design. Design principles start with the deduction of its requirements. Defining requirements does not directly create characteristics of a potential artifact but provides a suggestion for a solution from a generic point

of view. Walls *et al.* explain that without adequate requirements defined, creating an artifact is challenging. The main characteristics of the artifact to be created originate from the task this thesis is based upon. Further requirements for building the historical network generator result from meetings between the author and the supervisors from the NCR. Technical requirements and related use cases are summarized in Chapter 4.1, Requirements.

The technical design process is building upon the requirements created in the previous subactivity, created in cooperation with the NCR. To design and develop a historical network generator, it has to be partly adapted to the current infrastructure in the NCR, while still being generic enough to provide a contribution in the general field of network traffic generation. The current design of the NCRs infrastructure will impact how the generator is designed and developed. The technical design will explore various ways of designing the artifact through continuous testing and finally present a design for a proposed artifact as a tentative design. The main activities in design and development are covered in Chapter 4.2, Technical Design.

### 3.1.3 Activity 3: Development

The development activity creates an artifact based on the tentative design outputted as a suggestion from the last activity. It aims to address the problem and solve the defined objective by providing a working artifact. The output from this activity is an actual artifact that will be used for further demonstration and testing in the next activity. The development activity is covered in Chapter 5, Development and Implementation.

#### Choice of programming language

An artifact has to be developed using a programming language or a combination of several languages. For developing the artifact for this thesis, Python is selected as it is both requested by the NCR and a suitable language to solve the problem. Python itself might not solve all the problems and require supplementation by Bash or PowerShell scripts depending on the underlying operating system. Python is suitable due to the wide range of available packages and the fact that its an interpreted language meaning it executes instructions directly without being precompiled into a binary file. This makes prototyping more effective and facilitates easy code and debugging.

#### Technical specifications

Table 3.1 lists the technical specifications used for the final development. More in-depth specifications are provided in Chapter 5.2, Implementation.

GIT is a vital source control tool to track changes and store the code externally. Source control is kept by GIT integrated into VSCode and through GitKraken, a

**Table 3.1:** Overview of tools and specifications used for artifact development

| Environment | Software | Version |
|---|---|---|
| Programming Language | Python | v3.10.2 |
| IDE/Code Editor | Visual Studio Code | v1.77 |
| Git GUI | GitKraken | v9.2.0 |
| Browser | Chrome | 112.0 |
| Client/dev platform | Windows | 10 |

GUI for administering GIT repositories. This thesis case is located in a private repository on GitHub.

#### Technological scalability

One of the requirements mentioned in Chapter 4.1 is the requirement for the artifact to be developed so that it can be expanded on and scaled in the future. To support this, the development will focus on using classes and modules with a structure such that new modules can be added in the future and seamlessly integrated into the already-designed base system.

### 3.1.4 Activity 4: Demonstration

Demonstrating the artifact is done to see if the artifact solves one or more instances of the problem. The demonstration can be part of a case study, experimentation, simulation, or similar during later stages. The defined problem does not have to be perfectly solved since DSR is a cycle that facilitates future artifact improvements.

The artifact in this project is the historical network generator. The generator should be tested inside an infrastructure to measure performance and reliability. Furthermore, the generated logs should be compared to a baseline of normal traffic in order to measure how well the artifact works.

### 3.1.5 Activity 5: Evaluation

Evaluation will use the results from the demonstration to determine the work's strengths and weaknesses. The results from the demonstration will be observed and measured as to how well the artifact provides a solution to the problem. The primary metric to be evaluated for the artifact in this project is how the generated logs are differentiated from normal logs for a forensics investigator and the generator's performance in terms of speed.

The evaluation will include a direct comparison of functionality between the proposed artifact and a selection of other generators reviewed in Section 2.2, Related works, and State-of-the-art. A selection of the logs is handed to an external analyst for a non-biased evaluation of the generated traffic. This will contribute to the depth of analysis of the generated network traffic.

Additionally, we will evaluate to what degree the technique used to generate the logs fulfills the requirement of generating logs faster than in real time and in a historical timeframe. The evaluation of the artifact will be presented in Chapter *7, Evaluation and Discussion*.

### 3.1.6 Activity 6: Conclusion

The final activity in DSR, conclusion, will communicate the original problem and its importance. It will conclude the process and results, and describe how the project contributes to other researchers and relevant audiences. This step also includes publishing the process and results publicly available for contributing to further research in the area and finally concludes the thesis.

# Chapter 4

# Requirements and technical design

## 4.1 Requirements

This chapter discusses the requirements for building a historical log generator. The requirements are a product of discussions in meetings with the supervisors from NCR, as they have insight into what specific requirements the artifact should be able to meet to work in a cyber exercise/CR.

The general requirements from the meetings are as follows:

- customizable benign traffic
- customizable and simple malicious events
- remote starting of traffic generation on client
- CLI-interface on the client itself to start generator
- traffic from Windows and/or Linux
- configurable timeframe for logs
- realistic distribution of benign traffic
- custom distribution of malicious traffic
- easy to expand with more features
- capture traffic in suitable format
- process captured data to flow data

Most network events will be from benign traffic caused by emulated users using the clients daily. Investigators will analyze the network events generated during an exercise. Therefore, it is important that they are not standing out from other traffic and provide human-like behavior. The benign traffic must also be customizable regarding what websites should be visited or similar, as this can vary between the scenarios.

The malicious network traffic should present some simple activities. In the case of logs generated from Windows clients, an action could mimic an action from

MIMIKATZ [1]: It is a tool commonly used by threat actors to extract information from a system. A specific event to be generated is exfiltrating a file and sending it from a client in the network using PowerShell to a server allegedly belonging to a threat actor.

Since the log generation is to be integrated and automated with the setup of the cyber range infrastructure, it has to support communication with other clients. Providing a REST interface for configuration is desired as it is a standardized way of network communication. These REST routes are for the clients generating traffic and will be used to set the parameters for the log generator. The REST interface has to support and accept configuration alternatives as:

- Start time of logs
- End time of logs
- Schedule (generate 24/7 or 8-16)
- Generator speed

The traffic generator needs functionality for starting it directly on the clients. The possibility of starting the generator on the client also opens up the possibility of using different methods for starting the generator, such as through launch scripts on the client, in addition to manually starting on a client if needed. The CLI interface should support the same arguments as the interface for remote starting.

The traffic generated needs to be from Windows or Linux clients. Since Windows is most used in enterprises, it is prioritized. Large portions of the code can be reused cross-platform, with some OS-specific APIs and compatibility customization.

The network activity has to be generated within a certain timespan with a set start and end date. The start date might be one month ago, and the end date is the day the exercise begins. This timeframe has to be customizable for each scenario and support historical dates.

The distribution of benign traffic may differ for each scenario as it depends on what kind of business is attending the exercise. In the case of a hospital or other business operating 24/7, there might be more activity during nighttime. Other businesses operating on a normal work schedule would have significantly less traffic during the night. This kind of traffic distribution has to be supported and can be achieved through distribution templates (one for 24/7 and one for a normal work schedule)

In certain cases, the timeframe for activity from threat actors can be linked to their 8-16 work schedule in their respective time zone. For instance, malicious traffic from APT28, named Tsar Team by Mandiant and located in Russia, may perform most of their action-on-target activities between the 8 and 16 in Moscow, or purposedly during the night local time [2]. Allowing custom time zones for malicious activity may provide more realism and contribute to threat-actor attribution if that is a part of the exercise.

[1] https://github.com/gentilkiwi/mimikatz

[2] https://www.mandiant.com/resources/insights/apt-groups

Designing the artifact so that it is easy to expand with more features is one of the more important requirements. This ensures that future development on the artifact doesn't require significant restructuring and can be built using the same structure.

Network data should be saved in a format suitable for further processing. The traffic must be captured so that all data is kept and captured on-client to facilitate log-exporting from the client to a SIEM solution. Lastly, the captured data has to be converted to valid network flow data, if not already in such format.

## 4.2 Technical Design

The problem presented in the thesis can be solved in multiple ways. This thesis will attempt to solve it by manipulating the clock of the clients and performing user activities on the client through a script. Another possible technique not explored will be presented in section 8.2, Future work. The artifact's design aims to fulfill and solve the requirements presented in the previous section. The proposed design will primarily achieve two things; provide clients with software which are emulating users to generate true-to-life network traffic with randomness and also a system to create network logs faster than real-time by manipulating the system clock on the clients.

A cyber range consists of several clients making up an infrastructure, and Yamin *et al.* states that the NCR has successfully deployed an infrastructure of 400 clients in the past[4]. It would therefore be beneficial to have a centralized way of controlling the traffic generation of all the clients.

The traffic generator's proposed architecture is split into two parts, a manager and an agent, where the manager controls all agents. An agent is a separate Python script on its own virtual server inside the infrastructure. A manager's only task is to distribute work to the agents.

Beau *et al.* deployed a real-time network traffic generation system at the NCR using a manager and agent for generating traffic [12]. They encountered a challenge with traffic between the managers and clients, which was of significant volume and showed up in the monitored network traffic, which the blue team analyzed. In order to not cause traffic between the manager and agent to interfere with the recorded network logs on the client, it is possible to utilize a network interface. To communicate between the manager and clients, the manager and client can be provided a separate subnet on a separate network interface, e.g., *interface 2*. This way, network traffic captured on *interface 1* will not contain events between the manager and agents as it happens on a different network. Figure 4.1 shows how using two different interfaces, and thereby two different networks can separate manager-client traffic from the traffic captured for generating the network logs.

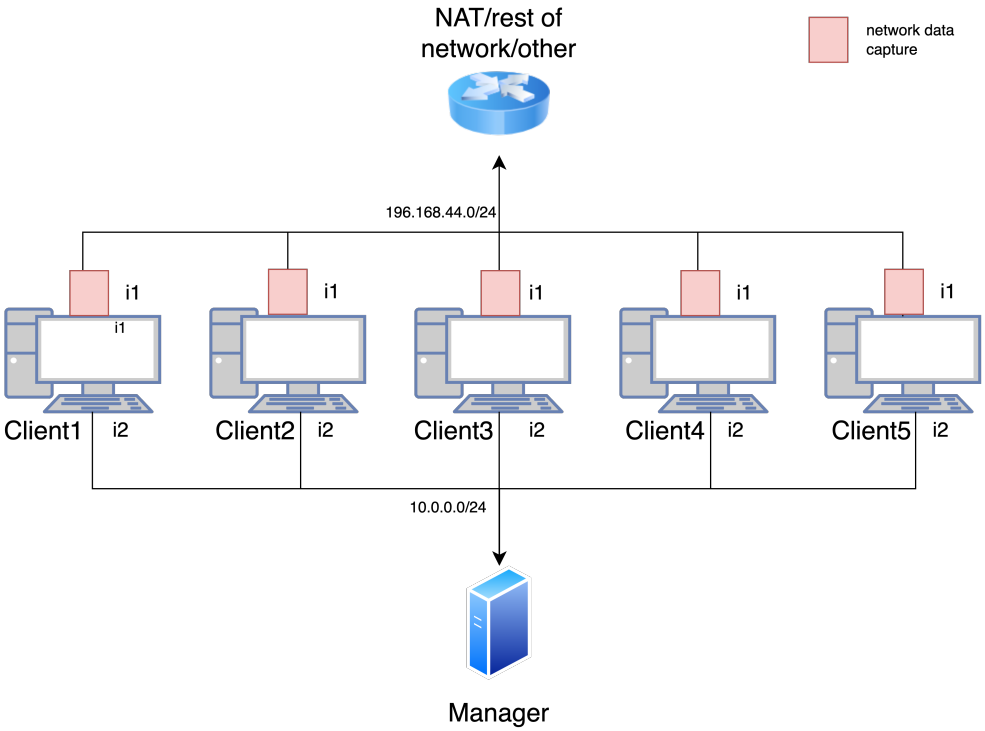



**Figure 4.1:** Illustration of using different interface for manager communication

### 4.2.1 Manager

The manager needs user input to be configured for that particular scenario. There are numerous ways of passing configurations to a script, where one is to use command-line inputs. Since the scripts are made using Python and will be running directly on the system as Python, they can easily be configured using command-line arguments when launching the script. For instance, as "*python3 manager.py -s 01012022 -t 01022022*". The downside to using this is the constraints of the command line prompt, which limits the number of characters. For systems running Windows XP or newer, this limit is 8196 characters which might not be sufficient considering the possible need to pass the IP of each client to the manager when starting [26].

Using a configuration file read by the main script on startup is easier to maintain and more scalable. A JSON file is not limited and can thus provide more data to the Python script and be directly transferred to Python objects.

**Code listing 4.1:** Example structure of config.json read by manager at launch

```
{
  "start":string(dd/mm/yyyy),
  "stop":string(dd/mm/yyyy),
  "schedule":enum("normal","247")
  "agents":[ip]
}
```

Listing 4.1 provides a scalable way of configuring the manager. It can be further expanded to allow more specifications and handle an array of IPs representing the clients with listening agents.

A manager can interact with clients in several ways: one of which is through SSH. The newest builds of Windows 10 and Windows 11 include a pre-installed SSH client based on OpenSSH [3]. Using SSH, the manager would have to SSH into each client with Administrator privileges and start the agent script through commands.

Another way of remotely starting the agents is by providing a REST interface to the agents and starting the agent script on the client through a lunch script from the Hypervisor or schedule. A manager could then POST the start parameters to the agent.

### 4.2.2 Agent

The agent is deployed to all clients from the manager. The agent software is a Python script with four primary tasks, timeline generation, manipulation of the system clock, and generating benign and malicious network events. The secondary tasks for the client are to capture network data and export it to the Wazuh SIEM solution used in the NCR. Figure 4.2 shows a class representation of the agent. The classes responsible for generating traffic are represented in orange, the timeline and clock classes are in blue, and the main agent connecting everything is illustrated in red.

#### Timeline generation

Generation of the timeline is happening in the Scheduler class, whose primary purpose is to generate a timeline of events to execute by the agent.

A timeline contains events the agent will execute and is distributed in a realistic manner regarding regular work hours, as defined in the requirements. A visualized model shown in Figure 4.3 shows a dynamic distribution of event frequency throughout the day, with a peak of events around midday. Having a dynamic distribution of events makes for a realistic traffic pattern with little to no traffic during nighttime, increasing in the morning and decreasing in the afternoon.

Another requirement was to support various work hours for the clients, for instance, simulate a hospital open 24/7. Since these types of workplaces also have employees working at night, there was a need to generate a new timeline to reflect this work pattern.

The timeline of events will be generated using randomization to add diversity to the agent's actions. There are several actions to choose from, each with a set of options for an event. This event option is specific per action and is different between action types. The example in Code Listing 4.2 shows an event for browsing a website and two options.

[3] https://learn.microsoft.com/en-us/windows/terminal/tutorials/ssh

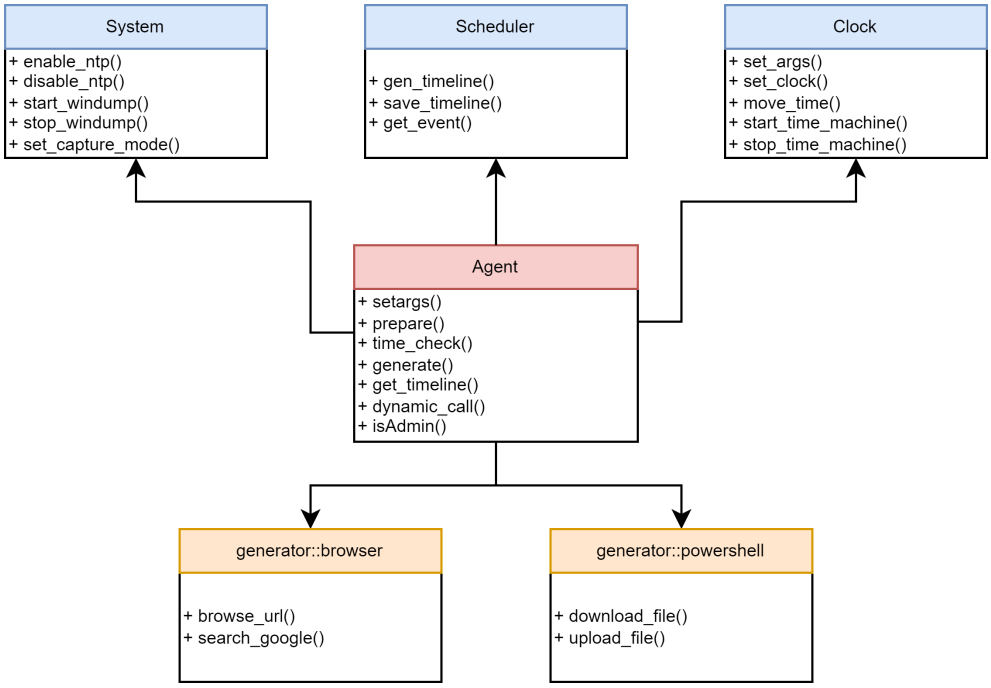



**Figure 4.2:** Class diagram for the client agent

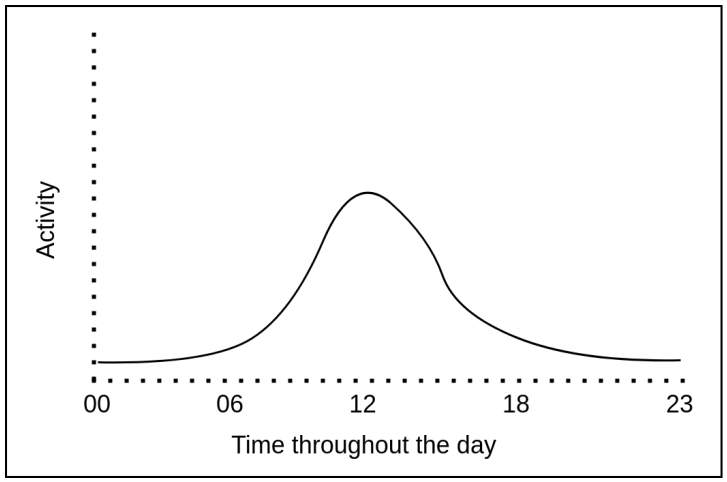



**Figure 4.3:** Client-action distribution on normal work day

**Code listing 4.2:** Example of the event to browse an URL

```
{
  "name": "Browse website",
  "classification": "benign",
  "clock": [],
  "module": "browser",
  "method": "browse_url",
  "done" : false,
  "options": [
    [
      "https://vg.no",
      "https://helseboka.no"
    ]
  ]
}
```

### Generating network traffic

The client's systems will generate background traffic by doing automated operations such as NTP requests, updates checks, ARP traffic, etc. The network data does, however, also need human-behaviour traffic to make it realistic. This network traffic complements the background traffic created by the clients and is not a replacement. As mentioned in Chapter 4.1 about requirements, the modules for generating traffic need to support various options to be customized for different scenarios at the CR.

It is possible to download a website through Python. However, downloading a single HTML file does not provide realistic network traffic. To simulate a real user browsing the internet, Selenium is implemented through Python. Selenium is an open-source project aiming at browser automation and can be integrated with several browsers such as Chrome, Firefox, Edge, and more.

**Code listing 4.3:** Basic usage of Selenium

```
from selenium import webdriver
from webdriver_manager.chrome import ChromeDriverManager

# Create driver for Chrome (can be created for Firefox or other if desired)
driver = webdriver.Chrome(service=Service(
         ChromeDriverManager().install()), options=options)

# Browse to Google
driver.get("https://www.google.no")
```

Incorrect system time can introduce errors related to website certificates upon request. Since this artifact will likely encounter related issues, it must be dealt with. Websites' HTTPS certificates are set with an issue and expiry date. If the client browsing the web server has a system time prior to the issue time or after the expiry time, the browser will throw a **ERR_CERT_DATE_INVALID** error and consider the requested site to be insecure.

As seen in Figure 4.4, the news agency *vg.no* has a certificate that browsers and other clients will invalidate if the time is before January 26th or after April

26th.

```
ubuntu@linux-client-entry:~$ openssl s_client -showcerts -connect vg.no:443 2> /dev/null
notBefore=Jan 26 00:00:00 2023 GMT
notAfter=Apr 26 23:59:59 2023 GMT
```

**Figure 4.4:** OpenSSL posting issue and expiry date of www.vg.no (as of 25th of march

.

Handling errors related to this and forcing the visiting of the website is different between which webdriver Selenium is using. The various options of what flag to launch the driver with are listed in Table 4.1. Passing these flags is merely a workaround, not a solution, as the certificates are still invalid for the browser. The NCR will implement a Public Key Infrastructure (PKI)-service for certificates in the future, and this thesis will not consider how to browse with a valid certificate.

**Table 4.1:** Launch options for browsers to ignore certificate error

| Webdriver | Launch flag |
|---|---|
| Chrome | –ignore-certificate-errors |
| Firefox | accept_untrusted_certs |
| Internet Explorer | acceptSslCerts |

Supporting malicious events is also a requirement. After threat actors have gained a foothold in infrastructure, they will try to accomplish lateral or horizontal movement in a system. These actions often require new tools to be downloaded onto the system. Since Browsers will verify downloaded files and have built-in security measures, it is ineffective at downloading malicious files. However, PowerShell has access to all parts of the host using the .NET framework and is a trusted application that is almost always allowed to execute scripts with impunity [27].

Implementing PowerShell events to upload/download files is, therefore, suitable to match malicious traffic in the generator. PowerShell supports web requests using the *Invoke-Webrequest* method. Using Subprocesses in Python, we are able to run PowerShell commands on the system directly from the main Python script.

### 4.2.3 Controlling time

Controlling time correctly is necessary for two reasons. First, generating logs faster than in real-time requires the generator to serve its purpose in a cyber range. Secondly, it has to be able to generate logs with timestamps set in the past and not the current date and time. The following subsections present various ways of controlling the system's time, which was explored and tested, and which one was more suited for implementation.

### Time-shifting using NTP

The Metwork Transfer Protocol (NTP) handles clock synchronization between computer systems. Therefore, shifting the systems' clocks back in time is suitable for generating logs with correct timestamps.

By default, Windows systems only allow NTP-sync offsets of up to 48 hours for systems running anything newer than Windows Server 2008 [4]. For generating logs months back in time; this is not sufficient. This can be bypassed by changing two registry keys related to the Windows Time Service named *MaxPosPhaseCorrection* and *MaxNegPhaseCorrection*, which allows for overriding this 48-hour limit.


**Figure 4.5:** The two time-offset limit keys in the registry

By changing these values to *0xFFFFFFFF*, the time can be pulled from the NTP-server no matter what the offset compared to the Windows Systems time is.

During testing of time-shifting using NTP for Windows Clients, after five to ten synchronizations of the clock through NTP, Windows failed to adjust the time anymore and reported an error that the time offset from the system clock was too large. This error was reported even though all time offsets were removed through the registry keys in Figure 4.5, and it worked only seconds ago. It is uncertain if the error presented by the Windows Time service was a bug or intended. This behavior, in combination with Windows not supporting NTP Broadcast packets, rendered this solution not optimal for implementation.

### Time skipping

The timeframe of the logs to be generated is significantly larger than the time available for generating them. To solve this, the log generator can skip through time between actions on the clients. Instead of waiting for an activity to happen on a client, it will skip the system clock to the following action and instantly execute them one after the other. Logically it can be presented as "set time to 8:30, browse www.vg.no, set time to 9:30, search on www.google.com, set time to 10:40, search on www.google.com"

This can be achieved in two ways, by controlling the time of the system itself or the time of a specific application. System time can be controlled by utilizing system calls in both Linux and Windows. Controlling the time of a particular application can be used with the library faketime/libfaketime, which allows a user to make the underlying system report a fake time to your application [28]. The library is attached to the program's in-memory image through the library loader and intercepts and handles the system calls related to the system time. This pro-

[4] https://learn.microsoft.com/en-US/troubleshoot/windows-server/identity/configure-w32ime-against-huge-time-offset

gram is available for Linux applications as a Debian package [5]. No program or library with similar functionality has been discovered for Windows.

A drawback of this technique is that it leads to a gap in the logs between events on the client, where there may be several hours during the night with no packets. This is not a realistic-looking scenario since clients automatically send packets between themselves and externally, such as ARP requests, NTP time sync with external servers, etc.

### Manipulating clock speed

Controlling the speed of the clock provides the ability to traverse large timeframes in a short amount of time.

Accelerify is an application that automates this process of speeding up time on the computer[6]. The program was developed by Cylance Inc. in 2013, a software firm later acquired by Blackberry in 2019. The program has a command line interface that accepts two parameters, *-i*, which sets the interval of how often it should skip time, and *-a*, which sets how many seconds it should skip on each iteration. The program's official description was "Accelerate the system clock to observe OS behavior.". It was used, amongst other things, in sandboxes to observe malware activity as the system time progressed quickly.

```
PS C:\Users\Christian Isnes\Desktop> .\accelerify.exe -i 1 -a 30
Accelerify Version 1.0
By Shane D. Shook, Copyright 2011 - All Rights Reserved
Starting time: 10:32
Press any key to exit and restore time. Interval: 0
Press any key to exit and restore time. Interval: 1
Press any key to exit and restore time. Interval: 2
Press any key to exit and restore time. Interval: 3
Press any key to exit and restore time. Interval: 4
Press any key to exit and restore time. Interval: 5
```

**Figure 4.6:** Accelerify test configuration

Figure 4.6 shows a setup for moving time by 30 seconds every 1 second. This practically moves the system clock by 30 days in 24 hours/1 day. In the specific run illustrated, the starting time was 10:32, and the program set to traverse time at 30 seconds every 1 second. The program ended after six intervals, and after running for 6 seconds, the system's time changed to 3 minutes and stopped at 10:35. The technique used by the program is to some degree similar to time skipping, as described above. However, it often skips small intervals instead of skipping intervals between events. This way, there will be no significant gaps in the logs, and the system will generate background traffic as usual.

Accelerify does not have an open-source codebase, but this functionality can be recreated through a Shell/Bash or Python Script accessing the Win32 API to control the system's clock.

[5] https://packages.debian.org/stable/faketime

[6] https://betanews.com/2013/05/07/accelerify-speeds-up-your-pc-clock/

### 4.2.4 Capturing network traffic

The final network traffic should be prepared for importing into a SIEM solution as the WAZUH framework, backed by an ELK stack for log visualization and processing. The NCR already has Suricata installed on clients for capturing data, so it was naturally tested if it would work with this artifact. The artifact requires manipulating the system's time during runtime, which Suricata failed to support when capturing the traffic. When Suricata starts, it reads the system clock into memory and keeps an internal clock inside the program to limit the number of system calls to the clock service. If the system's time changes when Suricata runs, it won't know of the change and will keep timestamping packets using its internal clock as if nothing has happened. An attempt was made to alter Suricatas codebase and build a new binary from a source that supports hardware timestamping for each packet. Still, the extensive size and complexity of the codebase made this not doable with the limited resources. Although Suricata supports real-time capturing of packets, it also accepts offline processing using data from PCAPs [7]. This opens the possibility of using other software to capture the network traffic and pass the PCAP to Suricata for processing. The offline mode can read from a PCAP file, process the traffic, and export flow data, if captured, live.

A PCAP captured by WINDUMP[8] with the modified timestamping engine may be passed to Suricata to make sure events added to Eve.json are correctly timestamped before being pushed to a SIEM solution for analysis. WINDUMP is the Windows version of TCPDUMP[9] and can watch, diagnose, and save network traffic to disk [29]. WINDUMP utilizes WINPCAP[10] as its capture engine and shares a similar default behavior to Suricata when handling timestamping of packets. The library is synced with the computer clock only at the beginning of the packet capture and keeps an internal clock in the program using CPU ticks for tracking progression. As a result, any changes to system time during runtime won't be reflected in the packets. WINPCAP supports different ways of timestamping, which can be modified in the registry key in Code Listing 4.4 [30]. The different allowed modes are presented in Figure 4.7.

**Code listing 4.4:** Reg. key for manipulating WINPCAP timestamp mode

```
HKLM\System\CurrentControlSet\Services\NPF\TimestampMode
```

The registry key adds support for configuring the timestamp mode of packets captured through WINPCAP. The key value suited to solve the problem is mode **2**, which equals **KeQuerySystemTime**. The behavior for timestamping packets using this mode is to query the Windows system call *KeQuerySystemTimePrecise* for each packet and thus get the correct system time and attach it to each packet in the PCAP.

[7] https://suricata.readthedocs.io/en/suricata-6.0.0/command-line-options.html
[8] https://www.winpcap.org/windump/
[9] https://www.tcpdump.org/
[10] https://www.winpcap.org/default.htm

| Registry Key Value | Timestamp Mode | Comment |
|---|---|---|
| 0 | KeQueryPerformanceCounter | Less Reliable |
| 2 | KeQuerySystemTime | More Reliable on Modern Computers |
| 3 | The i386 Read Time-Stamp Counter (RTSC) | Less Reliable, compatibility issues with 64-bit Windows OS |

**Figure 4.7:** Configuration alternatives for TimestampMode [30]

Using WINDUMP for capturing the network data at the network interface level as PCAP files and processing them through Suricatas offline-replay mode achieves the goal of capturing network data with modified timestamps and processing to a format suitable for flow analysis.

### 4.2.5 Design discussion

Figure 4.8 illustrates the proposed architecture. The architecture components are designed based on the various options for each component discussed throughout this chapter. The log generation agent, situated in the middle of the mentioned figure, is responsible for all the client's actions. The proposed design can be summarised as the following:

- **Timeline generation:** Generating the timeline means creating a distribution of events throughout the day. The frequency of events every hour varies based on the time of the day and activity level during working hours. The events need to be selected randomly amongst a set of possible events and have customizable options.
- **Clock manipulation:** The agent will manipulate the clock to generate logs faster than in real-time and with historical timestamps. In order to generate logs faster than in real-time, functionality similar to the Accelerify program can be recreated in Python with access to the Windows API. The module needs support for setting the clock to a specific time and adjusting the perceived speed of the clock too much faster than in real-time to traverse large timeframes quickly. The clock should move fast between the events in the timeline and at normal speed during execution.
- **Generating benign events:** Generating benign events is important to generate realistic data. The agent should be equipped with Selenium for simulating web browsing, mimicking a user's daily online activities. The generator's code should be developed to facilitate easy implementation of new data-generating modules, such as email.
- **Generating malicious events:** The agent should support generating events with malicious characters using PowerShell. PowerShell can be used to both upload and download files, and is often used by threat actors for this purpose, both exfiltrating data and downloading tools to the system.
- **Network capture:** WINDUMP can capture the network data at the network

interface and use the manipulated system clock to timestamp the packet. The PCAP logs can further be processed in Suricata using its offline replay mode to generate structured flow data for indexing in, for instance, Splunk or the ELK-stack.

The proposed design summarised above aims to cover the general requirements for the artifact defined in 4.1. Whether the requirements are successfully fulfilled will be evaluated in Section 7.1.6

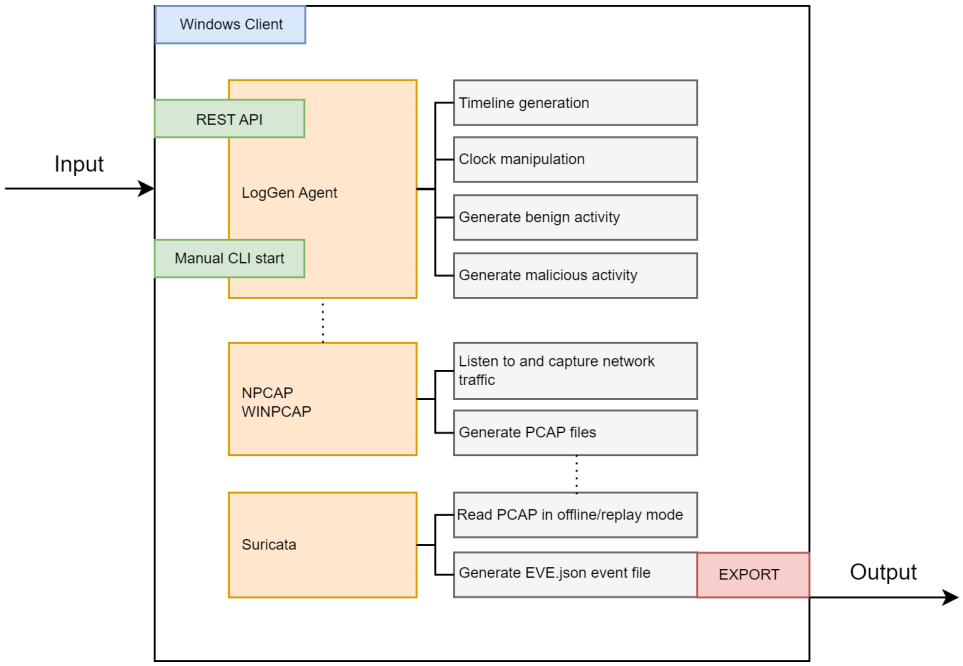



**Figure 4.8:** Summarized architecture with functionality defined

# Chapter 5

# Development and implementation

## 5.1 Development process

The process model used for development was Prototyping. This model involves creating a working design relatively quickly as proof of concept. A big advantage of this model is that it is relatively fast and efficient at ensuring the artifact meets the defined requirements. Since we have no reference point for other historical network log generators, the prototyping model is great at testing various solutions and figuring out which is better for further development.

Other models, such as waterfall, was considered, but it does not work well when the project risk undergoing significant changes during the development phase. Prototyping, on the hand, works excellently for projects like this, where design decisions are made during the development as various techniques are tested, and the vision of the final deliverable is not yet set.

Kanban boards were combined with the prototyping model to keep track of progress. A kanban board is used to manage tasks and was created with the following columns: "To do", "In development", "In testing", and "Done". Having a Kanban board helps facilitate a structured process and easy overview of what stage any functionality is in at any time.

Practically, the prototyping process and kanban boards were used concurrently. Since we had no reference point as to which proposed solution was more suited for implementation due to limited research on the area, all artifact components were individual prototypes. The proposed artifact implemented was the product of these various prototypes, which originates from the ideas in the technical design.

### 5.1.1 Documentation and version control

Version control of the code was kept through a GitHub repository. Code was only committed to the main branch when functioning without errors, while other in-

development changes were kept in a separate experimental branch. This ensures that the main branch always contains working examples of the artifact and can easily be reverted to if needed.

## 5.2 Implementation

Section 4.2 suggested an artifact-design which is used as an input for this next DSR-activity which is development/implementation of the suggested artifact. The implemented solution is developed using Python. Further details on implementing the artifact in production are covered in Chapter 6.1, Deployment.

### 5.2.1 Interaction

The manager or a client can start the agent through the CLI. When started directly on the client, it requires a set of arguments which are defined in Code Listing 5.1

**Code listing 5.1:** Definition of accepted CLI arguments

```
parser.add_argument('--start', type=int, required=True)
parser.add_argument('--stop', type=int, required=True)
parser.add_argument('--schedule', choices=['normal','247'], required=True)
parser.add_argument('--speed', type=int, required=True)
```

Start and Stop define the timeframe of when to generate logs in the format DDMMYYYY. It will generate traffic from 00:00 on the Start-date until the end of the day on the Stop-date. Schedule support either "Normal" or "247" representing the work schedule, further explained in Section 5.2.3. The Speed argument will set the clock multiplier determining how fast the clock should move when not performing an event, for instance, during the night or on the weekend. The current speed argument implementation only accepts 10, 20, or 30.

### 5.2.2 Control the time

#### Prerequisite

NTP has to be disabled on the client's initialization to ensure it won't interfere with the time manipulation. The interval for NTP sync in Windows varies between Windows versions and regions, although it is around two hours, in addition to a sync during system boot.

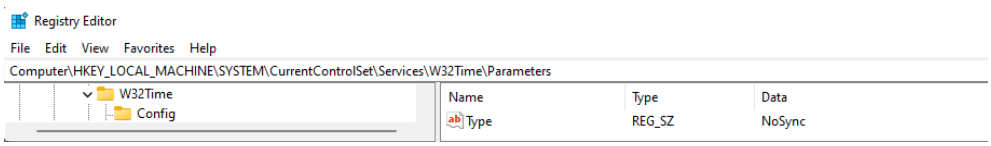



**Figure 5.1:** Registry key to change sync behavior of NTP

Registry keys can be interacted with through Powershell and thus disabled as shown in Code listing 5.2. A similar function exists to enable NTP after the program is finished by changing the *Type* parameters value to "NTP".

**Code listing 5.2:** Example of an event to browse a webpage

```
def disable_ntp(self):
    """
      What:     Disable the NTP sync service on Windows.
      Purpose:  Make sure client does not sync during run.
    """
    try:
      subprocess.run(r'C:\Windows\System32\Windll\v1.0\powershell.exe\
        Set-ItemProperty HKLM:\SYSTEM\CurrentControlSet\services\W32Time\Parameters
        -Name "Type" -Value "NoSync"', shell=True)
    except subprocess.CalledProcessError as e:
      print(e.output)
      raise ValueError("Error occured when disabling NTP")
    else:
      return True
```

### Manipulating clock

The host systems clock is administered through the Windows API using the *win32api* wrapper for Python. Taking inspiration from the Accelerify.exe binary mentioned in Section 4.2.3, its functionality is replicated in Python with more functionality and precision. Figure 4.2 shows the functionality in the Clock class. First and foremost, it can set a completely new system time to whenever the log generation is supposed to start. Further, a system named time-machine is implemented. The functionality of the time machine is to run the system clock faster than in real-time effectively.

Taking inspiration from JavaScripts built-in setInterval method [31], similar functionality was replicated in Python. In order to not cause operations from blocking the main thread in Python, it is implemented in a separate thread of its own. The functionality of the original setInterval method is to call a function at specified intervals. The loop is ended by canceling/terminating its thread. The implemented code is taking inspiration from [32] in order to create a threaded timer, with its full code found in /src/modules/clock.py

The time-machine module primarily supports arguments for how often the time should be skipped (interval) and by how much (shift). The interval decides how often to call the function changing the time of the system, called *moveTime()*. That particular function will, in turn, read the system's current time, add a delta time equal to the shift number, and write the clock back on the system. This functionality will effectively move the time of the system forward by a tiny amount, and when being called on a short interval as five to ten times per second, create the perception that the time of the system is moving significantly faster than in real-time. The time-machine module can be started and stopped at any time to turn the clock back to running at average speed and back to fast speed.

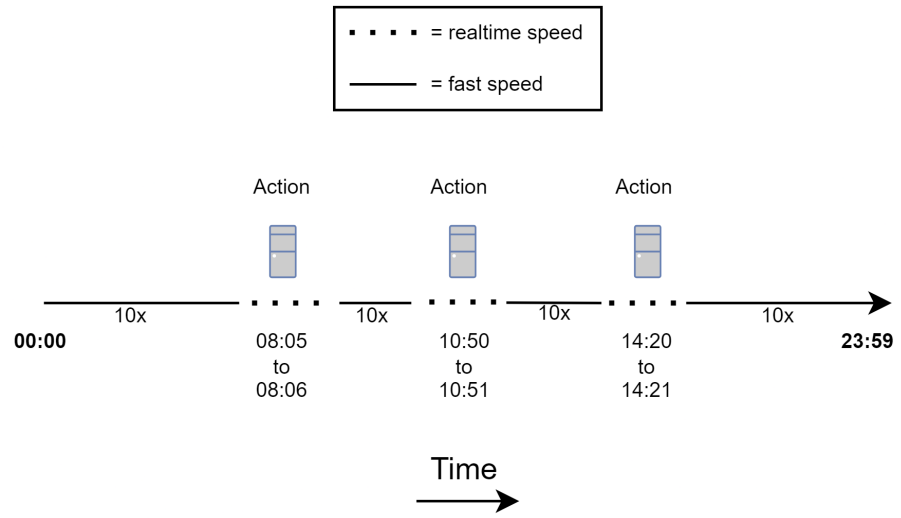



**Figure 5.2:** Visualization of clock speeding and slowing down to real-time during log generation of 3 events.

Figure 5.2 illustrates how the time machine will control the speed between events from the timeline, which will be generated and covered in 5.2.3. The illustrated timeline shows a clock-speed multiplication of 10 between the events. This is just a placeholder and can be adjusted to facilitate the faster or slower generation of the logs.

### 5.2.3 Generating timeline of events

Timeline generation is customizable in terms of work schedules. The timeline is populated with events simulating a real user during the time they are at work. In a normal work environment, it may be from 7-15 during the day, while for hospitals and other facilities open 24/7/365, this schedule will contain more traffic during the night. This is solved by an array data structure with 24 cells, each representing an hour daily. The value of a certain index represents the number of events to be generated at that particular hour. This data structure can be represented in the following bar chart indicating the distribution of events throughout the day for both a normal work schedule and a facility open during the night, though with some decrease in activity.

In addition to generating traffic during the night for facilities with a 24/7 work schedule, it will also generate events for weekends with a 50% reduction in frequency. For a normal work schedule, no events will be generated for weekends. The frequency distribution is illustrated as bar charts in Figure 5.3, with each bar representing the number of events for that particular hour during the day.

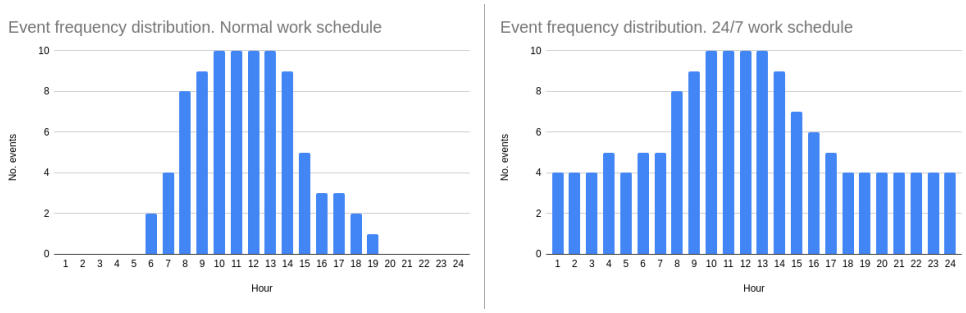



**Figure 5.3:** Bar chart visualizing frequency distribution of events for a normal and 24/7 work schedule

.

### Generating network traffic

The modules for generating network traffic are needed to provide true-to-life activities emulating users on a network.

### Benign: Web browsing with Selenium

Providing realistic network traffic is crucial, so several modular browsing techniques have been created. The first event implemented is a simple event to browse a specific webpage directly and close the browser afterward.

The second event is slightly more complex, which is to make a search on Google. The event contains a set of options for what to search on Google. The function implemented will open a Chrome instance, search for the specified query and view the results. Once Google provides the results, the script will make a decision to view the next page of results or stay on the first page. Afterward, it has an 80% chance of clicking a random result and browsing that page. In 20% of cases, it won't find the desired result and close the browser. Between the set of actions, random delays are added to introduce randomness in the browsing and make the behavior seem more like a human than a script. This approach is logically visualized in Figure 5.4.

**Code listing 5.3:** Event for browsing Google

```
{
  "name": "Search on google",
  "time": "1675508400",
  "module": "browser",
  "type": "search_google",
  "options": [
    "how to make a sandwich",
    "how to make a cake",
    "how to make a pizza"
  ]
}
```

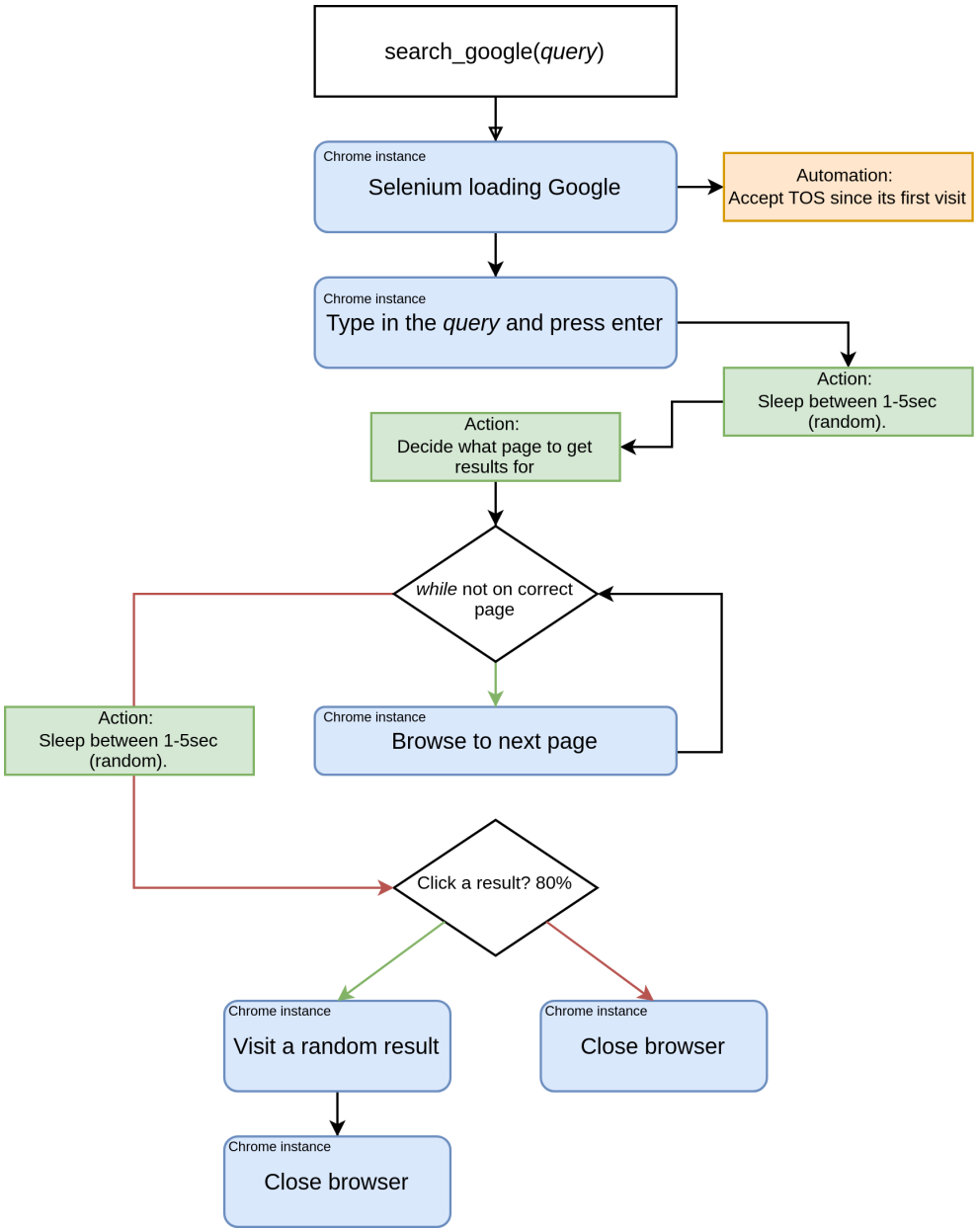



**Figure 5.4:** Logical flow diagram for making a Google search with Selenium.

Chrome's webdriver used by Selenium supports launching with several arguments that ignore various SSL certificate errors. The arguments used to enable browsing through certificate errors are seen in Code Listing 5.4

**Code listing 5.4:** Disable certificate checks in Seleniums Chrome-options

```
options = webdriver.ChromeOptions()
options.add_argument("--ignore-certificate-errors")
options.add_argument("--disable-extensions")
options.add_argument("--ignore-ssl-errors")
options.add_argument("--ignore-certificate-errors-spki-list")
```

After the class responsible for browsing websites is initiated, the illustrated options are added to the launch arguments.

### Malicious: Make web requests using PowerShell

A threat actor with a foothold in a system can, in many cases, have access to the local PowerShell framework. Specifically, two functions have been implemented to emulate malicious traffic from a threat actor inside a client. The first functionality is to download a specific file from a URL using PowerShell illustrated in Code listing 5.5

The raw event for a malicious event is seen in Code Listing 5.6

**Code listing 5.5:** Download a file using Powershell from URL

```
try:
  subprocess.run(f'C:\Windows\System32\powershell.exe Invoke-Webrequest "{url}"',
  shell=True)
except subprocess.CalledProcessError as e:
  print(e.output)
  raise ValueError("Error when downloading file")
else:
   return True
```

**Code listing 5.6:** Malicious base event

```
{
    "name": "Download file",
    "classification": "malicious",
    "clock": "",
    "module": "powershell",
    "method": "download_file",
    "done" : false,
    "options": [
        [
            "https://webhook.site/c926c08d-0882-478e-8170-06740db5a159/privesc.sh",
            "https://webhook.site/c926c08d-0882-478e-8170-06740db5a159/winenum.sh"
        ]
    ]
}
```

The URLs available for this event are specifically to download files with malicious names indicating post-exploitation frameworks or tools to elevate privileges. The other event using Powershell is highly similar, but with the functionality of POST a file from the client to a web server, also using PowerShell.

### 5.2.4 Capture and process network data

Cyber Range (CR)'s often have integrated SIEM solutions for log analysis, whereas Norwegian Cyber Range (NCR) is currently running the WAZUH framework backed by an ELK-stack. In order to capture all the data to and from the client-generating traffic, the generator will utilize WINDUMP for listening to and capturing all data passing through the network interface. Code Listing 5.7 shows snippets of the commands for achieving starting packet capturing on the network interface. An important prerequisite is to set the TimestampMode of the WinPcap library to "2" to make it timestamp each packet using the actual system time. The implementation of WINDUMP for capturing data on the network interface will save the raw data as a PCAP file in the project's directory once the generator is finished with its timeline.

**Code listing 5.7:** Snippets from the code for activating WINDUMP on the network interface

```
#Set timestamping mode
subprocess.run(r'powershell.exe Set-ItemProperty
                HKLM:\SYSTEM\CurrentControlSet\services\NPF
                -Name "TimestampMode" -Value 2', shell=True,
                check=True, stderr=subprocess.PIPE)

# Start capturing service
wdump_path = os.path.dirname(full_path) + '\\windump.exe'
self.capture = subprocess.Popen([wdump_path, '-i','1','-w','captured.pcap'],
                   stdout=subprocess.PIPE,
                   stderr=subprocess.PIPE)
```

### 5.2.5 Implementation discussion

The final artifact created successfully generates network logs faster than in real-time and within a customizable timeframe. The agent creates a timeline of events to perform within the provided timeframe and proceeds with setting the system time to the start date and running quickly until the first event in the timeline. Once an event in the timeline is hit, the system clock returns to real-time speed and does the action on the client, such as browsing a website. Once the action mimicking a user is completed, the clock will proceed quickly until the next event. This cycle goes until all events in the timeline are executed and the end of the last day has been reached. Meanwhile, all network flows are captured on the interface and saved as a PCAP file.

Automatic conversion from PCAP files to flow data using Suricata is not implemented as a part of the artifact as it serves a different purpose than traffic generation. Additionally, different Cyber Range (CR)s may want different formatting of their logs before importing in a SIEM solution.

# Chapter 6

# Deployment and demonstration

This chapter will cover limitations for deployment and prerequisites for the log generator to work properly on the deployed system. The fourth activity of our implemented DSR methodology is to demonstrate and test the artifact provided by the development phase. This chapter will cover details regarding the testing of the historical traffic generator. The results from the testing will be evaluated in the next chapter.

## 6.1 Deployment requirements and limitations

The created artifact has some requirements, which also can be perceived as limitations. The target system must meet the following conditions for the traffic generator to work properly.

- Windows operating system.
- WINDUMP requires WinPCAP to be installed.
- Administrator privileges
- No external clock synchronization

**Windows** is the operating system for which the artifact is developed and tested. Testing was primarily performed on Windows 10 22H2 Pro. A few test runs were also done on Windows 11 22H2 Pro, and no issues occurred. There is no indication that the techniques utilized won't work on other Windows versions either. The proposed solution for generating historical logs, and manipulating the system clock, introduces some problems that only occur in Windows's Enterprise version. The problem encountered in testing the artifact was regarding the License-handling of Enterprise-systems. The Enterprise edition of Windows has a more strict Windows Licence Monitoring Software (WLMS), which will issue a shutdown command to the system when the license period for Windows Enterprise has expired.

The event logged as ID1074, *System shutdown by user or process*, occurred shortly after setting the systems clock to the start date when attempting to generate logs and is seen in Figure 6.1. This action does not make the system clock

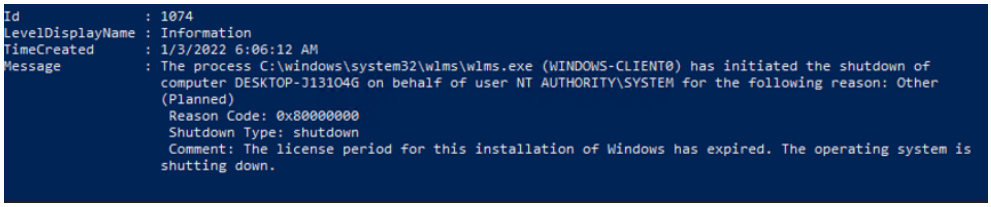



**Figure 6.1:** Event from WLMS initiating a shutdown due to expired license.

pass the original expiry date of the system. Still, a theory is that Windows is interpreting it as an attempt to circumvent its evaluation period by moving the clock back in time and is therefore issuing a shutdown command.

**WINDUMP**, the Windows version of tcpdump, is responsible for capturing all data on the network interface. In order to properly function, it relies on libraries supplied by WinPcap, which has ceased development. Both programs are free under the BSD license and function using the latest supplied version. WinPcap v4.1.3 and v3.9.5. WinPcap has to be installed on the system for WINDUMP to work. A reboot is not required after installing WinPcap.

**Adminsitrator** privileges are required for accessing the Windows API calls used for controlling the clock on the system. The script will check for these permissions on start and terminate if the condition is unmet.

**External clock synchronization** outside of NTP has to be disabled, specifically synchronization from hypervisors to the guest operating system. The scripts provided with this thesis disable NTP synchronizing before starting. This ensures the system won't issue a periodic time-synchronization request and will reset the clock while the script generates the network logs. Sometimes, hypervisors synchronize the clock with the guest operating system outside of using NTP. VMWare can be configured with a periodic time synchronization which will synchronize the guest operating systems clock with the host's through VMware-tools [33]. Kernel-based Virtual Machine (KVM) as a hypervisor also offers this functionality through *kvm-clock*. Performing external time-synchronization while the script runs will compromise the generated logs, thus the requirement to disable this type of external clock-synchronization. When the artifact was deployed on NTNU Skyhigh, which uses KVM as its virtualization engine, the system clock on the guest operating was periodically synced to its host, even though NTP and Windows Periodic Time synchronization were disabled. On VMWare hypervisors, this functionality is disabled by default and does not interfere with the traffic generator.

## 6.2 Demonstration and testing

The fourth activity in the DSR methodology is to demonstrate the artifact provided by activity four, the development phase. Figure 6.2 illustrates the topology used for testing. The network comprises five guest Windows 10 clients on an internal network with internet access using Network Address Translation (NAT) through

the host.

### 6.2.1 Test environment

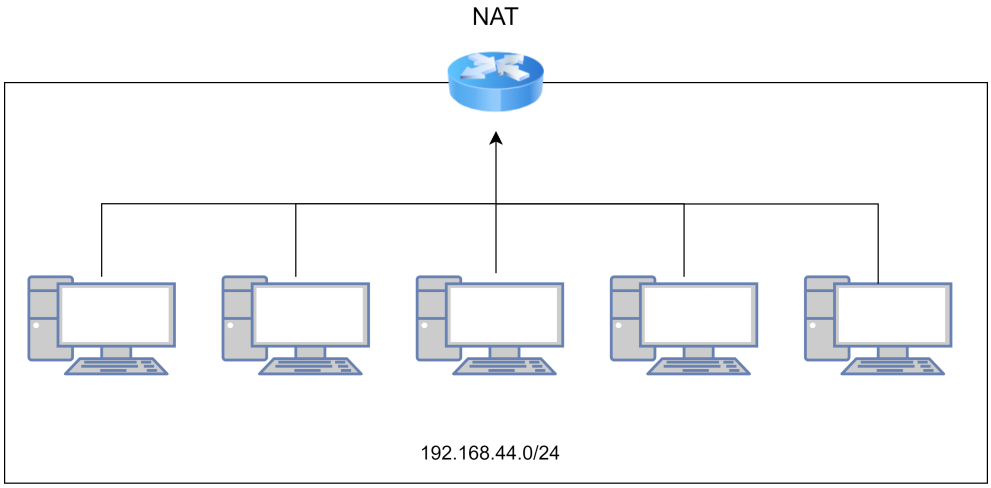



**Figure 6.2:** Topology of the network used for testing

The client generating said logs for testing has internet access and is in a network with 4 other Windows Clients. The other clients on the network will provide background traffic from broadcast packets and other client-to-client requests occurring automatically in the background. All clients are virtual, running with 2 virtual cores and 4GB of memory. Specifications of the clients in the infrastructure are listed in Table 6.1.

**Table 6.1:** Test environment specifications

| Category | Product | Version |
|---|---|---|
| Hypervisor | VMWare | Workstation 17 Pro |
| Operating System | Windows | 10 Pro 2H22 |
| Software | Python | v3.11.3 |
| Software | WINPCAP | v4.1.3 |
| Software | WinDump | 3.9.5 |
| Software | Chrome | v113.0 |
| Guest HW | 2 vCore and 4GB RAM | |

Apart from the client recording logs, the other four clients in the network were idle during all test scenarios. WINDUMP which is used for capturing network traffic, does by default limit the capture length of packets to reduce the size of the PCAP files. This is handled by the *snaplen* flag when initiating the WINDUMP capture process, which limits the daily PCAP size to about 120 MB. This value is

set to default for all test runs except for Test 3, which will run with *snaplen* set to 0 which will capture absolutely all data. This is sufficient for extracting the extra data available for analysis and evaluation.

### 6.2.2 Generating data

Changing the clock speed changes the time it takes to generate the logs. When the speed multiplier is 1x, the system runs in real-time, while at 10x, it runs at 10 times the normal speed between events. Results are generated in the mentioned infrastructure using the parameters presented in Table 6.2. The results of the logs generated are presented in Section 7.1.2

**Table 6.2:** Log generation test parameters

| Nr: | Start | Stop | Schedule | Speed |
|---|---|---|---|---|
| 1 | 03/01/2022 | 03/01/2022 | Normal | 1x |
| 2 | 03/01/2022 | 03/01/2022 | Normal | 10x |
| 3 | 03/01/2022 | 03/01/2022 | Normal | 20x |
| 4 | 03/01/2022 | 09/01/2022 | Normal | 20x |
| 5 | 03/01/2022 | 09/01/2022 | 247 | 20x |
| 6 | 03/01/2022 | 03/01/2022 | Normal | 20x (full packets) |

Based on a normal schedule, the timelines were generated with 86 user events to perform per day on the weekdays and zero events during the weekends. Since the historical network logs are only to be generated on one client in the test scenarios, the agent is started directly on the clients without remote activation. The clients running on the 247 schedule, which generates some events during the night and weekends, produce 143 events per day hours, regardless of the day of the week.

Figure 6.3 illustrate how the network traffic generator is started directly on the client through the CLI. The primary python script *agent.py* is called with the required flags –start, –stop, –schedule and –speed. Once called, the script will initiate all required modules and ensure they're correctly started before generating traffic based on the timeline.

At the end of each log generation, a PCAP is saved by the generator containing all the log data collected on the network interface in that period of time.

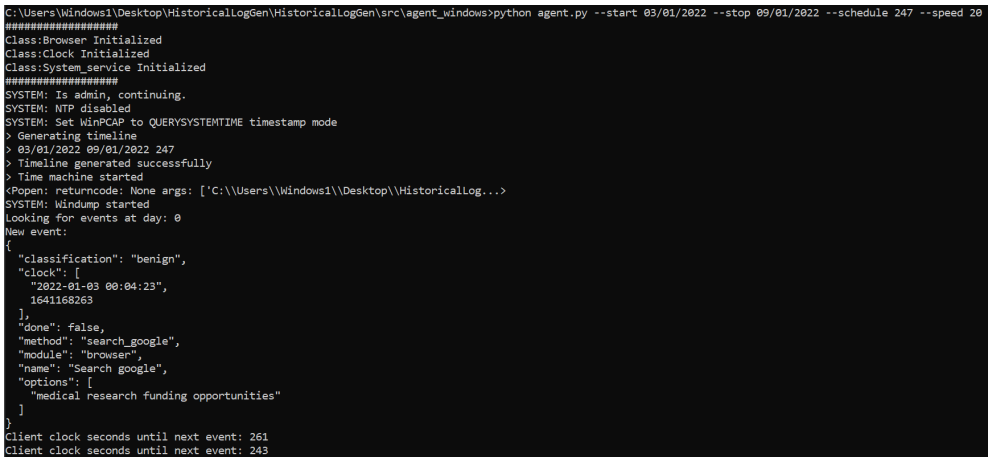



**Figure 6.3:** CLI input for starting Test5

# Chapter 7

# Evaluation and discussion

This chapter evaluates the artifact's results and discusses the limitations of the current version.

## 7.1 Evaluation

The fifth activity in the DSR methodology is evaluating the artifact. The evaluation-activity is utilizing the outputs from the demonstration and aims to assess how well the artifact solved the problem and answered the research questions. The primary problem and challenge in this project were to create a method to generate logs both faster than in real-time and also with timestamps set back in time. Both of these objectives had to be achieved while still providing network logs with a realistic structure replicating the traffic of a normal user. The artifact results are also assessed in regard to the requirements defined in section 4.1, which are used as guidelines for developing the historical network traffic generator.

In order to provide a baseline of what "normal" traffic would look like in that particular infrastructure, normal logs were generated. The historical network traffic generator was set to record the network traffic for 24hrs in the infrastructure, while the agent generated user events as normal. The speed was set to 1x, meaning the clock was running at normal speed at the present time. This will help with, amongst other things, determining changes in flow integrity and structure of the logs generated historically compared to the logs captured in real-time. A 24hr network sample will provide a baseline for traffic during the daytime but also during the night.

### 7.1.1 Analysis platform

The raw generated logs are files in a PCAP format that may grow gigabytes large for longer timeframes and be challenging to analyze alone. In order to process and evaluate the logs in a more structured manner, it is analyzed in Splunk. Splunk makes it possible to search, analyze and visualize data in a structured manner. To

make data available for analysis in Splunk, it has to be in a format Splunk is able to index.

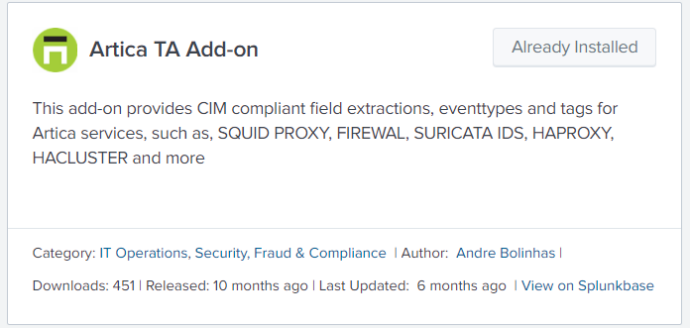



**Figure 7.1:** Artica TA Add-on for Splunk

Splunk supports various add-ons, including one for Suricata named "Artica TA Add-on", as shown in Figure 7.1. This add-on will provide Common Information Model (CIM) compliant field extracts for services, including Suricata, which add support for Suricata's exported eve.json file. In order to generate flow data from a PCAP file using Suricata, it was run using its offline replay mode [1]. Figure 7.2 shows how the file *03-09.pcap* was converted to flows and outputted to *eve.json*. After the data was converted, it was imported to an index in Splunk using the Artica schematic and made searchable.

```
C:\Users\Christian Isnes\Desktop\dataset>"C:\Program Files\Suricata\suricata.exe" -c suricata.yaml -r 03-09.pcap
26/5/2023 -- 23:06:44 - <Info> - Running as service: no
26/5/2023 -- 23:06:44 - <Notice> - This is Suricata version 6.0.9 RELEASE running in USER mode
26/5/2023 -- 23:06:44 - <Warning> - [ERRCODE: SC_ERR_CONF_YAML_ERROR(242)] - App-Layer protocol ikev2 enable status not set, so enab
ling by default. This behavior will change in Suricata 7, so please update your config. See ticket #4744 for more details.
26/5/2023 -- 23:06:44 - <Warning> - [ERRCODE: SC_ERR_INVALID_ARGUMENT(13)] - No output module named eve-log.ike
26/5/2023 -- 23:06:44 - <Warning> - [ERRCODE: SC_ERR_INVALID_ARGUMENT(13)] - No output module named eve-log.quic
26/5/2023 -- 23:06:44 - <Error> - [ERRCODE: SC_ERR_STATS_LOG_GENERIC(278)] - eve.stats: stats are disabled globally: set stats.enabl
ed to true. See https://suricata.readthedocs.io/en/suricata-6.0.9/configuration/suricata-yaml.html#stats
26/5/2023 -- 23:06:44 - <Notice> - all 17 packet processing threads, 2 management threads initialized, engine started.
26/5/2023 -- 23:06:58 - <Error> - [ERRCODE: SC_ERR_PCAP_DISPATCH(20)] - error code -1 invalid packet capture length 273724797, bigge
r than snaplen of 96 for 03-09.pcap
26/5/2023 -- 23:06:58 - <Notice> - Signal Received.  Stopping engine.
26/5/2023 -- 23:06:58 - <Notice> - Pcap-file module read 0 files, 6212456 packets, 557950059 bytes

C:\Users\Christian Isnes\Desktop\dataset>
```

**Figure 7.2:** Converting PCAP files to flow data using Suricata's replay function.

### 7.1.2 Performance of the generator

Part of the research task was to generate valid network logs faster than in real time. The proposed artifact solves this by running the clock speed significantly faster than normal. The time to generate the logs is presented in Table 7.1, with the Nr. column representing the respective set of parameters from Table 6.2.

From the point of development and configuration, there is no known limitation on how high the speed multiplier can be set. Increasing the multiplier will likely keep reducing the volume of recorded background traffic as it is captured over a shorter time (real-time), and increase the flow duration for the packets captured.

[1] https://docs.suricata.io/en/latest/command-line-options.html

**Table 7.1:** Statistics of generated logs

| Nr: | Total Packets | Flow events | Time to generate | PCAP-size |
|---|---|---|---|---|
| 1 | 1 769 007 | 22 764 | 24hr 0min | 181MB |
| 2 | 964 155 | 20 916 | 2hr 52min | 97MB |
| 3 | 1 213 491 | 20 504 | 1hr 27min | 124MB |
| 4 | 6 212 456 | 109 703 | 10hr 15min | 923MB |
| 5 | 9 171 135 | 199 749 | 11hr 37min | 971MB |
| 6 | 1 255 002 | 28 602 | 1hr 24min | 1.4GB |

### 7.1.3 Analysis of generated logs

Table 6.2 provide statistics for the time needed to generate logs for a set period of time. The statistics show the provided artifact can generate network logs significantly faster than in real-time.

Figure 7.3 and 7.4 show the event distribution through a day of captured network. The volume of traffic for each hour results from the generated timeline based on the schedule "normal" which is a curve relatively similar to the bell curve to cause an increase in traffic in the morning and a decrease in the afternoon. Both distributions show a relatively similar pattern, with the baseline traffic having 11% more events than the logs generated at 20x speed.

Given that Test 3 is generating traffic in nearly 1/20th the time of the baseline traffic, the variance in the number of events is acceptable. It could also result from other external factors, such as unreliable background traffic from other clients in the network during log generation.

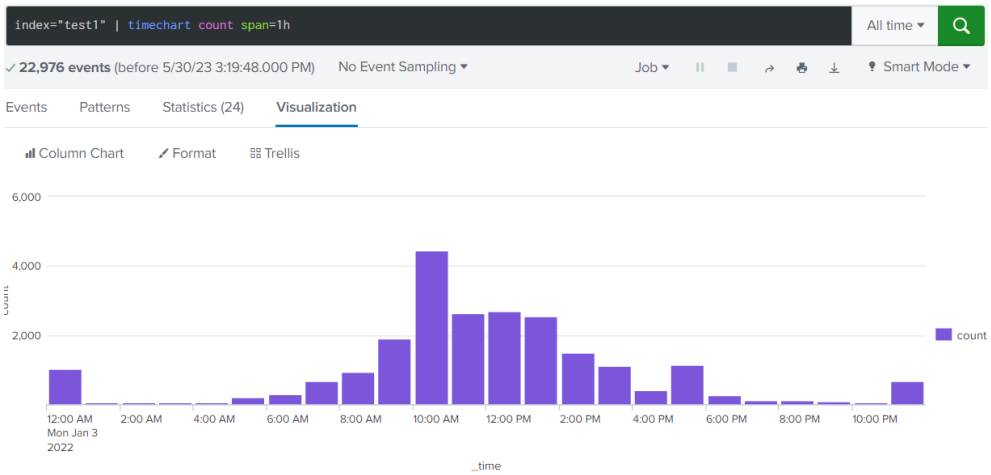



**Figure 7.3:** Event distribution of Test 1 (24h BASELINE)

Figure 7.5 shows the distribution of events of Test 4, 7 days generation on a normal schedule at 20x speed. The chart confirms that the timeline service can

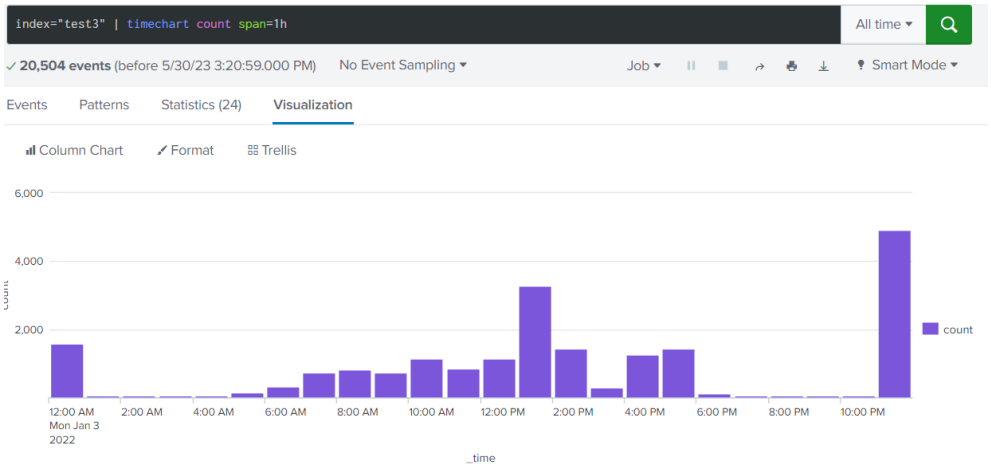



**Figure 7.4:** Event distribution of Test 3 (24h 20x)

successfully generate traffic in a relatively realistic manner in terms of volume, with little to no traffic during the night and on weekends.

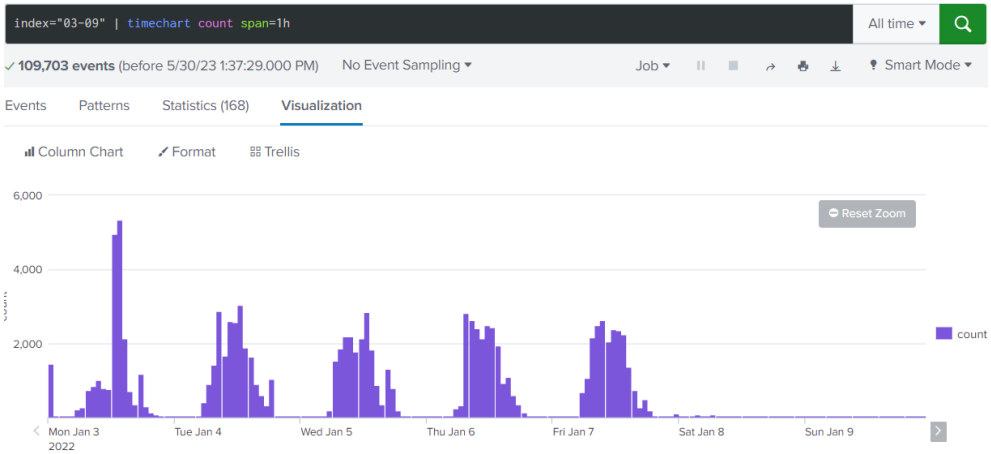



**Figure 7.5:** Event distribution of Test 4 (7d 20x)

One of the requirements of the historical network log generator was to have support for different traffic distributions to emulate different work hours. The two implemented schedules are 'normal' and '247', with the latter generating some traffic during the night and weekends, in addition to the daytime. 7 days of traffic using a timeline schedule generating events 24/7 is generated in test 5 with a speed of 20x. This traffic is displayed in Figure 7.6, and shows how the generated traffic is more evenly distributed throughout the week, although still with more traffic during the day than at night.

Using Test 6 (1d 20x with full packet capture), we can look at the event type categorization by Suricata in Figure 7.7. It shows that 44% of the events imported

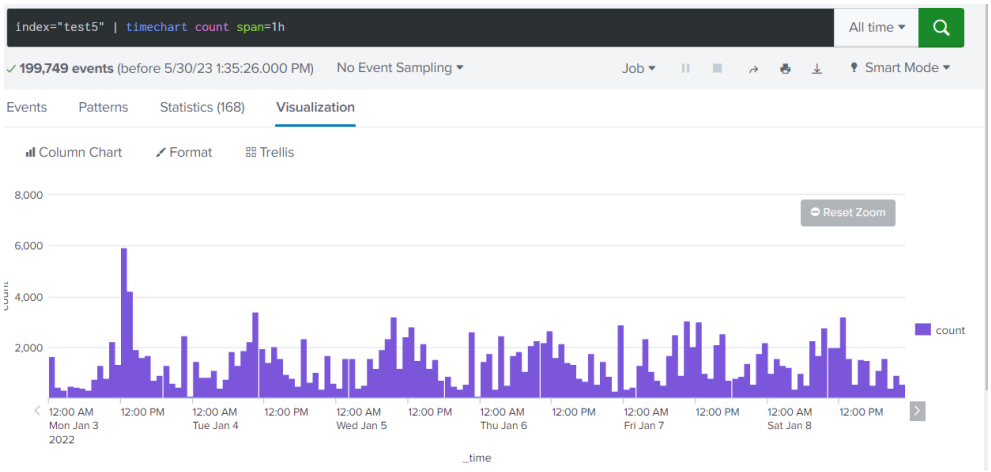



**Figure 7.6:** Event distribution of Test 5 (7d 20x 247-schedule)

to Splunk are classified as DNS, with 42% of the events being a flow. Searching within the flow event type, we can see the most common destination ports for the flows in 7.8. Primarily 53/DNS, 443/HTTPS, and 80/HTTP, which are to be expected as the current iteration of the artifact only emulates user activity on browsing websites.

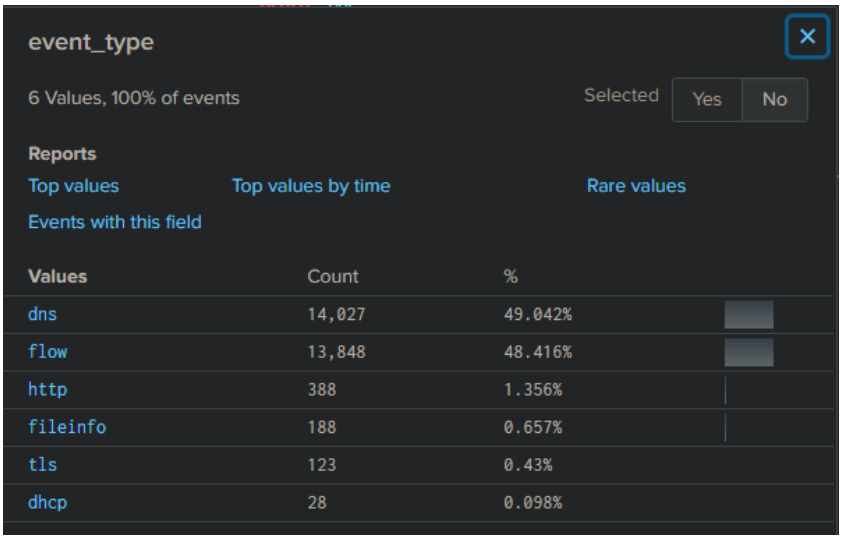



**Figure 7.7:** Event types categorized by Suricata

One can access additional information for certain event types, such as Domain Name System (DNS) and Transport Layer Security (TLS). TLS events include data about the certificate for the visited site. Two of those fields are the "notbefore" and "notafter" which indicate the validity time range for that particular certificate. The certificates are sent from the visited web server and reflect the current certificate

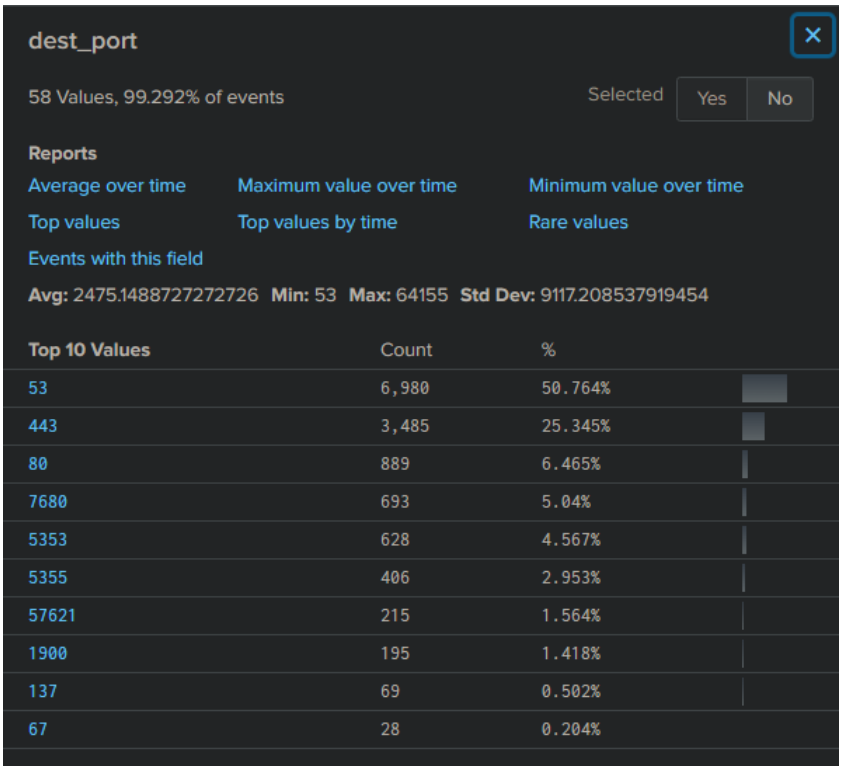



**Figure 7.8:** Destination port destination port of flow events from 7.7

available when viewing the website. Since this information is sent from the server, it is not affected by the manipulated clock on the server, and the timestamp may not be within the valid range. This is briefly touched upon in section 4.2.2. An example event for TLS is displayed in Figure 7.9. The timestamp for the event is from 03/01/2022, even though the certificate of the visited website is valid from 30/06/2022 until 30/09/2023, effectively being out of the valid timespan. The currently provided artifact is not able to deal with this. Additional screenshots of logs in raw PCAP format are found in section B.

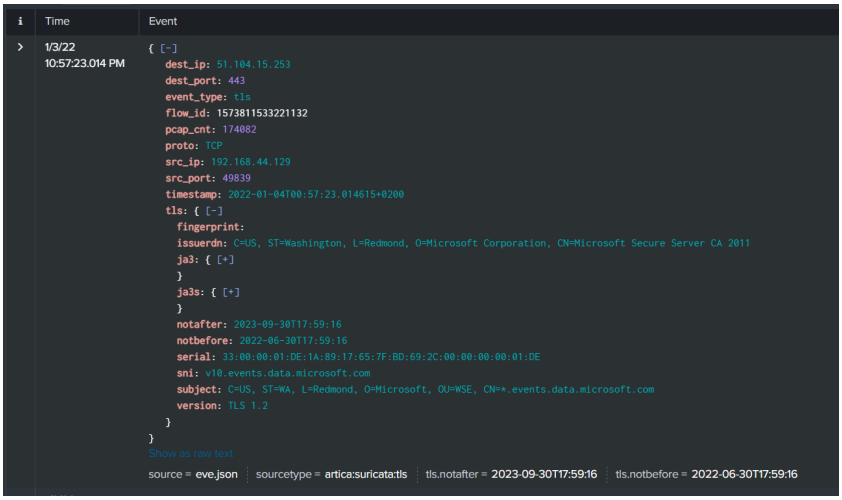



**Figure 7.9:** Randomly selected TLS event viewed in Splunk

### 7.1.4 External expert evaluation

An external expert working with network analysis and incident handling was given the logs to get a more objective evaluation of the logs. Prior experience is from three years of work with threat hunting and network analysis.

The expert did know how the logs were generated and what the thesis was about. None of the author's findings or thoughts were communicated to the expert before he had made up some thoughts and presented his findings. The analysis was performed in the same environment described above, with flow data converted by Suricata and loaded into Splunk. Some specific guidance questions were given along with the data samples:

- How are the logs different from regular traffic?
- Apart from what you see in the logs, can you think of anything that may compromise the integrity of the generated logs?
- Are the logs realistic-looking? Why/why not?

**Analysis:**

At first glance, they look realistic independently; nothing immediately sets the logs apart from normal. The logs generated at high speed show fewer events during "off-hours" when the client is not performing user actions. This is likely because the client is observing less background traffic since what may appear as 8hrs of the night in the logs is just 30 minutes of network traffic captured in real-time. This makes the generated logs distinguishable compared to a baseline of normal traffic captured at normal speed. There is an overall need for more background traffic, although that could be a challenge with the current technique used for generating logs faster than real time.

For HTTP(S) flows, the HTTP header "Date" is set by the server the client is talking to, reflecting the time of the web server and not the client's time. The response from the webservers will therefore have a header that does not correspond with the time of the logs as the client's time is being manipulated. This traffic is by default encrypted when using HTTPS and thus won't affect the integrity of the logs for the forensics investigators if they only have access to encrypted data. If the network data is decrypted, this information will be available and cause a mismatch between the "Date" header and the packet timestamp. B.1 show an example of the server-date found in an unencrypted HTTP session in a PCAP from Test 6. The timestamp of that particular HTTP packet-stream 15:56 on 3rd of January 2022. Over a year before the date issued by the web server.

### 7.1.5 Evaluation of functionality

Other network log generators exist with various functionality. Comparing this thesis' artifact with other network log generators helps determine what new functionality this artifact contributes and what tasks it solves differently than other solutions. A set of traffic generator functions are compared on a basic level in Table 7.2.

**Table 7.2:** Comparison of a selection of traffic generators

| Product | Generate traffic | User-realism | Capture to file | Historic | Remote activation |
|---|---|---|---|---|---|
| Cisco TREX | Benign: Yes<br>Malicious: No | No | Yes | No | No |
| Ghosts Framework | Benign: Yes<br>Malicious: Partly | Yes | No | No | Yes |
| **THESIS ARTIFACT** | Benign: Yes<br>Malicious: Yes | Yes | Yes | Yes | Yes |

The provided artifacts' primary task was to address the limitation of generating historical logs, as no other known tool is doing this for realistic-looking network

traffic. The GHOSTS framework mainly supports malicious traffic since it does not by default, but it is an open-source project with modules that can easily implement this functionality. Of the compared generators presented in Table 7.1, our proposed network traffic generator is able to achieve all of the evaluation points, which are a result of the limitations found during the study on state of the art.

### 7.1.6 Evaluation of requirements

An important step of the evaluation phase is to cover the requirements defined in Chapter 4.1 and to see to what degree they are fulfilled. A requirement defined can be fulfilled in either the design or implementation phase. If a requirement is fulfilled in the implementation phase, it is implied that it is also covered in design. Evaluating these requirements help grasp to what degree the thesis has reached and completed its objectives and solved the defined problem.

The requirements and their fulfillment for the historical network traffic generator are listed in table 7.3, and accompanied by a detailed evaluation of their fulfillment:

#### Customizable benign traffic

The dynamic benign traffic is primarily created by web-browsing agents simulating a real user. The search strings are fully customizable for each scenario and can be tailored for businesses by simply replacing the list of possible queries. Each event has also introduced randomness to make further the benign traffic look realistic.

#### Simple malicious events

Malicious events are implemented in terms of PowerShell commands. The commands simulating malicious activity are also customizable regarding the URL to download from or which file to upload.

#### Remote starting of traffic generation on client

The agent provides a module for remote activation. It supports communication over HTTP through a REST interface listening to incoming requests. The functionality is limited, but it provides the same functionality as starting. This requirement is classified as partly implemented, as this thesis does not supply a program to start the generator on several clients remotely. The REST interface is implemented on the agent and activated with the '–api' flag during CLI-start as discussed in 5.2

#### CLI-interface on the client itself to start generator

The agent software supports launching arguments directly on the client, generating the logs without needing a manager. Supported CLI arguments are *–start*

**Table 7.3:** Requirements and their fulfillment

| Requirement | Fulfillment |
|---|---|
| Customizable benign traffic | Design: Yes<br>Implementation: Yes |
| Customizable and simple malicious events | Design: Yes<br>Implementation: Yes |
| Remote starting of traffic generation on client | Design: Yes<br>Implementation: Partly |
| CLI-interface on the client itself to start the generator | Design: Yes<br>Implementation: Yes |
| Traffic from Windows and/or Linux | Design: Yes<br>Implementation: Yes |
| Configurable timeframe for logs | Design: Yes<br>Implementation: Yes |
| Realistic distribution of benign traffic | Design: Yes<br>Implementation: Yes |
| Custom distribution of malicious traffic | Design: Yes<br>Implementation: Yes |
| Easy to expand with more features | Design: Yes<br>Implementation: Yes |
| Capture traffic in suitable format | Design: Yes<br>Implementation: Yes |
| Transform PCAP to flow data | Design: Yes<br>Implementation: No |

which defines the date the generator should start at. *–stop* is specifying the stop date for the generator. *–schedule* accepts either "normal" or "247" indicating which work-schedule the events should be distributed for during a day. The final parameter *–speed* accepts either 10, 20 or 30, setting the clock-speed multiplier for when the system is not generating events.

### Traffic from Windows and/or Linux

The traffic generator is currently developed to work for Windows. Everything except the code controlling the Clock should, in theory, work on a Linux system without altering the codebase. The Clock modules use Windows-specific APIs and do not have cross-operating system compatibility.

#### Configurable timeframe for logs

The timeframe is set by a start and stop date of when to generate the logs for. The timeframes can be set several years in the past, and the generated logs will reflect this in their timestamps.

#### Realistic distribution of benign traffic

The agent has support for two different timeframes of logs simulating work schedules. One schedule is "normal", with most activity between 8 and 15, Monday to Friday. The other schedule is 24/7, with slightly more traffic during the day but still with traffic during the night and weekends, simulating activity levels at a hospital or similar. The benign traffic is fit to these schedules.

### Custom distribution of malicious traffic

The scheduler for malicious traffic has a slightly different configuration than benign traffic. The malicious traffic is set to a much lower frequency, with a 50% chance of generating 1 malicious event per day. The malicious-looking event will be generated from 21 in the evening to 04 in the morning, outside of normal, benign traffic.

### Easy to expand with more features

The code is designed in a class-based structure with future expansion in mind. It can easily be added with more modules to support more types of user activities, other work schedules, and other functionality. The main traffic generator, which reads events from the timeline, is designed so that functions for actions to do are dynamically called based on 'module' and 'method' for that particular event. This code design allows for implementing new features and sending emails without changing the generator's code. This functionality is found in the *dynamic_call* function in *src/agent.py*

## 7.1.7 Capture traffic in suitable format

The logs are captured and saved raw as PCAP files. The current configuration of the artifact supplied with this thesis will limit the capturing of packets to a certain size to keep the size of the PCAP file down. With no limit on packet size in the capture file, the PCAP will size about 1GB per day worth of logs and can easily be modified with the *snaplen* flag on WINDUMP, as per its documentation. The PCAP file is saved in the log generators directory and is ready for further processing if required.

### 7.1.8 Transform PCAP to flow data

This particular step was covered in the design phase, with a proposed solution being to use Suricata for transforming the captured PCAP files to flow data. As this functionality is to some degree outside the scope of the traffic generator itself and is on post-processing of data, it is not implemented in the current version. The technique of converting PCAP files to flow data is described above in Section 7.1.1.

## 7.2 Limitations

An extension of the limitation defined in Chapter 1, Introduction, with additions of limitations identified while evaluating the generator's performance. The generator mainly explored the generation of web-browsing traffic as the network data type emulating a user. The way the system is implemented makes for a relatively larger flow duration than normal, which limits the realism of the logs themselves, which could be distinguished from "normal" traffic.

The current iteration of the network traffic generator does not automatically supply the logs to a SIEM solution. The logs are available as raw files on the client who generates them and can be imported to any SIEM solution supporting logs from Suricata. However, importing the generated logs should not be challenging if the client already has a log exporter for a SIEM solution, such as WAZUH.

Even though the timestamps of the packets are correct in terms of being historical, some parts are not affected. For instance, HTTP packets have a "Date" header set by the web server before sending to the client. This header is not affected by the system clock and will therefore show real-time the website was visited. However, this requires access to the decrypted traffic to be viewed.

When the system is web-browsing using Selenium, it ignores all certificate errors, and the browser shows a warning that the website is not secure. However, the traffic is still encrypted and has no known impact on the captured PCAP traffic. The NCR plans to implement a PKI service for handling certificates in the future, and ignoring all certificate errors was a sufficient solution to the problem for this thesis.

The agent is designed to work on Windows systems, where the requirements are defined in either Windows or Linux. The Windows operating system dominates the market share for desktop computers with a 73.5% market share, making Windows a clear priority over a Linux agent [34].

## 7.3 Ethical considerations

The primary ethical consideration for this project is license requirements since it is heavily based on development. To my knowledge, no strict license agreements on components and code are used, limiting the implementation. The Accelerify clock tool discussed in 4.2.3 was the only component with an unknown license.

It was also one of the main contributing factors in recreating its functionality in Python from scratch.

The traffic generator has functionality implemented to replicate malicious traffic. It is, however, not dealing with any malicious and harmful data and only replicates patterns and characteristics of it.

# Chapter 8

# Conclusion and future work

## 8.1 Conclusion

The final activity in our implemented DSR methodology is to conclude and communicate the process and results. This thesis has looked into generating historical network traffic logs for use in cyber exercises and the related challenges. Even though the closely related topic of the real-time generation of network traffic has been discussed in numerous papers, there was no research on the generation of network logs that have historical characteristics and are generated significantly faster than in real-time. This conclusion, backed by a literature review, provides arguments for the first research question defined in Section 1.3, about state-of-the-art and limitations for historical log generation.

The second research question the thesis builds upon is how the limitations of state-of-the-art historical log generators can be addressed. Research already existed on generating network logs in real-time and with realistic user behavior, which made addressing the limitations regarding the time perspective of the logs the main subject of this thesis. This was addressed by manipulating the speed of the system's clock, simple user simulation on the system, and capturing the traffic on the network interface.

The developed artifact generates network logs in a PCAP format. The proposed network traffic generator in this thesis solves its problem by simulating a fast-moving clock on the system in order to traverse large timeframes in a short amount of time. During the day, the client will perform several user actions according to a generated timeline of randomized events following a work schedule. Meanwhile, all traffic is captured on the network interface and timestamped with the manipulated system time. The final result is client network logs generated for a customizable timeframe with user-simulating behavior at a significantly faster speed than the timeframe of the logs themselves.

The artifact shows a proof of concept of generating logs faster than in real time to show one way to solve the problem. The artifact does have some limitations in the current iteration. TLS certificates sent from the server may have issues and expiry dates outside of the timeframe of the logs themselves, and HTTP headers

sent from a web server to the client will have the server's timestamp and not the client. Additionally, the logs are only saved as PCAP and require further processing to be imported into software such as Splunk or ELK-stack. The source code for the artifact, including data samples, can be found on the Norwegian Cyber Range (NCR) GitHub[1], in addition to attachments to the thesis.

## 8.2 Future work

### HTTP-Header "Date" mismatching

For HTTP(S) flows, the HTTP header "Date" is set by the server the client is talking to, reflecting the time of the web server and not the client's time. This can theoretically be fixed by processing the PCAP and looking for HTTP packets to replace the "Date" value with the client's system clock at that specific time. To increase accuracy, the "Date" time value could be set to the current time, excluding the Return Trip Time (RTT) between the client and the server. Actions similar to this may corrupt the integrity of the packets and require a recalculation of the checksums.

### Unrealistic flow duration for background traffic

Background traffic is always generated, including when the clock on the system is moving significantly faster than in real time. Every flow has a timestamp for the first packet (flow.start), and when the last packet in that particular flow is received, a timestamp is created for the last packet (flow.end). Since the system's clock is moving significantly faster than in real-time, the timestamp difference between the first and last packet in a flow is larger than if the system clock was running at normal speed. This type of behavior is a byproduct of the technique used for generating logs fast. This might be addressed using a different technique, as mentioned below.

### Investigate alternative method for generating background traffic

A different technique for background traffic generation briefly touched upon in this thesis but not explored is to generate background traffic using traffic templates. These templates could include various background traffic patterns and assemble realistically in a tool such as Scapy to create PCAPs. With proper timestamp manipulation, it could be possible to generate historical network logs quickly and not compromise the flow duration as the technique explored in this project does.

[1] https://github.com/ncr-no/historical-log-generation

# Bibliography


[1] Purplesec. 'Cyber security statistics, the ultimate list of statsdata, & trends for 2023.' (2023), [Online]. Available: https://purplesec.us/resources/cyber-security-statistics/ (visited on 14/03/2023).

[2] G. TAG. 'Fog of war: How the ukraine conflict transformed the cyber threat landscape.' (2023), [Online]. Available: https://blog.google/threat-analysis-group/fog-of-war-how-the-ukraine-conflict-transformed-the-cyber-threat-landscape/ (visited on 14/03/2023).

[3] N. Gov. 'The cyber range: A guide.' (2023), [Online]. Available: https://www.nist.gov/system/files/documents/2020/06/25/The%5C%20Cyber%5C%20Range%5C%20-%5C%20A%5C%20Guide%5C%20%5C%28NIST-NICE%5C%29%5C%20%5C%28Draft%5C%29%5C%20-%5C%20062420_1315.pdf (visited on 16/03/2023).

[4] M. M. Yamin and B. Katt, 'Modeling and executing cyber security exercise scenarios in cyber ranges,' *Computers & Security*, vol. 116, p. 102 635, 2022.

[5] M. M. Yamin, B. Katt and V. Gkioulos, 'Cyber ranges and security testbeds: Scenarios, functions, tools and architecture,' *Computers & Security*, vol. 88, p. 101 636, 2020.

[6] Cloudshare. 'Why you need cloud-based cyber range simulation for cybersecurity training.' (2022), [Online]. Available: https://www.cloudshare.com/blog/cloud-based-cyber-range-simulation-for-cybersecurity-training/ (visited on 23/03/2023).

[7] NTNU. 'Norwegian cyber range.' (), [Online]. Available: https://www.ntnu.no/ncr#:~:text=Norwegian%5C%20Cyber%5C%20Range%5C%20er%5C%20en,for%5C%20cybertrusler%5C%20blir%5C%20stadig%5C%20st%5C%C3%5C%B8rre. (visited on 23/03/2023).

[8] NTNU. 'Ntnu has a training arena for handling cyber attacks.' (), [Online]. Available: https://norwegianscitechnews.com/2022/01/ntnu-has-a-training-arena-for-handling-cyber-attacks/ (visited on 23/03/2023).

[9] G. W. Øverli. 'Fullskala krisehåndteringsøvelse i norwegian cyber range.' (), [Online]. Available: https://www.oa.no/fullskala-krisehandteringsovelse-i-norwegian-cyber-range/s/5-35-1433118 (visited on 23/03/2023).

[10] R. Uetz, C. Hemminghaus, L. Hackländer, P. Schlipper and M. Henze, 'Reproducible and adaptable log data generation for sound cybersecurity experiments,' in *Annual Computer Security Applications Conference*, 2021, pp. 690–705.

[11] M. Hasbi, A. R. A. Nurwa, D. F. Priambodo and W. R. A. Putra, 'Infrastructure as code for security automation and network infrastructure monitoring,' *MATRIK: Jurnal Manajemen, Teknik Informatika dan Rekayasa Komputer*, vol. 22, no. 1, pp. 201–214, 2022.

[12] S. Beau and R. Rouquette, 'Under review: Automatic and realistic traffic generation in a cyber range,' Académie Militaire de Saint-Cyr Coëtquidan, Tech. Rep., 2022.

[13] D. D. Updyke, G. B. Dobson, T. G. Podnar, L. J. Osterritter, B. L. Earl and A. D. Cerini, 'Ghosts in the machine: A framework for cyber-warfare exercise npc simulation,' CARNEGIE-MELLON UNIV PITTSBURGH PA, Tech. Rep., 2018.

[14] M. Swann, J. Rose, G. Bendiab, S. Shiaeles and N. Savage, 'Tools for network traffic generation–a quantitative comparison,' *arXiv preprint arXiv:2109.02760*, 2021.

[15] C. Javali and G. Revadigar, 'Network web traffic generator for cyber range exercises,' in *2019 IEEE 44th Conference on Local Computer Networks (LCN)*, IEEE, 2019, pp. 308–315.

[16] F. Erlacher and F. Dressler, 'How to test an ids? genesids: An automated system for generating attack traffic,' in *Proceedings of the 2018 Workshop on Traffic Measurements for Cybersecurity*, 2018, pp. 46–51.

[17] J. L. Gjerstad, 'Generating labelled network datasets of apt with the mitre caldera framework,' M.S. thesis, 2022.

[18] M. L. Just Hacker Things, 'Converting pcap files to network flow data,' Tech. Rep., 2021.

[19] J. Vykopal, R. Ošlejšek, P. Čeleda, M. Vizvary and D. Tovarňák, 'Kypo cyber range: Design and use cases,' 2017.

[20] A. Hevner, S. Chatterjee, A. Hevner and S. Chatterjee, 'Design science research in information systems,' *Design research in information systems: theory and practice*, pp. 9–22, 2010.

[21] J. Iivari, 'A paradigmatic analysis of information systems as a design science,' *Scandinavian journal of information systems*, vol. 19, no. 2, p. 5, 2007.

[22] J. F. Nunamaker Jr, M. Chen and T. D. Purdin, 'Systems development in information systems research,' *Journal of management information systems*, vol. 7, no. 3, pp. 89–106, 1990.

[23] J. Eekels and N. F. Roozenburg, 'A methodological comparison of the structures of scientific research and engineering design: Their similarities and differences,' *Design studies*, vol. 12, no. 4, pp. 197–203, 1991.

[24] 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.

[25] J. G. Walls, G. R. Widmeyer and O. A. El Sawy, 'Building an information system design theory for vigilant eis,' *Information systems research*, vol. 3, no. 1, pp. 36–59, 1992.

[26] Microsoft. 'Command prompt (cmd. exe) command-line string limitation.' (), [Online]. Available: https://learn.microsoft.com/en-us/troubleshoot/windows-client/shell-experience/command-line-string-limitation (visited on 31/03/2023).

[27] T. Micro. 'Tracking, detecting, and thwarting powershell-based malware and attacks.' (2023), [Online]. Available: https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/tracking-detecting-and-thwarting-powershell-based-malware-and-attacks (visited on 19/03/2023).

[28] U. Manpage. 'Ubuntu manpage faketime.' (), [Online]. Available: https://manpages.ubuntu.com/manpages/trusty/man1/faketime.1.html (visited on 20/05/2023).

[29] R. Technology. 'Windump overview.' (), [Online]. Available: https://www.winpcap.org/windump/ (visited on 07/05/2023).

[30] Elvidence. 'Understanding time stamps in packet capture data (.pcap) files.' (), [Online]. Available: https://www.elvidence.com.au/understanding-time-stamps-in-packet-capture-data-pcap-files/ (visited on 18/05/2023).

[31] Mozilla. 'Setinterval().' (), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/setInterval (visited on 28/03/2023).

[32] J. Oduor. 'How to perform threading timer in python.' (), [Online]. Available: https://www.section.io/engineering-education/how-to-perform-threading-timer-in-python/ (visited on 28/03/2023).

[33] Vmware. 'Configuring time synchronization between guest and host operating systems.' (), [Online]. Available: https://docs.vmware.com/en/VMware-Tools/11.1.0/com.vmware.vsphere.vmwaretools.doc/GUID-C0D8326A-B6E7-4E61-8470-6C173FDDF656.html (visited on 29/04/2023).

[34] Scaler. 'What is the operating system market share?' (), [Online]. Available: https://www.scaler.com/topics/operating-system-market-share/ (visited on 06/05/2023).

# Appendix A

# Attached Material

Additional materials are delivered as attachments to the project report.

- Historical log generator code
  - /results - PCAP and Suricata flow data for test 1-5.
  - /src - All code used for the historical log generator

# Appendix B

# Network logs


**Figure B.1:** Visible HTTP-Date header compromising the integrity of the logs

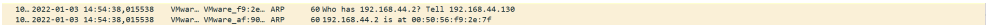



**Figure B.2:** ARP background traffic in the captured file

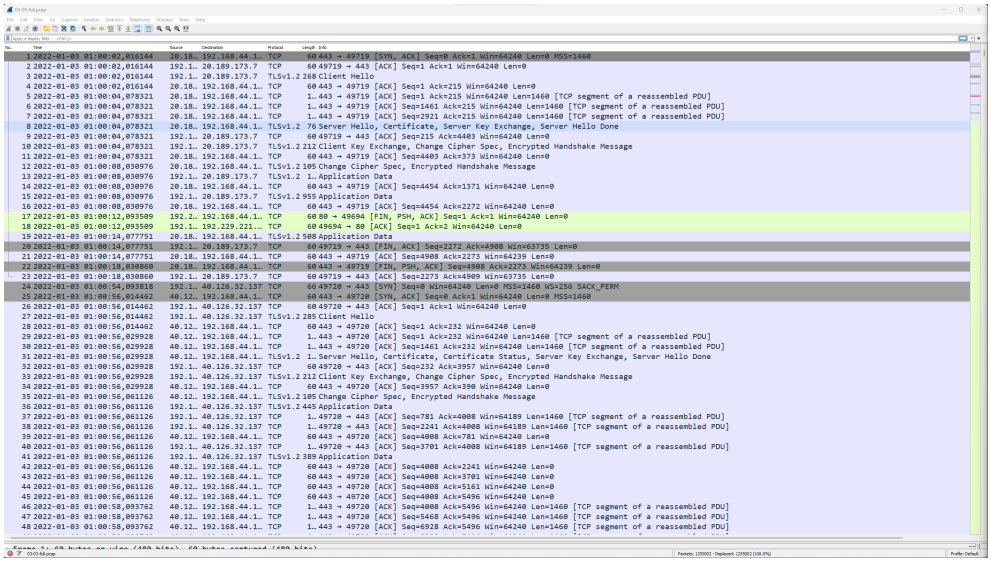
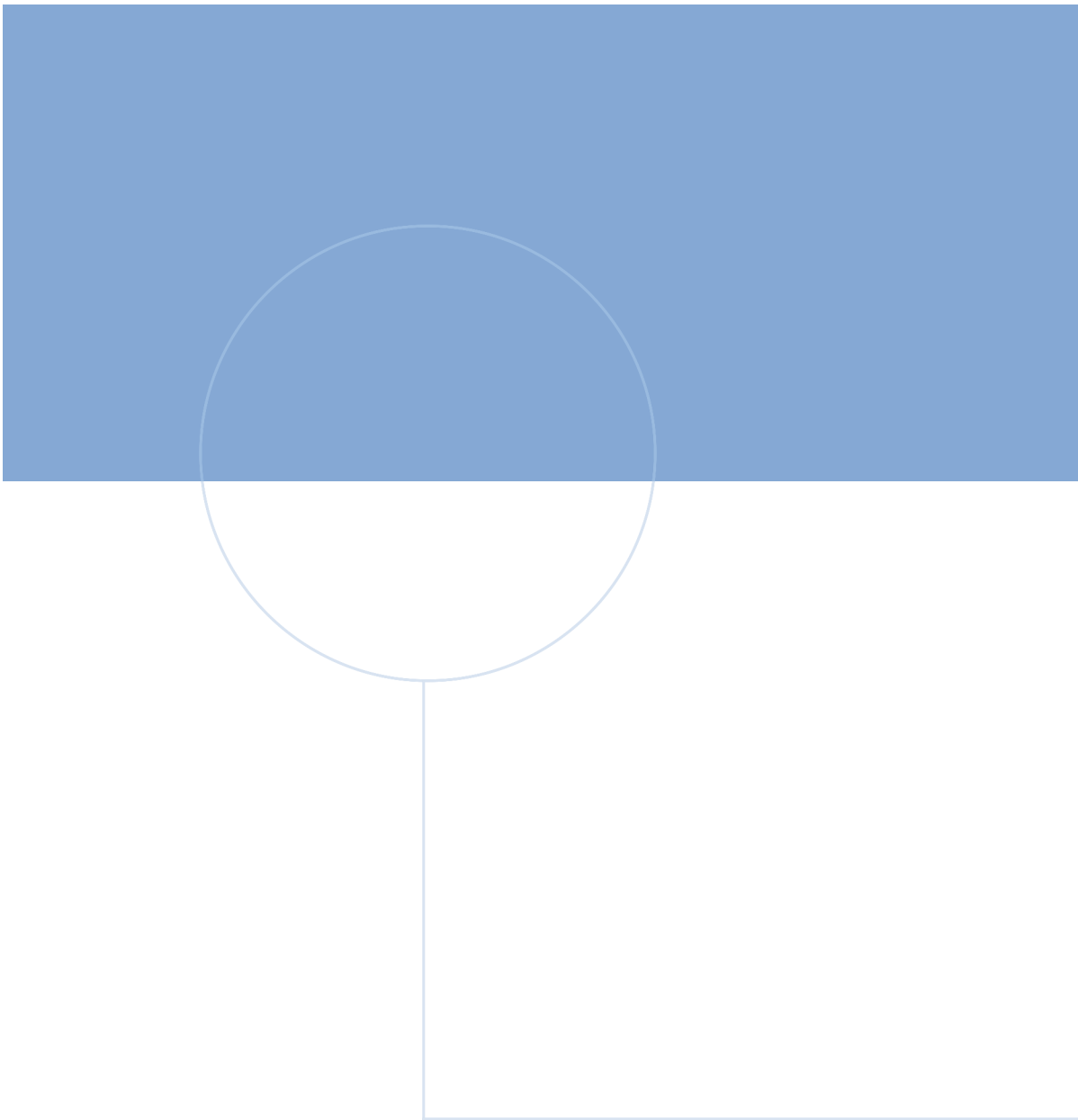


**Figure B.3:** The start of the PCAP file for Test 6.



NTNU

Norwegian University of
Science and Technology