TU WIEN Informatics

Qualitätsverbesserung der Labels für modellgesteuerte Benchmarkdatenerzeugung für Intrusion Detection Systeme

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering und Internet Computing

eingereicht von

Maximilian Frank
Matrikelnummer 01325864

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.univ.Prof. Dr. Andreas Rauber
Mitwirkung: Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Dr.rer.soc.oec. Florian Skopik
Dipl.-Ing. Max Landauer

Wien, 20. Mai 2021

Maximilian Frank	Andreas Rauber

TU WIEN Informatics

Quality improvement of labels for model-driven benchmark data generation for intrusion detection systems

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Maximilian Frank
Registration Number 01325864

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.univ.Prof. Dr. Andreas Rauber
Assistance: Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Dr.rer.soc.oec. Florian Skopik
Dipl.-Ing. Max Landauer

Vienna, 20th May, 2021

Maximilian Frank | Andreas Rauber

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Maximilian Frank

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20. Mai 2021

Maximilian Frank

Acknowledgements

I want to thank my current employer for facilitating this thesis and all my colleagues who supported me throughout this process. In particular I want to thank Florian Skopik and Max Landauer who not only provided valuable input and guidance for this thesis, but also had the displeasure of proof reading the first draft. Furthermore, I would also like to thank Prof. Andreas Rauber for supporting this thesis.

Last but not least I thank my family and friends for not only supporting me during this thesis, but also throughout my life. Especially, my brother who assisted in finding expert volunteers for evaluation of the thesis results. Finally, I wish to thank those that stood by me even though this thesis has put 9000km between us.

Kurzfassung

Computersysteme und Netzwerke werden jedes Jahr komplexer. Sie werden auch immer mehr in die verschiedenen Aspekte des täglichen Lebens und der globalen Industrie integriert. Daher wird die Gewährleistung der Sicherheit von Computersystemen immer wichtiger, während aber zugleich auch die Anzahl und Komplexität von Angriffen auf solche Netzwerke steigt. Intrusion Detection Systeme verwenden verschiedene Algorithmen und Techniken um Angriffe auf Computersysteme und Netzwerke zu erkennen. Sie werden daher oft von Cyber Security Teams als Teil des Sicherheitskonzepts verwendet.

Netzwerkverkehr und System Log Datensätze von sowohl normalen Benutzer als auch Angreifer Aktivitäten werden verwendet um Intrusion Detection Systeme zu entwickeln, zu verifizieren und um sie bezüglich ihrer Erkennungsrate zu vergleichen. Diese Datensätze beinhalten normalerweise auch Labels das heißt, Einträge in den Datensätzen die durch Angreifer Aktivitäten entstanden sind wurden dementsprechend markiert. Solche Markierungen sind notwendig um Algorithmen und Techniken korrekt bewerten zu können. Existierende Arbeiten über die Generierung von solchen markierten Datensätzen legen den Fokus hauptsächlich auf die Datensätze selbst. Die Prozesse für das Zuweisen von Labels werden nur als sekundäre Resultate betrachtet und sind meist nicht für andere Datensätze wiederverwendbar. In dieser Arbeit stellen wir Cyber Range Kyoushi vor, eine Methodik und ein Framework für die Generierung von markierten Datensätzen. Cyber Range Kyoushi baut dazu auf State-of-the-Art Arbeiten zum Thema Datensatzgenerierung mit modellgetriebenen Testbeds auf.

Das vorgestellte Framework wurde auch mit Open Source Technologien und Software Bibliotheken, die im Rahmen der Arbeit speziell entwickelt wurden, implementiert. Weiters wurde die vorgeschlagene Methodik auch analysiert und ausgewertet. Hierzu wurden ein Cyber Range Testbed und zwei Referenz Angriffsszenarios erstellt und mit dem Cyber Range Kyoushi Framework umgesetzt. Die Qualität der daraus entstandenen markierten Datensätze wurde dann auch anhand einer Expertenumfrage überprüft. Die Implementierung der Referenzszenarien und die darauf folgende Expertenauswertung zeigen, dass das vorgeschlagene Framework die effiziente Generierung von präzise markierten Datensätzen zur Evaluierung von Intrusion Detection Systemen ermöglicht.

Abstract

Computer systems and networks become more complex every year. They are also integrated into ever more aspects of daily life and global industries. As such, the security of computer systems becomes more and more important, while at the same time the threats and attacks on these systems increase in frequency and complexity. Intrusion Detection Systems, using various algorithms and techniques to detect malicious activities on systems or computer networks, are therefore commonly used by cyber security personnel to counteract intrusions.

Datasets containing traffic and system logs of both benign and malicious events are used to develop, verify and benchmark the algorithms and detection techniques used by Intrusion Detection Systems. These datasets usually also have to be labeled, i.e., entries corresponding to malicious activities are marked as such, to properly evaluate the correctness of tested algorithms or techniques. Existing works on the generation of such labeled datasets mostly focus on the datasets themselves. The used labeling processes are treated as secondary results and are usually applicable and specific to the original work only. In this thesis we propose Cyber Range Kyoushi, a framework and methodology for generating and labeling datasets based on previous work done on dataset generation using model-driven simulation testbeds.

The proposed framework was also implemented using open source technologies and custom software libraries developed as part of this thesis. The soundness of the proposed approach was further evaluated and analyzed. For this, a cyber range testbed and two reference scenarios were designed and realized using the Cyber Range Kyoushi framework. Quality of the generated labeled datasets was also verified through an expert survey. The reference scenario's implementation and subsequent expert survey show that the proposed framework enables to efficiently create precisely labeled datasets for Intrusion Detection Systems.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Computer systems and networks, especially those that are connected to the Internet, are under constant threat of cyber attacks. Modern society has seen the ever increasing integration of information technologies into many aspects of life, for example, in the form of Internet of Things (IoT) devices such as home assistants, smart watches or fridges. Industry and infrastructure have also seen a steady adoption of more complex and inter connected computer systems. So called Cyber-Physical Systems (CPS) have been created bridging the gap between operational and informational technologies sometimes not only connecting devices within a factory, but also across a network of factories. The energy sector also has seen the advent of so called smart energy grids using complex computer and sensor networks to manage and control energy distribution. The adoption of Cyber-Physical Systems and increased integration of information technologies into critical infrastructures, such as the energy sector, has also introduced both new attack vectors as well as targets for cyber attacks, making it evermore critical to be able to properly protect vital systems.

One of the first steps to prevent or recover from a cyber attack is detecting the malicious activity. The changing technological and cyber attack landscapes have therefore created strong economic and research interest into the ability to detect both known and still unknown attacks. This is commonly achieved using Intrusion Detection Systems (IDS) which are used to detect ongoing or past attacks on a network (NIDS), host (HIDS) or both. Early intrusion detection systems relied only on rule-based signatures and stateful packet inspection to detect cyber attacks, but these mechanism were not sufficient to detect sophisticated attacks and not previously encountered attacks. Anomaly detection, where the IDS tries to detect attacks by looking for deviations from normal behavior, are one type of techniques developed to be able to detect such attacks. These techniques usually

are self-learning and able to detect malicious activity autonomously after establishing a baseline for normal behavior [Cic+12; SM12].

Realistic labeled data sets containing both normal user and malicious behavior are needed not only for researching and training these anomaly-detection systems, but also to benchmark them. Due to the rapid changes in the attack landscape it is important that data sets contain state of the art attacks to ensure that new attack types can also be detected by developed systems. Labels, used to assign data set entries to a class, e.g., normal or attack, are also an important requirement for good IDS data sets. As such there exists an ongoing effort to create newer and better IDS datasets [Rin+19; SM12].

1.2 Problem Statement

One way to generate data sets for intrusion detection systems is by simulating real world attack scenarios in cyber security testbeds; or sometimes also referred to as cyber ranges. Cyber security testbeds can be used to simulate ICT, OT, mobile and physical systems. Depending on the testbed, this might be achieved, for example, by using virtualization technologies, replication of infrastructure using physical components or a combination of both. Data sets generated from cyber security testbeds are sometimes referred to as semi-synthetic or synthetic data sets. For example, Ring et al. [Rin+17] used a cyber security testbed, based on the open source cloud computing platform OpenStack[1], to generate a synthetic data set for network intrusion detection systems. Synthetic data sets generated using testbeds can be labeled very well as both the environment and the attack scenario are well understood and can be controlled by the researchers. Additionally, synthetic data sets can be published in their raw state since there is no need to redact or anonymize data, which would not be possible due to privacy concerns and regulation with data sets collected from real world systems [Lan+20b; Oge+15; Rin+17].

Landauer et al. [Lan+20b] show in their work how a model based cyber security testbed can be used to simulate attacks on realistic information & communication technology systems and generate data sets for host intrusion detection systems. By using such an approach it is fairly easy to generate different data sets by simply tweaking the constraints and variables, once a system and attack model are defined, but labeling the resulting data sets still proves a challenge involving manual and time-consuming tasks. Landauer et al. [Lan+20b] rely on a two stage labeling process to label data sets generated using their methodology. For the first stage they assign labels based on attack timings while the second stage involves the manual extraction of complex matching rules from a data set generated without the noise of normal user operations. Depending on the complexity and variability of the defined testbed and attack model this manual process might have to be repeated for each testbed instantiation. Thus potentially making it a major bottleneck slowing down rapid data set generation.

[1] https://www.openstack.org/

While aforementioned works show the potential for rapid data set generation and thus satisfy the increasing demand, the effectiveness of their proposed approaches is seriously hampered by bottlenecks in the methodology. As discussed above, the labeling process is one of these bottlenecks, even if a system can produce many data sets in a short time spawn the resulting data sets would not be useful for things like supervised learning or benchmarking as long as they are not labeled. This raises the need to not only support rapid data set generation, but also a system or framework that makes it possible to quickly and efficiently label the generated data sets.

From the problem described above follow several questions concerning the definition, creation and evaluation of a data set labeling framework or methodology. The following are the most relevant questions:

RQ1 Based on the attack scenarios, what data can actually be labeled and in which level of detail?

RQ2 What are artifacts of the testbed simulation process and how can they be recorded?

RQ3 What requirements for automated labeling framework for cyber security testbeds can be derived from the scenarios and to what extent does our proposed solution fulfill them?

RQ4 To what extent do cyber security experts agree with the labeling provided by the framework?

RQ5 How can state of the art sampling techniques be used to efficiently evaluate the proposed solution?

Questions 1-3 are at the core of this thesis and concern the creation and definition of the proposed framework. We will explore the problem space of labeling data sets for Host Intrusion Detection Systems based on these questions. Afterwards we will try to answer question 4 and 5 in an effort to evaluate the resulting prototype.

1.3 Aim of the Work

The aim of this thesis is to design, implement and evaluate an attack simulation and labeling framework which can be used to rapidly generate data sets for training and researching host intrusion detection systems. The to be developed framework should be based and improve upon the work done by [Lan+20b] with a focus on introducing a higher level of automation for the labeling process. This will be achieved by making the labeling process an inherent part of the attack model, allowing researchers to define labeling rules and functions as part of the attack model. The labeling rules and functions are then used to correlate the model and information artifacts recorded during attack simulations to the generated log data.

As such the proposed framework, in combination with the work already done by Landauer et al. [Lan+20b], aims at providing a solution for the problem of generating realistic labeled data sets for researching, evaluating and training machine learning algorithms for Host Intrusion Detection Systems. The proposed framework shall be called Cyber Range Kyoushi (kʲoːɕi), or short CR Kyoushi, in further sections of this thesis. This name is taken from the Japanese where it is the word for *teacher*. The word is also used in Japanese terminology for training data set (i.e., kyoushi dataset).

This thesis aims to achieve the following contributions by providing the above mentioned framework:

- a system and methodology description for labeling Host Intrusion Detection System data sets generated using cyber range systems
- two reference attack scenarios which can be used to verify the correctness of the system and methodology and compare it to other approaches
- a proof of concept implementation of said system
- an evaluation of the methodology by applying it to the reference scenarios and determining the quality of the resulting labels

1.4 Methodological Approach

First we will explore state of the art Intrusion Detection System data sets. This exploration will have a particular focus on data sets generated using cyber range testbeds as they are within the same problem space as this thesis, but other data sets e.g., those generated from real attacks will also be considered. The state of the art data sets and cyber ranges will give us an insight into both simulation and labeling techniques that have been previously used.

Using the gathered information we will then define two reference attack scenarios. Based on the explored state of the art data sets and these two reference scenarios we will then define initial requirements and create a partial proof of concept implementation of the labeling framework. We will then fully implement the two reference scenarios using this partial implementation and the cyber range technology stack described by Landauer et al. [Lan+20b] and Leitner et al. [Lei+20]. Further we will also generate data sets from the implemented reference scenarios. Based on the generated data sets we will then further refine the our proposed labeling methodology, proof of concept implementation and labeling rules. As a result of this process we will obtain two data sets labeled by our proof of concept implementation.

Finally, we will aim to evaluate and validate our labeling methodology and proof of concept implementation. To this end we will explore state of the art data set sampling techniques. For this we will put and particular focus on sampling techniques that can

be used to approximate precision recall curves. Sampling techniques are reviewed and carefully selected to ensure that the sample size required for the evaluation process is as small as possible. This is necessary since the validation of labeled log lines is a time intensive process that can only be done by validators with specialized security and domain knowledge. Precision recall curves will then be used to evaluate the quality of the labeled data sets and validate the correctness of the labeling framework.

1.5 Structure of the Work

The remainder of this work is organized as follows: Chapter 2 surveys existing Intrusion Detection System technologies and datasets, with a focus on works that use virtualization and simulation technologies. In chapters 3 and 4 we then define a cyber security testbed and two attack scenarios. We use the testbed and the attack scenarios as reference models during the development and evaluation of the proposed IDS dataset generation and labeling framework and methodology. In chapter 5 we then discuss the proposed framework and its components in detail. Application of the framework to the two reference attack scenarios is discussed in chapter 6. We show how the proposed framework is adequate to both generate datasets for the scenarios as well as label them. In chapter 7 we further discuss the qualitative evaluation of the resulting labeled datasets. For this we use and expert survey to gather cyber security expert consensus on the quality of the labels assigned using the framework. Finally. we conclude the thesis and discuss future work in chapter 8.

CHAPTER 2

Background & Related Work

Within the last few years, rapid growth in information technologies and ever increasing integration thereof into various processes and businesses (e.g., industry 4.0, smart grids) occurred hand in hand with an increase of cyber attacks. The steady flow of every increasingly complex and frequent cyber attacks as, for example, discussed by Ray et al. [Ray+20] necessitates the ability to automatically detect these attacks. This chapter therefore provides a review of existing intrusion detection systems and specifically focuses on security datasets that enable the evaluation of their detection capabilities [VAB18; WL13].

2.1 Intrusion Detection Systems

Attacks compromising the confidentiality, integrity and availability of a system or otherwise try to bypass security mechanisms of a computer network are commonly referred to as *intrusions*. Software or hardware systems which monitor computer networks and systems and are able to automatically detect intrusions are called Intrusion Detection System (IDS). Intrusion Detection Systems have become an essential part of modern network and host security systems, in fact, works as early as Bace and Mell [BM01] discuss that when creating a security concepts for computer systems it should never be a question if an IDS should be used, but rather what type of IDS should be used. Note that Intrusion Detection Systems are not to be confused with Intrusion Prevention Systems (IPS). Intrusion Detection Systems only detect malicious activity and produce alerts, while Intrusion Prevention Systems usually use an IDS to first detect an attack and then try to actively stop it, for example, by reconfiguring the system (e.g., blocking a malicious IP address) [SM07; BM01].

2.1.1 Detection Methodologies

According to Liao et al. [Lia+13] and the NIST IDS guidelines as defined by Scarfone and Mell [SM07], Intrusion Detection Systems detection methodologies can be classified into three major categories: Signature-based Detection, Anomaly-based Detection and Stateful Protocol Analysis.

Signature-based Detection. In the context of cyber security systems, e.g., IDS or malware analysis and detection, signatures are defined as recognizable patterns that can be linked to specific known attacks or threat actors. Signatures can be based on different aspects, for example, behavioral signatures might be defined as event sequences describing the patterns of various attacks and attack steps. Structural signatures on the other hand are defined on, for example, the binaries or systems involved in the attack (e.g., file hashes or attacker controlled host IP addresses) [Nai+19; KK04; Lia+13].

Definition of signatures requires thorough knowledge of the attacks or threat actors. Accordingly, signature-based detections is thus also sometimes referred to as *knowledge-based detection*. As such, signatures work well with known attacks, but are unable to detect new unknown attacks. Note that sophisticated attackers will also employ various signature evasion techniques to avoid detection. For example, Kolesnikov and Lee [KL05] discuss how attackers use polymorphic worms to avoid IDS signature detection mechanisms [SM12; Lia+13].

Stateful Protocol Analysis. Stateful protocol analysis, also described by Liao et al. [Lia+13] as *specification-based* detection, is based on definitions of default (benign) behavior for the observed systems or protocols. Such definitions (specifications) are usually based on generic profiles provided by the software or systems vendors and the internationally standardized protocol definitions [Lia+13]. Intrusion Detection Systems employing stateful protocol analysis techniques must be able to observe and fully inspect and understand the network packets or applications logs as to have access to the *state* of an user connection or interaction [Lia+13].

While signature-based detection follows a blacklist approach, i.e., signatures define malicious patters that indicate attacks, stateful protocol analysis techniques follow a whitelist approach. That is, the benign traffic or interaction profiles define what is allowed (i.e., the whitelist) and everything that does not match these profiles is considered to be malicious [Lia+13; Lan18].

Anomaly Detection. As the name suggests, anomaly detection techniques try to find anomalies, i.e., deviations from normal (benign) user behavior or network traffic, and classify these anomalies as suspicious or possibly malicious. Anomaly detection is thus also commonly referred to using the more general term behavior-based detection. Anomaly detection techniques monitor different aspects of systems and networks and detectable anomalies depend on the defined activity or traffic profiles, e.g., sudden spikes in the number of failed login attempts or increased/decreased CPU usage [Lia+13; Alm+17].

While both anomaly detection and stateful protocol analysis rely on benign network traffic or user profiles to define a baseline, the used profiles and techniques are different. Stateful protocol analysis usually employs static or generic profiles provided by the vendors, while profiles used in anomaly detection are usually *learned* from real networks or systems. That is, user interactions and traffic known to be benign is observed by the system and baseline profiles are extracted from the resulting traffic or system logs [Lia+13; Alm+17; Vin+19].

2.1.2 IDS Types

Each of the three discussed detection techniques has its advantages and disadvantages. For example, as already discussed above, signature based detection is very good at detecting known attacks, but is not able to detect unknown attacks. Conversely, anomaly detection can also detect unknown attacks, but is more prone to produce false positives (i.e., benign traffic/interaction incorrectly detected as malicious). Modern Intrusion Detection Systems therefore usually implement hybrid approaches, i.e., they are able to employ multiple or even all three described detection techniques [Lia+13].

Intrusion Detection Systems are therefore often not categorized by their detection techniques, but rather by the domain they monitor. The NIST intrusion detection and prevention system guide ([SM07]) defines four types of IDS types based on this:

- **Network-Based:** This type of IDS monitors network traffic and analyzes high level network protocols and application specific network protocols to detect malicious activity. Such monitoring is often done at the borders of two or multiple network segments (e.g., on the firewalls separating two subnets).

- **Network Behavior Analysis (NBA):** While Network-Based Intrusion Detection Systems analyze network packets and protocols, network behavior analysis systems analyze so called network flows, i.e., statistical aggregations of the current traffic (e.g., request counts per protocol, unique source IP addresses, etc.). These types of systems are usually placed at the borders separating internal systems from the external networks (e.g., internet).

- **Wireless:** This is a special type of Intrusion Detection Systems used to monitor low level wireless network traffic that transports packets for the more higher level network protocols. Wireless networks are often attractive targets for attackers as they do not require a physical network connection (e.g., ethernet), but rather attackers can interact with wireless systems by simply being in their proximity.

- **Host-Based:** Host-based systems are deployed on single hosts within a network. These systems might monitor a hosts system logs, metrics (e.g., CPU load) and network traffic.

Both network-based and network behavior analysis IDS are also commonly referred to as Network Intrusion Detection System (NIDS). Similarly host-based systems are referred to as Host Intrusion Detection System (HIDS), host-based systems that also monitor a hosts network traffic are sometimes also referred to as Network and Host Intrusion Detection System (NHIDS) [Cic+12].

2.2 Anomaly Detection and Machine Learning

As discussed above many Intrusion Detection Systems utilize anomaly detection mechanisms to identify malicious network or host activity. These systems usually employ machine learning techniques to build the benign activity and network traffic models used as base lines to detect anomalies. IDS datasets are commonly used for this purpose.

2.2.1 Training Modes

Anomaly detection techniques can be categorized into three different categories based on the types of datasets used during the training phase [CBK09].

Supervised Anomaly Detection. This type of anomaly detection technique requires labeled IDS datasets. A label is a marker placed on log lines or network traffic that indicates whether the corresponding data point is malicious or benign. Note that more detailed labels, i.e., indicating specific attack types or user behavior, are also possible. The labels enable that the anomaly detection algorithm learns a profile for both benign and malicious activity [CBK09].

Semisupervised Anomaly Detection. In contrast to supervised anomaly detection techniques, semisupervised techniques only require that the benign activity is labeled in the training dataset. This makes these techniques easier to apply to problem domains for which anomalous behavior is hard to model or predict. Semisupervised anomaly detection techniques usually build benign profiles based on the training dataset and detect anomalies based on deviations from the trained benign profiles [CBK09].

Unsupervised Anomaly Detection. Unsupervised anomaly detection techniques do not require any labeled training data at all. These techniques work under the assumption that benign activities and traffic outweigh any anomalies that might occur. Since unsupervised anomaly detection techniques do not require any prior knowledge about the monitoring data, they can be applied to many detection problems for which supervised techniques cannot be used. It is also to note that semisupervised anomaly detection techniques can be adapted to operate in an unsupervised mode by pre-training them with unlabeled dataset samples [CBK09; Lan+18].

2.3 IDS Dataset Generation

Datasets are essential for IDS research and development. As such there is an continuous effort by the research community to create realistic and up-to-date datasets. Throughout the years datasets have been produced using various different technologies and methodologies, e.g., using cyber ranges, honey pots or by emulating network traffic. We summarize a few selected works on dataset generation below.

Cyber Ranges & Testbeds Cyber ranges or security testbeds provide researchers with the ability to simulate complex computer systems and networks. This is usually achieved through virtualization hypervisor technologies and software defined networking. Depending on the utilized cyber range technologies it might also be possible to integrate hardware devices into defined virtual networks. Cyber range testbeds are utilized for a variety of IT security related activities, e.g., to provide a secure training or research environment. They are also often used to create IDS datasets [FLP17; Lei+20; YKG20; Lan+20b].

Landauer et al. [Lan+20b] describe in their work a model driven cyber security testbed generation approach. Their approach allows them to rapidly instantiate multiple variations (i.e., amount and types of machines or users) of a testbed which can be used to generate datasets for both network as well as host intrusion detection systems. Their labeling process uses two mechanisms. The first mechanism is based on a simple time based matching by recording the attack timings of the simulated attacks and matching them against the generated log data. Their second labeling mechanism involves the manual extraction of complex matching rules for the specific attack through the analysis of a dataset generated without the noise of normal user activity.

Similar to Landauer et al. [Lan+20b] Ring et al. [Rin+17] also use an OpenStack[1] based cyber security testbed to generate their network intrusion detection system dataset. In their work they also describe a four stage labeling process which relies on meta information such as designated attacker and non-malicious IP addresses and attack time stamps. The used testbed also contained one Internet connected component for which only a partial labeling was done as it was not possible to definitively differentiate between malicious and non-malicious traffic originating from the Internet.

Containerization. Container technologies provide a light weight alternative to classical hypervisor virtualization. They work by running applications in isolated environments on the same operating system. That is, by using network, system and application isolation techniques, containers are isolated from each other, but still use the same underlying operating system. While this makes containerization less powerful than virtualization, e.g., different hardware architectures cannot be simulated, it also reduces the required overhead significantly as there is no need to run a virtual operating system for every containerized application.

[1] https://www.openstack.org/

Grimmer et al. [Gri+19] developed the Leipzig Intrusion Detection - Data Set (LID-DS) to address two main issues they found with previously available datasets. For one they thought the publicly available datasets at that time were outdated (i.e., were based on out of date software). The second issue they had with most of the available datasets was that multi processor architectures and multi threading was not considered enough during dataset generation. Thread information was not readily available thus limiting their usefulness for researching modern IDS techniques. Their dataset generation heavily relies on containerization using Docker and recording of system calls using Sysdig. Using these two technologies they record multiple runs of attacks and normal user behavior. In their dataset labeling procedure they limit themselves to marking runs as either containing malicious activity or not. For runs with malicious activity it is also noted at which point the malicious activity starts.

Honeypots Similar to cyber security testbeds, honeypots are simulated computer systems and networks, but in contrast to testbeds, honeypots are not meant to provide a secure environment for experimentation. In fact, the opposite is the case, since honeypots are deliberately designed and created to be insecure systems that can be targeted by real malicious actors. Another major difference between honeypots and cyber security testbeds is their placement. While testbeds are usually isolated systems, honeypots are most of the time placed alongside production system, i.e., honeypot machines are placed in the same network zones as normal production systems [MA07; Son+11].

Song et al. [Son+11] created and operated multiple honeypots over the span of three years in an effort to create a dataset for Network Intrusion Detection Systems. They placed 348 honeypots with a variety of different machine configurations, e.g., different security patch levels and operating systems, in various network locations inside and outside the Kyoto University's network infrastructure. The machines were configured to automatically reset to a fresh installation state when malicious activity was detected. They further label the raw dataset collected from the network traffic of the honeypot machines using various IDS systems, e.g., Ashula[2] detection rules.

[2] https://sites.google.com/a/secure-ware.com/securewareproducts/ashula

CHAPTER 3

Testbed & Simulation

In this chapter we describe the cyber range testbed we created to evaluate our proposed labeled dataset generation approach. First we will describe the testbed environment, i.e., the network and system configuration of the testbed. Then we will discuss the user simulation model which is used to generate base line traffic and activities within our testbed. The two attack scenarios for our two proof of concept datasets are discussed separately in chapter 4.

3.1 Environment

Figure 3.1 shows an overview of the networks and server components present in our cyber range testbed. This cyber range testbed is an evolution of the testbed used by Landauer et al. [Lan+20a] to generate their IDS dataset. As such we also follow the same principle of a model driven testbed definition and generation process described by Landauer et al. [Lan+20b] and Leitner et al. [Lei+20]. Using the model-driven simulation testbed methodology we generated a single static testbed configuration that is suitable to for both of our attack scenarios discussed below.

The testbed shown in fig. 3.1 contains a total of 33 computer systems and is modeled to simulate a small to medium companies network infrastructure. The simulated company has two network zones separated by a firewall:

- the *intranet* containing the employee desktops and all servers that are only used internally
- the Demilitarized Zone (DMZ) primarily used for servers which provide some service that is accessible through the *internet*.

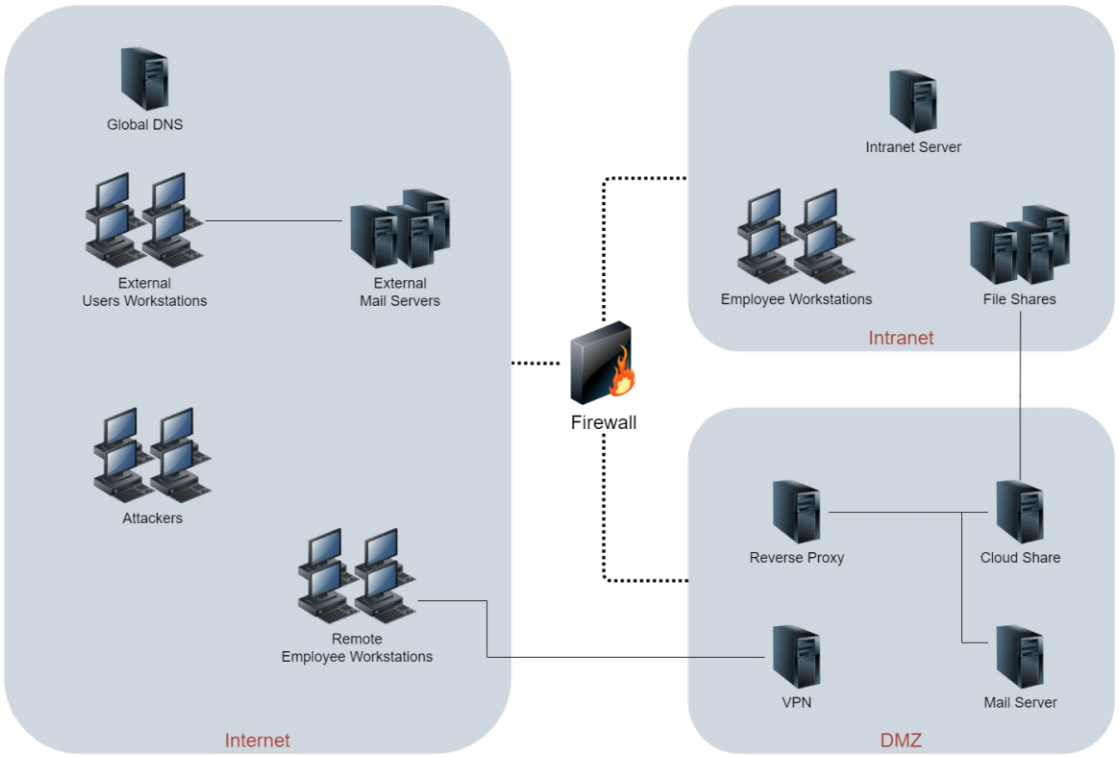

Figure 3.1: Testbed Network Overview

The internet as depicted in the network overview figure is special construct as it is a virtual network created as part of the testbed. Inside the testbed it serves two purposes. First of, it contains all company external machines i.e., those which would interact with the companies firewall through the internet in a real world scenario. Second, it serves as gateway network to the real internet which might be accessed as part of the scenario or the normal operations of servers contained within the company network (e.g., automatic software update checks).

Below we will briefly discuss the various machines that are part of the testbed configuration. Note that multiple server or desktop system icons above a single text label indicate a server or client group. For example the group *Employee Workstations* is used to show that multiple desktop clients for company employees exist within the intranet.

Global DNS This server is used as the authoritative DNS server for the whole testbed internet zone as well as resolver for DNS queries of hosts outside the testbed (i.e., real internet hosts). For this purpose Dnsmasq[1] is configured as DNS resolver and forwarder and MaraDNS[2] provides the authoritative DNS features.

[1] `https://thekelleys.org.uk/dnsmasq/doc.html`
[2] `https://maradns.samiam.org/`

Attackers The Attackers server group is used for the attacker work stations used in various scenarios. For both our evaluation scenarios a single attacker work station is sufficient, but it is possible to instantiate an arbitrary number of attackers if one wishes to simulate multiple scenarios at once or if a scenarios' attack chain requires more than one attacker server.

Firewall This is the companies internet zone facing firewall it provides both internet and internal routing for the company networks. In addition to being the intranet facing firewall, the server also acts as the internal DNS server used to resolve all company owned DNS domains. For the firewall functions Shorewall[3] is used and for DNS Dnsmasq and MaraDNS are used; the same as the *Global DNS* discussed above.

VPN The Virtual Private Network (VPN) server is configured with OpenVPN[4] to provide simulated home office employees access to the company network. Note that this VPN connection is provided via the default OpenVPN UDP port (1194) and made accessible to the internet through the Network Address Translation (NAT) configuration on the firewall.

Reverse Proxy The reverse proxy server acts as the central access point for all company web servers. All HTTP and Hyper Text Transfer Protocol Secure (HTTPS) traffic from both within the company network and the internet zone goes through the server is then reverse proxied to the correct web server using virtual host configurations. The internet accessibility is provided through DNAT of the common web ports 80 and 443.

Cloud Share This server is running an OwnCloud[5] instance and functions as cloud file storage and share for the simulated company employees. It contains various shared file directories accessible to different employee groups. It is also configured to provide access to some of the Samba file shares (see *File Shares* below). The cloud share can also be accessed through the internet via the reverse proxy.

Mail Server The mail server is the companies private mail server instance configured with Postfix[6] and Dovecot[7] to provide SMTP and IMAP functionality. The SMTP and IMAP ports of the mail server are available through the internet by use of DNAT. In addition to normal mail server configuration the server also has the Horde Webmail[8] service installed. This provides web based access (through the reverse proxy) to the

[3] https://shorewall.org/
[4] https://openvpn.net/
[5] https://owncloud.com/
[6] http://www.postfix.org/
[7] https://www.dovecot.org/
[8] https://www.horde.org/apps/webmail

e-mail functions for all simulated employees. Note that this is the same mail server configuration as used by Landauer et al. [Lan+20a].

File shares This server group is used to deploy various Samba[9] network file shares. Our concrete testbed instance is configured to contain exactly one such network file share which hosts various share directories containing sensitive information such as payroll charts, customer data and invoices. As mentioned above some of the share directories are also mounted and integrated with the cloud share server.

Intranet Server This is the companies internal news blog running WordPress[10]. In our testbed network configuration this is the only company web server which is not accessed through the reverse proxy, as it is only used by the simulated company employees and only accessible from the internal network zones.

Employee Workstations The testbed has two employee workstation groups, the *Employee Workstations* which are directly connected to the intranet and the *Remote Employee Workstations* which are part of the simulated internet zone. Both of these are essentially configured the same. They contain all the necessary software and configuration to run our user simulations (explained in detail in section 3.2). The *Remote Employee Workstations* group is used to simulate the presence of employees working from home (i.e., home office). These simulated employees can connect to the companies internal network using the above discussed VPN server.

External Users Workstations Identical in software configuration to the above discussed employee workstations the *External Users Workstations* are used to simulate users which interact with the infrastructure or employees of the company. In our testbed instance this is done solely through email communication between company employees and these external users.

External Mail Servers This server group is used for the mail servers used by the external users mentioned above. The number of mail servers as well as their used domains can be configured. In our testbed instance we have configured three mail server instances each managing a single mail domain. The software configuration of the mail servers is identical to the configuration of the simulated companies internal mail server.

3.2 User Simulation

In this section we will discuss the user simulation used to generate normal traffic within our evaluation scenarios. Below we will first give a general overview of the user simulation agent utilized in the scenario testbeds. Then we will further elaborate on its various sub

[9] https://www.samba.org/
[10] https://wordpress.com/

activities before finally discussing the different user profiles, resulting from configuration variations of the agent.

The user simulation agent discussed in this section is based on a state machine definition and implemented using the simulation library developed as part of this work. In this section we will mainly discuss the constructed state machine, its states and transitions. Discussions on how our simulation library can be utilized to actual implement such state machines while also enabling structured logging for the resulting simulation agents are discussed in section 5.2.

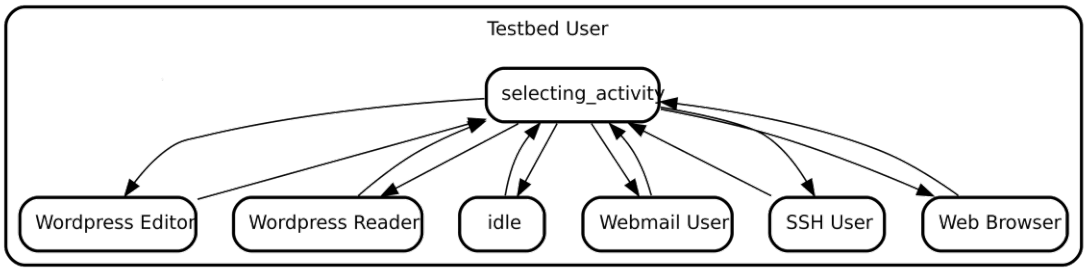

Figure 3.2: User Simulation State machine - Overview

Figure 3.2 shows a simplified overview of the state machine and its activities. Note that in the context of our user simulation state machines we often use the term *activity*. In this context we define it to refer to a sub graph of the complete state machine graph that encapsulates a specific user process e.g., using the webmail application. It is also to note that an activity can have an arbitrary number of sub activities (e.g., managing/viewing the calendar).

As shown in Figure 3.2 our state machine has a total of six main activities, each activity being modeled to simulate user interactions with a specific application. The only exception to this is the *idle* activity. As its name suggests if the state machine enters this activity state it will simply idle for a certain amount of time. This idle activity simulates two types of human behavior. First of, it simulates a human taking a break, i.e., a person leaving their computer and not interacting with any of the computer systems at all. Second, it is also used to simulate all actions a human might take during the course the day which are not observable within the scope of the testbed, e.g., using ones smart phone which is not connected to the companies network. The idle state thus represents all user actions outside the scope of the testbed as well as inaction. We choose this definition for our idle state to reduce the complexity of the user model. A more involved state machine user model might define multiple idle states for various unobservable actions. This would make it possible to define idle amount, activity entry probabilities and conditions for each of the idle activities separately, e.g., lunch break once a day for an hour and smoke breaks every other hour for a few minutes.

Below we will discuss each of the five activities in detail, but before that we will briefly explain few things on how we modeled our state machines and denoted them. Our user

simulation state machine can be mostly interpreted as a probabilistic state machine, but it does not limit it self to this construct. Most of the time the states shown in the diagrams below will be probabilistic states, i.e., each of their outgoing transitions has been assigned a certain probability.

There are two things to note in regards to this. One we usually do not show probabilities assigned to the transitions, because the shown state machine graphs are just a model and probabilities are only assigned to instances of this model, i.e., there might be two instances of our user simulation model with different probabilities assignments. Each of the instances would then represent a different user type, e.g., one user might be more likely to take breaks or another might always make sure to properly invalidate their web sessions.

Second the assigned probabilities might change in response to one or multiple transitions, limitations imposed by the state machine configuration (e.g., a daily activity entry limit can be set for each activity) or due to the observed state of the application that is being instrumented during the simulation process. For example, within our web mail activity (shown in fig. A.1) there exists an *open_mail* transition. When this transition is invoked a random e-mail in the users inbox will be opened. This is obviously only possible if the inbox contains at least one e-mail. As such the state is implemented to check the number of available e-mails and disable the transition, by setting its probability to 0, if there are no e-mails. Similar *sanity checks* have been implemented for various transitions.

Activity entry/exit states also are also modeled in a special way. All of the simulation activities, except the SSH user activity, are modeled as sub state machines without a final state. Meaning without an exit transition, returning from the activity to the main state machine graph, the user simulation process would never be able to stop or leave an activity. In real world scenarios a person might have many reasons to stop the activity they are currently engaging in. They might have achieved the goal they set out to do, e.g., checking and responding to their e-mails, they might have been interrupted by another person or another more urgent task that has come up. While for example the work of Kielar et al. [Kie+14] show that it is possible to simulate these intricacies of the human behavior and decision making process to some extent, we choose to keep the approach we use in this work more simplistic.

In our model each activity and sub activity has a main state. The main state is both the entry and exit point for an activity. Note that for sub activities we usually denote the exit transition with the keyword *return*. It also will have transitions for each of the activities direct descendant activities and actions. The exit transition is assigned a certain base probability which increases overtime. Specifically each time the simulation process leaves the main state through any transition other than the exit transition the current exit probability is increased by multiplying it with a configurable increase factor. This means that in our simplistic model a simulated user can only stop an activity from its main state and the chance of them doing so increases with time. We choose this type of model since it allows us to easily simulate users that stick with one activity for a

certain amount of time before changing to another task, e.g., a user might organize their calendar for a few minutes before then answering there emails.

3.2.1 Web mail

Figure 3.3 shows a simplified state machine of the web mail user activity. The simplified figure abstracts all sub activities to a simple state. In addition to the web mail activities' state machine graph the figure also shows the main activity selection state and the idle state. This is done to better illustrate the activity exit transitions and their interaction with the overall user simulation state machine. The full state machine graph for the web mail activity is shown in fig. A.1.

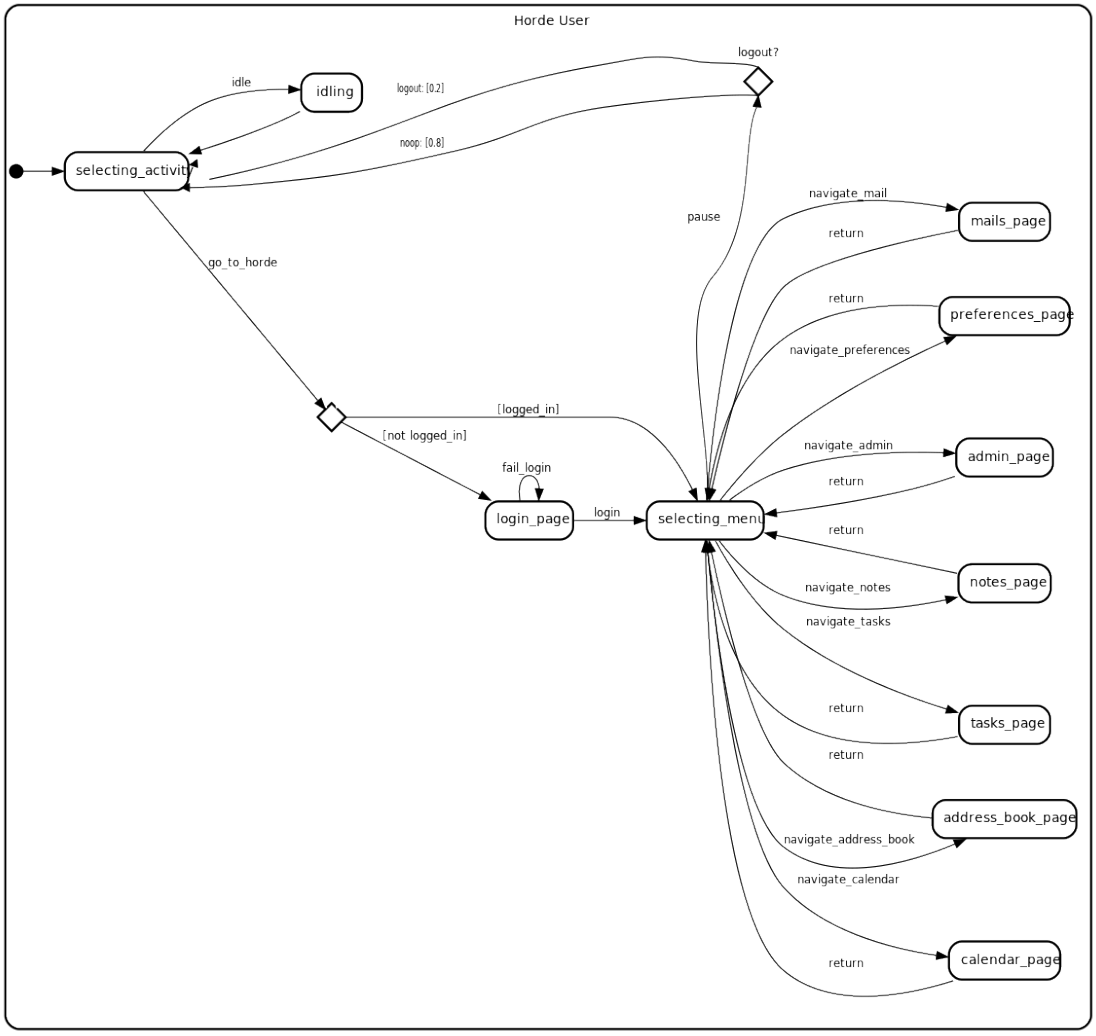

Figure 3.3: Horde Webmail Activity - Overview

The web mail activity for our user simulation is a re-implementation of the Horde group ware web mail user presented by Landauer et al. [Lan+20b]. The re-implementation was mostly done due to the adoption of the *Kyoushi Simulation* library, but in the process we also extended and improved upon the original state machine definition. For example the mails page sub activity was extended with additional states and transitions allowing for more varied user behavior. It is now possible for a simulated user to achieve the action of deleting or replying to an email both directly from their inbox or by opening the email in a pop up window.

The sub activities shown fig. 3.3 each correspond to one of the Horde web mails main menus. Each of the menus serves a different objective which allowed us to directly model them as separate sub activities. Note that as mentioned above each of the web mail users sub activities is modeled such that its activity exit probability increases with each transition from the sub activity main state. This behavior is different from the original state machine definition by Landauer et al. [Lan+20b] where the user returned to the state machines root state after executing any action (e.g., sending an email). Thus it was more likely that a user would rapidly switch between the various main menus. With the new definition a user is more likely to stay on a menu for a longer time before changing to another. For example the simulated user might first check their emails, open, delete, reply to and write new emails for a while before changing to the calendar menu and managing their events.

Another noteworthy change to the original state machine model is the configuration of attachments and email contacts. Previously the possible email attachments and contacts were simply configured as lists. That is each contact or attachment always had an equal chance of being selected. Using the configuration features of the *Kyoushi Simulation* library the new implementation now allows us to configure attachments and contacts with probability distributions. This makes it possible to create more realistic user simulation instances. For example this simple change makes it possible to model simple relationships between the various simulated users (i.e., by making it more likely for them to communicate with each other).

3.2.2 Cloud Share

Figure 3.4 shows the state machine graph for the cloud share user activity. This activity simulates user actions on a Owncloud cloud share instance. This such a share allows users to save their files and access them via a web interface or local application. In our setup we simulate the access via the web interface. Using a cloud share users can also cooperate with each other and share files. This is for example commonly done via share folders accessible to a user group or by sharing specific files or directories with other users.

While modern cloud share platforms do not only allow the sharing or accessing of files, but also often provide other features such as calendars or task list, our state machine limits itself to the file share features at the moment. Other features might be simulated

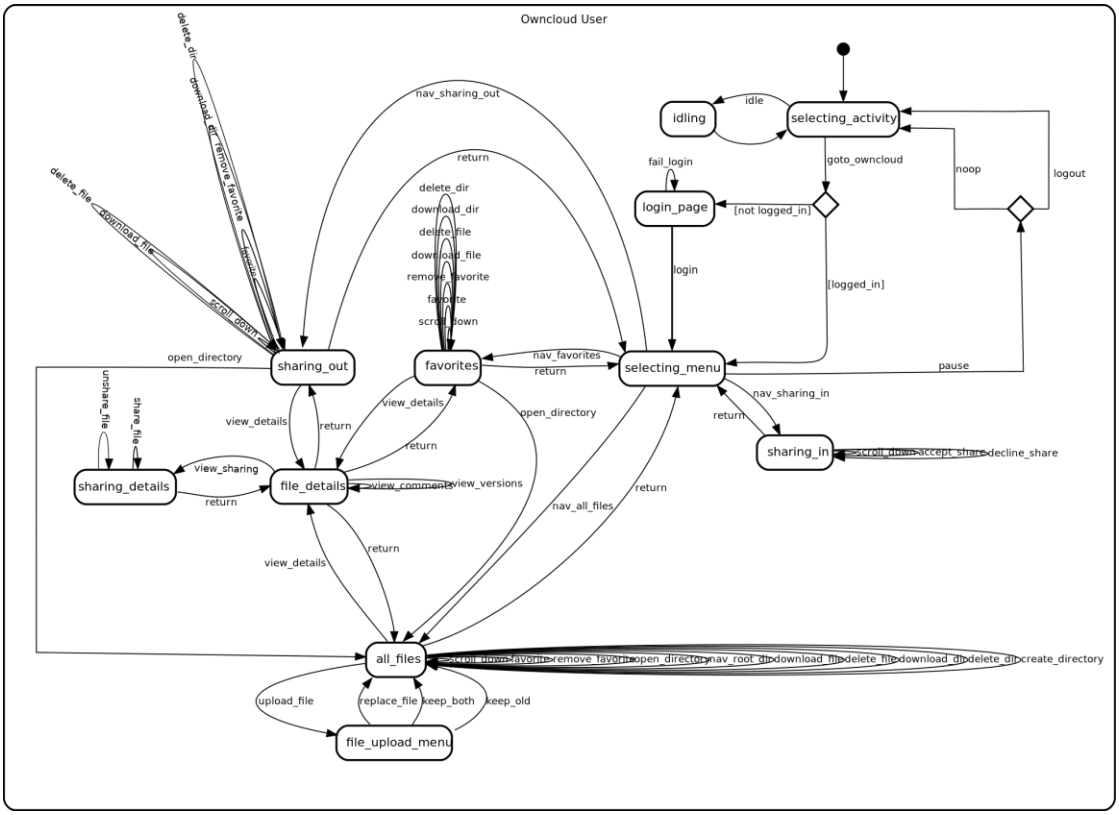

Figure 3.4: Owncloud File Share Activity

in future work. Four main file menu states have been defined to simulate the interaction with the file share.

- **All files** the main file menu showing *all files* a user has access to.
- **Favorites** a file menu only showing files or directories marked as favorites by the user.
- **Sharing-Out** a file menu listing all the files or directories the user has shared with other users.
- **Sharing-In** a menu showing all files or directories that other users have shared or wish to share with the user.

As can be seen in fig. 3.4 all of the file menus other than *Sharing-In* share a common set of transitions for the various file actions as the actions can be executed from either of their respective Owncloud menus. It is to note though that uploading files or creating directories is only possible on the *All files* menu. Also opening a directory will always

result in the state machine transitioning to the *All files* state. This reflects the Owncloud web interfaces behavior as the menu is automatically changed when opening a directory. The *Sharing-In* state does not have the same transitions as the other file menu states as it is purely used by users to accept or reject incoming file or directory shares.

It is also to note that every file menu (including *Sharing-In*) has a scroll down transition. As its name suggest this transition is used to scroll the current web browser view port further down. This user behavior was specifically modeled as a transition, because Owncloud employs a lazy loading approach on its file menus. That is the file lists are only loaded to fill the currently visible space. Should a directory hold more files or sub directories then they are only loaded once the user scrolls past the already loaded elements. The load operations for this are done via background HTTP requests and are thus observable in server logs.

Which files a user might upload and which users they might share files to is done similar to how email attachment and contacts are configured for the web mail activity. That is each uploadable file or user is assigned a probability allowing us to model relation ships with users and files or file types a user prefers to work with.

The state machine respects the file permissions configured on the Owncloud instance, i.e., the user will not try invoke an action which are restricted in the UI. Such actions would lead to transition errors and are thus disabled by parsing and checking the Domain Object Model (DOM) of the loaded view. In addition to these sanity checks it is also possible to configure additional restrictions on file or directory deletions. This can be useful for simulations when you wish to ensure that a certain directory structure will be maintained or file important for the simulated attack scenario are not accidentally deleted even when the user has the necessary access rights.

3.2.3 SSH User

Figure 3.5 shows the SSH user activity. As already mentioned above in contrast most of the other activities, the SSH user activity is modeled as a fine processes with an end state. A single execution of this activity can be interpreted as an SSH user connecting to a server, executing some commands and then disconnecting again. Then end state can be entered once all or at least a certain number of commands have been executed. While fig. 3.5 shows that the *disconnect* transition leads to a *final* state this does not mean that the user simulation would terminate at this point. In the context of the complete user simulation state machine this only indicates that the SSH activity ends and the simulation state returns to the *selecting activity* state.

As one can see the state machine model does not contain any specific command execution transitions and states, instead there only exists three generic states facilitating the execution of arbitrary commands:

- **Executing chain** state used to execute a sequential chain of commands. This state can only be left through the *finished* transition if all commands in the chain

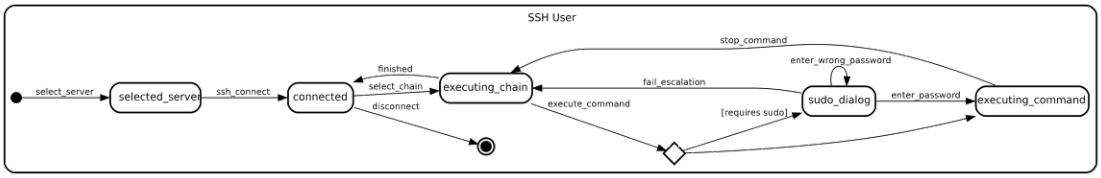

Figure 3.5: SSH User Activity

have been executed.

- **Sudo dialog** a special state used to handle privilege escalation. This state is only entered if the next command is executed with sudo and requires a password prompt.
- **Executing command** state in which a single command of a chain is executed until the expected output is observed.

Note that for the SSH User activity it is possible to configure a list of command chains for each SSH host separately as well as command chains that are used on all servers. This allows us to easily model both user behavior that might occur on all servers such as checking ones home directory and user behavior that is specific to a particular server environment, e.g., an admin checking the mail server logs.

A command chain can be configured as a single command or as a chain of sequentially executed commands. For each command it is also possible to configure an expected output regular expression used to check if the command has finished execution, by default a regular expression for the shell prompt is used. Note that some commands do not finish execution without further user input. Such commands can be modeled as command chains instead.

```
- - cmd: tail -F /var/log/syslog
    sudo: yes
    # we just wait for something to show
    expect: '.*'
    idle_after: medium
  - "\x03" # sigint

- - cmd: top
    expect: "avail Mem"
    idle_after: medium
- "\x03" # sigint
```

Listing 3.1: SSH command chains configuration snippet

For example listing 3.1 shows the configuration of two command chains used to execute the *tail*[11] command with the follow option active and the *top*[12] command. Both of these commands would run forever without user intervention. In the configuration one can see that each of the command chains contains the string `"\x03"` at the end. This is the *SIGINT* signal (CTRL+C) encoded as hexadecimal string. That means if one of these two command chains is selected for execution the state machine will execute the actual command first (i.e., top or tail) and then wait a *medium* amount of time before sending the interrupt signal and thus stopping command execution.

3.2.4 Internet Browser

Figure 3.6 shows the state machine graph for the web browser user activity. This user activity is arguably the simplest activity in our user simulation model. It consists of only two states and five transitions. This activity is used to simulate users accessing the world wide web.

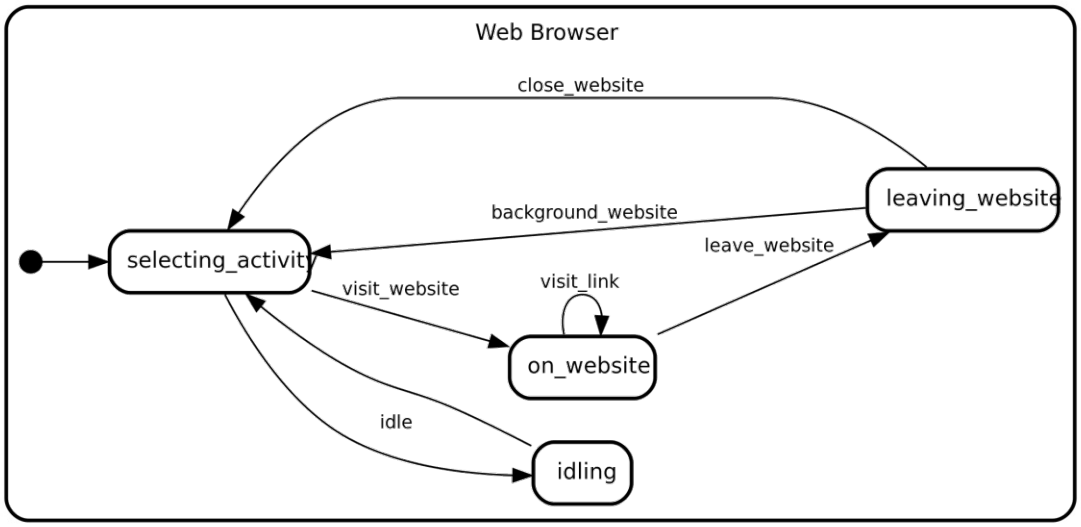

Figure 3.6: Web Browser Activity

Upon entering this activity a random URL is selected and visited (the *visit website* transition). Once the *on website* state is entered the user simulation will parse the websites DOM and compile a list of links. Here our state machine behaves very similar to a basic web crawler [Mir+14; SG15]. Next the state machine will either visit one of the collected links (*visit link* transition) or leave the website (*leave website* transition). It is possible to configure a maximum link depth, i.e., amount of times *visit link* can be selected consecutively.

[11] https://man7.org/linux/man-pages/man1/tail.1.html
[12] https://man7.org/linux/man-pages/man1/top.1.html

Note that our chosen web crawler like approach is very prone to errors, e.g., our web browser instrumentation might run into errors when trying to click on a link collected from the DOM which is not clickable. As such we implemented the activity in such a way that it will fall back to the *selecting activity* state if it encounters *any* error during simulation.

Also it is possible to exit the activity two different ways either by leaving the last opened web site or link open (*background website* transition) or by closing the website (*close website*). These two options were implemented, because some websites contain Javascript code that regularly updates content or polls information. This means that when the user simulation leaves the web browser activity by leaving the web site open and then enters for example the idle state we will still be able to observe these content updates and polls in our network logs (e.g., DNS, suricata).

The list of website links to use for the initial websites visits was taken from the *Top 500 Most Popular Websites* [Moz]. Also half of the website URLs were configured to use HTTP instead of the encrypted HTTPS. This was done to have a variance of both encrypted and unencrypted web browsing traffic. It is to note though that the amount of initial website visits via HTTP will be lower than the number of configured HTTP URLs. This is due to many of the websites being in the HTTP Strict Transport Security (HSTS) preload list which ensures a website is always accessed using an encrypted connection even when the user has never visited it before [HJB12; DL17; Hst].

3.2.5 Wordpress Publisher

Figure 3.7 shows the Wordpress publisher activity state machine model. This activity is modeled to simulate the behavior of a Wordpress user with editor privileges i.e., a user that is able to write, edit and publish posts on a Wordpress instance and also other related editor actions.

This activity has four sub activities of which two are do not have any outgoing transition other then the exit transition; referred to as *noop* (No Operation) in the figure. The *dashboard* and *media* states simply map to the similarly named Wordpress menus. In the current implementation it is only possible to navigate to these menus and thus enter the states, but there are no further actions available. Future work might add more transitions to this activity such as uploading, deleting a media file.

If the state machine enters the *post* state then there are only two possible transitions. Either it returns to the *selecting menu* state or it starts the process of creating a new Wordpress post. The *comments* page is used to simulate an editor reacting and responding to reader comments. When the states exit transition is not chosen, a random reader comment is selected and replied to.

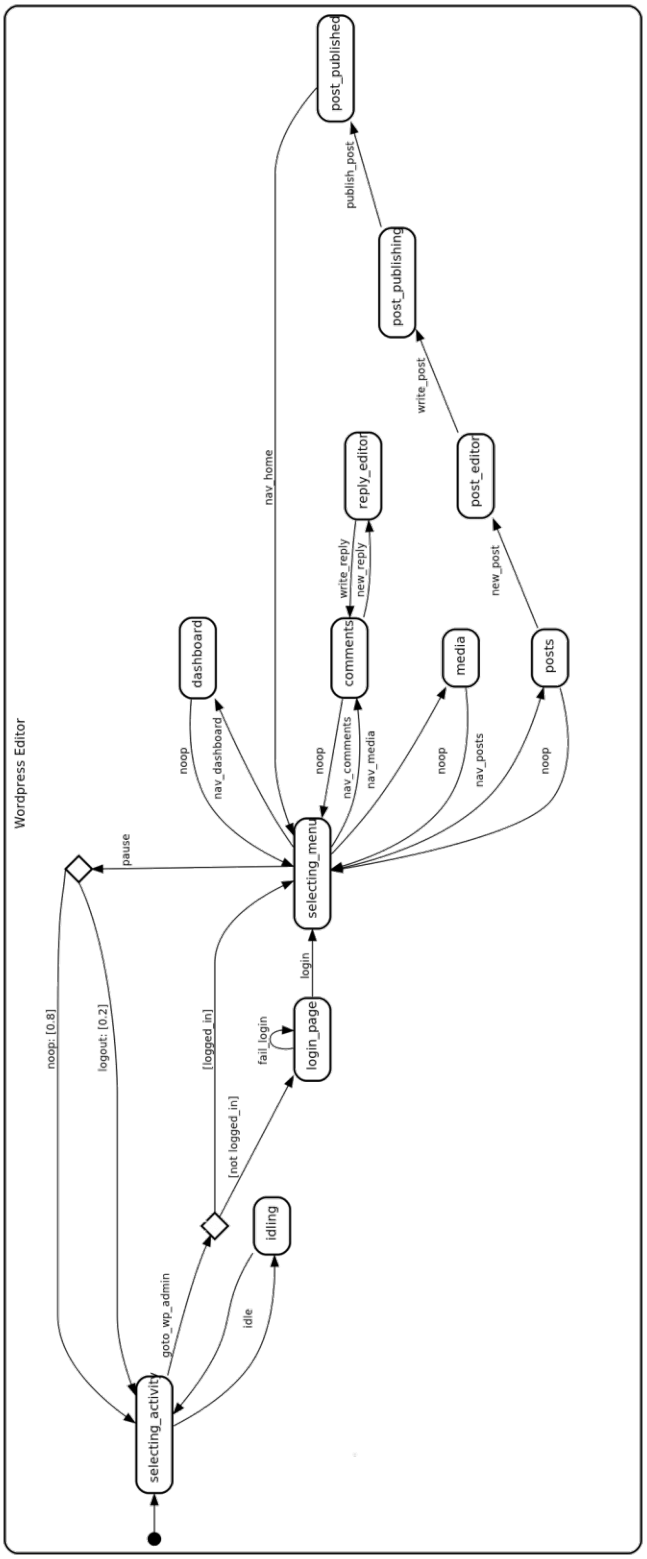

Figure 3.7: WordPress Publisher Activity

3.2.6 Wordpress Reader

Figure 3.8 shows the Wordpress reader activity state machine graph. This activity is so to say the counter part to the above discussed Wordpress publisher activity. It simulates a normal unprivileged user that simply consumes content on the Wordpress instance and maybe writes a few comments.

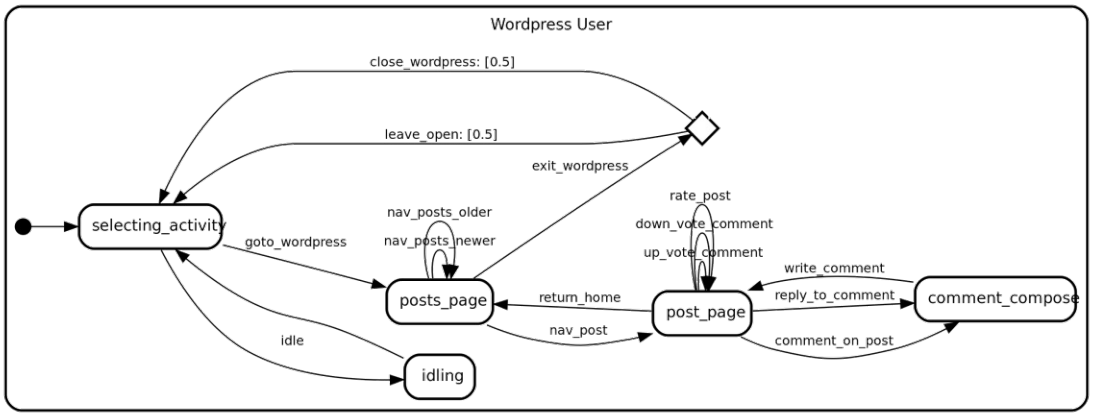

Figure 3.8: WordPress Reader Activity

Within the activity model there are two pages the simulated user might visit the *posts page* and the *post page* (mind the plural **s**). The *posts page* is the main view for most Wordpress blog themes it shows the latests posts and you can navigate to newer or older posts. The *post page* is the view for a single post it shows the full post text and also gives the user the option write or reply to comments, rate the post and up or down vote other users comments.

3.2.7 User Profiles

Above we discussed our user simulation state machine model and all the activities defined within it. In this section we will discuss the user simulation profiles we defined to use with our attack scenario. For the sake of simplicity we use the same probability configurations for all our user simulation instances. We deemed this to be enough for our proof of concept simulations used only to verify the soundness of our data set generation and labeling approach. Future work using our approach to generate IDS datasets might define the transition probabilities based on statistical analysis of log data collected from real world systems.

As such when we use the term user profile we mainly refer to user simulation instances using the same underlying basic transition probabilities, but having only certain activities enabled. Note that each simulated user also has their own configuration for things like usernames, password, etc. which *must* be unique. In our simulation we have four user profiles and a special profile modifier which are described below.

Normal Employee The normal employee profile represents a *normal* employee user, i.e., a user that has no special privileges on any of the observable systems. This user profile has the Web mail, Wordpress reader, web browser and idle activities enabled.

Publisher Employee This user profile simulates the group of users that have been granted reporter privileges on the intranet server instance. These employees are tasked with providing content and internal news via the simulated companies blog. They have all the same activties as the normal employee profile, but instead of the Wordpress reader activity they use the Wordpress publisher activity.

SSH Admin The SSH admin user profile is used to simulate the companies internal system administration teams. In addition to having all normal employee activities they also have the SSH user activity enabled. The activity configured to simulate various daily system administration tasks. Note that each admin user has a different SSH user configuration. Therefore the servers they connect to and commands they execute might be different.

External User This user profile is used for all external user simulation. As already mentioned in section 3.1 external users in our testbed are only used to simulate email traffic between company employees and external entities. Thus they are configured to only use the web mail and idle activities.

Remote Employee The remote employee user profile is not a complete user profile in itself. It simply modifies the three employee profiles and enables the usage of the VPN for them. Additionally we also have defined two different types of remote employees *lazy* and *eager* remote employees. A *lazy* remote employee will only open the VPN connection once they actual need them, i.e., the moment they try to do an activity that requires a resource that can only be accessed from within the companies internal network (e.g., the intranet server). The *eager* remote employees on the other hand will open their VPN connection as soon as they start work for the day.

Table 3.1 shows an overview of the user profiles assigned to the various simulated employees. ✓ indicates that the employee is a remote employee while the ✗ indicates the inverse. Note that external users are omitted from the table as they all are assigned the external user profile.

Hostname	Name	Profile	Remote
internal_employee_0	Konstantin Braun	Normal Employee	✗
internal_employee_1	Lewis Wood	Publisher Employee	✗
internal_employee_2	Stefanie Nowak	Publisher Employee	✗
internal_employee_3	Helena Kern	Normal Employee	✗
internal_employee_4	Enna Geier	Normal Employee	✗
internal_employee_5	Naomi Leithner	Normal Employee	✗
internal_employee_6	Marius Sperl	SSH Admin	✗
remote_employee_0	Elliot Fletcher	Normal Employee	✓
remote_employee_1	Emmanuel Burger	Normal Employee	✓
remote_employee_2	Felicitas Feichtner	SSH Admin	✓

Table 3.1: Employee User Profiles - Overview

CHAPTER 4

Attack Scenarios

In the previous above we discussed the configuration of our cyber range testbed and the user simulation model utilized within it. Below we will describe two separate proof of concept attack scenarios that we will use for creating labeled HIDS datasets using the CR Kyoushi framework. For each of our attack scenarios we will not only describe its various attack steps, but also the reasoning behind why we choose them for our framework evaluation. Note that the labeling framework configuration and its evaluation are discussed in chapter 7.

4.1 Server Takeover

Figure 4.1 shows a state machine graph representation of our first attack scenario. Throughout the remainder of this work we will often refer to this particular scenario as *scenario 1* or the *server takeover* scenario. In this section we will also sometimes refer to it simply as *the scenario*.

The server takeover scenario models an attacker that has obtained VPN credentials for a companies internal network and then proceeds to scan and attack the companies systems using this access point. While recent years have seen a strong interest in IT-security community to adopt zero trust security models over legacy perimeter security architectures, adoption of such architectures is still fairly limited and VPNs are still widely used. As discussed by Arkko et al. [Ark+21] and Ahmad [Ahm20] due to the recent COVID-19 pandemic VPN related traffic observed on the internet has increased drastically. With the increased numbers of remote workers and the usage of VPNs it is important for an IDS to be able to detect potentially malicious VPN user activity. For example a sudden change in the users behavior, like connecting outside their work hours or connecting from a new IP address. For this scenario we consider the actual acquisition of the VPN credentials to be out of scope for the generated dataset. In a real world situation this could happen in various ways e.g., through a phishing attack or a

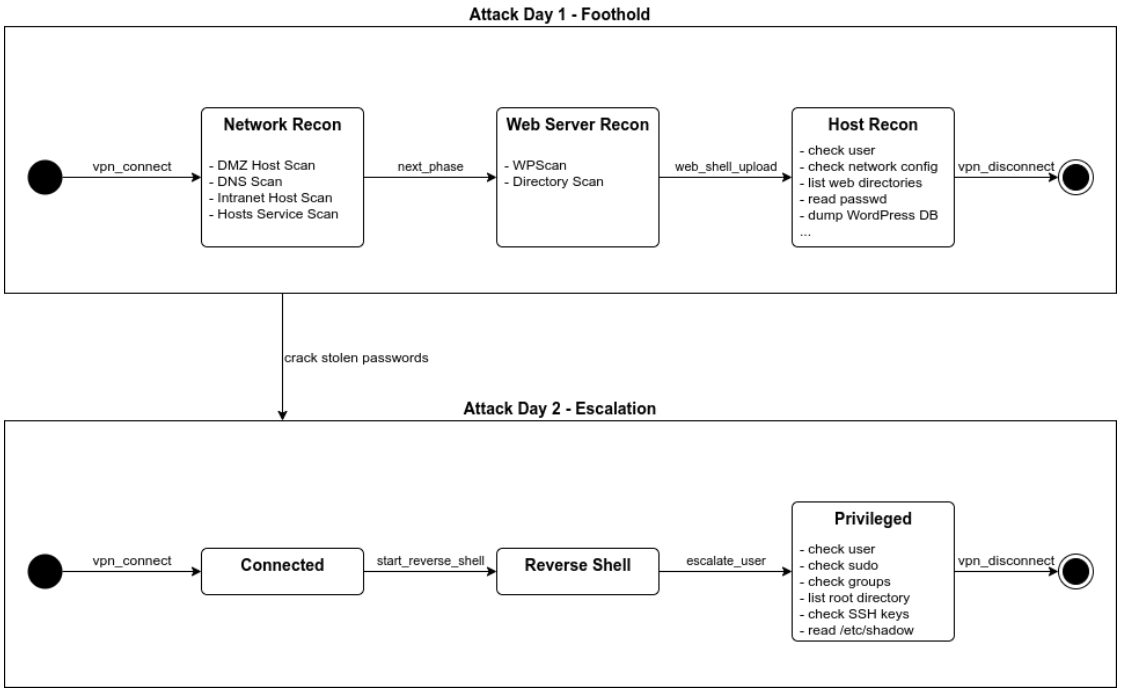

Figure 4.1: Scenario 1 - Server Takeover

compromised personal computer of an employee. Many of these possible scenarios would not be observable by a companies Intrusion Detection System (IDS), since they occur outside the companies network. As such we decided to omit part of the attack scenario from the dataset[Ark+21; Ahm20; Lia+16; Ros+20; Emb20].

In our attack scenario model the attack is split into two phases: *Foothold* and *Escalation*. As their name suggests the *Foothold* phase concerns the attackers initial breach of the companies network and compromise of an internal system. The *Escalation* phase then leverages the access achieved and information gathered in the *Foothold* phase to gain elevated privileges on the compromised system.

In fig. 4.1 some of the states are shown with bullet lists of actions. These actions indicate the steps that the attacker must execute before they can transition to the next state. As such they can be interpreted as self transitions (i.e., transitions that start and end in the same state). The attack steps shown as part of the states are executed in a random order, but will respect prerequisites of steps. For example for the *Privileged* state shown in fig. 4.1 the action *read /etc/shadow* will only be executed once the *check sudo* action has been executed. This controlled random execution is achieved through our Kyoushi Simulation library. Using our library the states have been implemented as tiered random queues. The tiered random queues are defined such that the next action is randomly selected from the upper most tier. Each action can have an arbitrary number of sub-actions (lower tier). Every time an action has been executed all its actions are

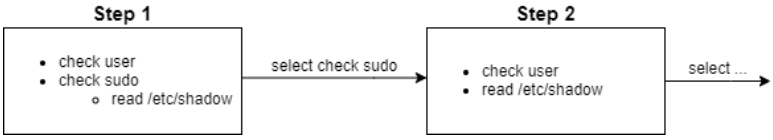

Figure 4.2: Tiered Queue - Example Steps

moved to the upper most tier. This process is repeated until the queue is empty at which point the state machine will transition to the next state in the attack chain. Figure 4.2 illustrates this queue process.

4.1.1 Foothold

The foothold phase is split into three states each of which is modeled to represent a specific part of the phases attack chain. The states and their attack steps are described below.

Network Recon. In this state the attacker is trying to get an initial picture of the network topology and systems present within it. For this they mainly utilize the commonly used network scanner nmap[1]. First the attacker will gather information about the DMZ network to which they are connected via the VPN (fig. 3.1 shows the network topology). For this they will execute both a network host scan and a DNS brute force scan. The DNS scan allows them to discover the presence and CIDR of the intranet network segment. A network host scan is then also executed for this segment. Finally the attacker executes a full host service scan on all discovered hosts to identify potential targets for exploitation.

Web Server Recon. After completing the network reconnaissance the attacker selects the intranet server as target for further information gathering. In this part of the attack chain they will execute a few targeted web service scans, as they were able to detect a Wordpress instance running on this server. Specifically two web services scans are executed. A Wordpress specific security scan using the tool *WPScan*[2], which allows the attacker to identify potentially vulnerably Wordpress versions, plugins, themes, etc. Second the attacker will also execute a generic web server directory brute force scan using the tool *dirb*[3]. These types of directory scans are often used by attackers to find potentially interesting directories or files on web servers.

Host Recon. In this attack state the attacker gains initial access to the previously scanned intranet server and performs reconnaissance on the system. During the Wordpress scan the attacker discovered that the version of the installed plugin *wpDiscuz*[4] has a

[1] https://nmap.org/
[2] https://wpscan.com/wordpress-security-scanner
[3] https://tools.kali.org/web-applications/dirb
[4] https://wordpress.org/plugins/wpdiscuz/

known Common Vulnerabilities and Exposures (CVE). Specifically *CVE-2020-24186* which is a unrestricted file upload vulnerability. Using this vulnerability the attacker is able to upload a PHP file. They upload a PHP web shell giving them the ability to execute arbitrary code and commands on the intranet server with the privileges of the web server user *www-data*. Using the web shell the attacker executes various host reconnaissance commands such as reading the Wordpress configuration, which allows them to obtain the database password and then dumping the whole Wordpress database. This includes the user table containing usernames and hashed user passwords [Owa].

4.1.2 Escalation

In the escalation phase the attacker utilizes the access and information gained in the previous phase to compromise the intranet server further and pivot from the web server user account to a user with higher privileges. During the previous phase the attacker was able to dump the Wordpress servers database and thus gather a list of usernames and password hashes. Within our scenario the attacker tries to crack these password hashes during the idle time between the two attack phases. They are able to crack the password for one the Wordpress admin users who also happens to have a system user account on the server. These user credentials will be used to escalate the attackers privileges. Note that no actual password cracking takes place this is purely simulated by the downtime between the two attack phases.

After cracking the hashed user password the attacker will again connect to the VPN and use the previously deploye web shell to establish a fully interactive reverse shell connection to their attacker server. This reverse shell allows them to actively interact with the command shell of the compromised system and in contrast to the web shell allows the attacker to directly execute interactive commands such as password prompts. Using the reverse shell connection the attacker will try to login to the user account for which they were able to recover the password.

Finally the attacker will use their newly obtained privileges to execute further commands on the intranet server host in hopes to find information that will help them to gain even more access to systems within the companies networks. The executed commands are fully configurable in our attacker model, but for our proof of concept scenario we limit us to a few simple commands which should be detected by an Intrusion Detection System (IDS). The attacker will for example inspect the root directory, search for SSH keys and read the systems */etc/shadow* file, which contains the hashed user passwords.

4.2 Data Ex-filtration

Below we discuss our second proof of concept attack scenario. For the remainder of the work we will refer to this scenario to as either *scenario 2* or the *data ex-filtration* scenario. Also as already done in the section above we will also refer to the data ex-filtration scenario as *the scenario* in this section.

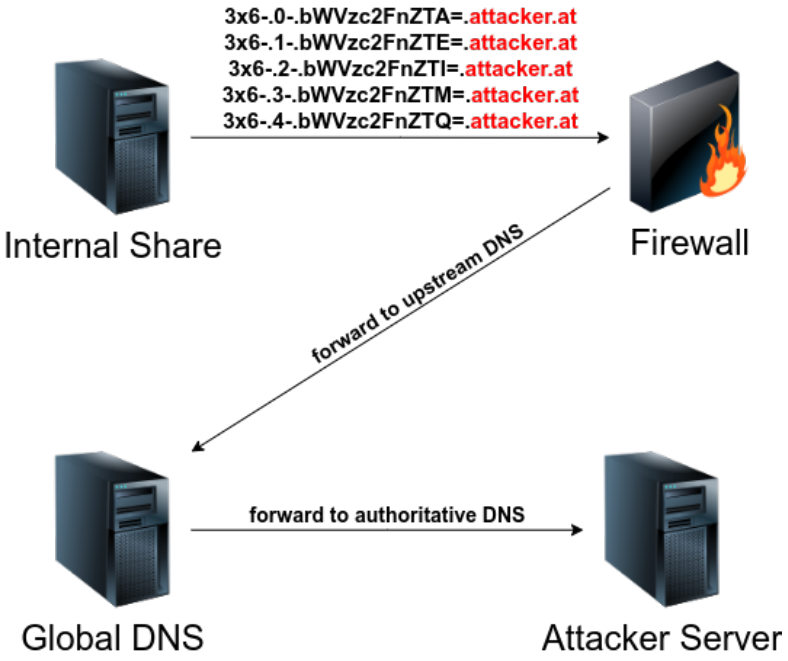

Figure 4.3: Scenario 2 - Data Exfiltration

The scenario is modeled as a data ex-filtration incident, i.e., an attacker has already compromised at least one of the company's systems and is now trying to transfer sensitive information from this system to their own servers. Data ex-filtration is often a central part of Advanced Presistent Threat (APT) campaigns that slowly take over a network and try to gather as much valuable information as possible. To this end they also often rely on covert channels to camouflage the traffic generated during data ex-filtration from the Intrusion Detection System (IDS). For this attackers have historically used different techniques, including limiting the speed at which data is ex-filtrated or sending the data via alternate inconspicuous protocols such as DNS [Col12; Ull+18; NAS19; DCY16].

Our data ex-filtration scenario assumes an attacker has already compromised the simulated companies internal Samba share and gained full system privileges on the server. After gaining access the attacker then registers a malicious service process that reads and tries to ex-filtrate all files from the servers Samba share directories. Within our scenario we assume that the above described process precedes the datasets observation window. The data ex-filtration process will then stop after a few days once all data has been ex-filtrated. This provides a special challenge to intrusion detection systems, especially those that use anomaly detection, as our malicious traffic could now be considered normal system behavior. The anomaly that would have to be detected would be sudden absence of the DNS traffic once the data ex-filtration has finished [Ahm+20; NAS19; Ull+18].

Figure 4.3 shows the DNS traffic flow for our data ex-filtration process. On the internal share server the malicious service process reads one file at a time. Each file is then

encoded into *base64* to conform with to the limited character set allowed for DNS queries. The *base64* encoded file is then split into parts and sent to the attacker server through the internal shares configured DNS server by creating a special DNS request for a attacker controlled domain. In our testbed configuration this is the companies firewall which forwards all requests for unkown domains to an upstream DNS server located in our simulated internet. This global DNS then finally forwards the requests to the attacker controlled server which is configured as the authoritative DNS server for the ex-filtration domain (in the figure the domain would be *attacker.at*).

On the attacker server, incoming DNS requests are accepted and decoded by a modified version of the open source data ex-filtration tool DNSteal[5]. We modified the original code[6] in two ways. One we added the option of configuring an attacker controlled domain to allow such indirect data ex-filtration as described above and shown in fig. 4.3. The original tool was only able to decode DNS requests sent directly to it. Second we implemented structural logging for the tool. We did this to make the resulting server logs easier to parse during our labeling process. This is especially important as in this scenario we do not have an attacker state machine and the DNSteal logs and network traffic observed on the attacker host are our only indicators for the malicious activity. The ex-filtration service on the internal share does not log or otherwise record attack information as this would taint the internal shares own logs.

Other than being able to configure on which server and which files the ex-filtration service is executed it is possible to configure the speed of the ex-filtration. The speed is controlled by three variables:

- **S:** This option defines how many sub domains are used to store data per DNS query.
- **B:** The number of bytes to send per sub domain.
- **Interval:** The time to wait in-between DNS requests. This is configured with minimum and maximum sleep values.

Note that a single DNS request can at most be 255 bytes long. This gives us an upper limit for how much data can be sent per request. The data will also be limited depending on other factors that take up space in the DNS requests such as the ex-filtration domain, file names and data indices, etc. It is also possible send data in a compressed format which both reduces data size and adds another level of encoding to the ex-filtration process.

[5] https://github.com/m57/dnsteal
[6] https://github.com/max-frank/dnsteal

CHAPTER 5

Cyber Range Kyoushi Framework

In chapter 2 we discussed state of the art works on the topics of Intrusion Detection System (IDS), anomaly detection and cyber range testbeds. Furthermore, in chapters 3 and 4 we discussed a cyber range testbed environment and two attack scenarios. Based on these discussions we present Cyber Range Kyoushi, a methodology and proof of concept implementation for the creation of labeled synthetic Host Intrusion Detection System (HIDS) datasets. For this we first formulate requirements for dataset generation and labeling systems. Then we discuss the overall architecture of Cyber Range Kyoushi and all its components. Finally we discuss each of the components and their implementations in depth. In chapter 6 we further discuss how we used Cyber Range Kyoushi to generate two labeled datasets for the previously discussed attack scenarios, while we discuss the evaluation of the resulting labels in chapter 7.

5.1 Architecture & Components

Cyber Range Kyoushi is built on previous work on the topic of cyber ranges and dataset generation using model-driven simulation testbeds by Landauer et al. [Lan+20b] and Leitner et al. [Lei+20]. The main use case for the methodology we propose is the generation and labeling of Host Intrusion Detection System (HIDS) datasets. The most common host intrusion detection data sources are text based log files or can otherwise be easily converted and represented to text based formats (e.g., Systemd Journal logs[1]). As such, the framework works processing and labeling components are designed to work with text based data formats. Other data formats must be converted to a text based representation during pre-processing.

[1] https://man7.org/linux/man-pages/man8/systemd-journald.service.8.html

5.1.1 Requirements

Gharib et al. [Gha+16] define a list of requirements for Intrusion Detection System (IDS) datasets, while their work has a focus on NIDS datasets, the requirements defined are largely applicable to HIDS datasets as well. Nevavuori and Kokkonen [NK19] also discuss requirements for Network and Host Intrusion Detection System (NHIDS) datasets and their generation in cyber range systems. Based on these works and other works, such as Ring et al. [Rin+17], Landauer et al. [Lan+20b], Buchanan et al. [Buc+21], and Sharafaldin et al. [Sha+18], providing or discussing IDS datasets we define requirements for a Host Intrusion Detection System (HIDS) dataset generation and labeling methodology.

R1 Reproducibility: For scientific work it is always important that it can be reproduced during the peer review process. This is of course also true for IDS datasets. The proposed methodology should therefore ensure that the dataset generation and labeling process are reproducible with reasonable resource expenditure. For example, usage of open source software or software that is otherwise free to use for academic purposes should be preferred over licensed software to make it possible for peers to reproduce datasets.

R2 Complex systems: Works like Sharafaldin et al. [Sha+18], Gharib et al. [Gha+16], and Nevavuori and Kokkonen [NK19] often stress the importance of realistic datasets. As Gharib et al. [Gha+16] mention in their work some attack scenarios are only possible in the presence of fully configured networks and systems. Thus it is vital for the proposed methodology to not only support the creation of such complex realistic systems, but also be able to cope with the complexities involved in generating and processing datasets using such systems.

R3 Benign interaction: An important part of any IDS dataset are the benign traffic and interaction records. A realistic system always contains a number of users interacting with various parts of the infrastructure producing both network traffic and various host logs. This is often referred to as benign traffic or interaction. Inclusion of such benign traffic is especially important for datasets, which are to be used for training or evaluating anomaly detection algorithms. That is, in absence of benign traffic and interaction attacks anomaly detection algorithms would not be able to properly learn the difference between normal behavior and anomalies (i.e., the attacks). A dataset generation and labeling methodology must thus include mechanisms to simulate realistic normal user behavior [CBK09; Gha+16; SM12; NK19].

R4 Multiple Attack Types: The cyber security attack and threat landscape is changing rapidly. A dataset generation and labeling methodology must therefore support a vast variety of attack scenarios and types. This is important to ensure that dataset generation efforts can keep up with the rapid changes of the attack and threat landscape. The methodology should also be able to generate data sets for various IDS algorithms, for example, there are various types of anomaly detection

techniques, such as frequency or correlation analysis. The proposed methodology and framework should thus be able to produce labeled datasets for evaluating and training such algorithms [Wur+18; Gha+16].

R5 Heterogeneous data sources: As discussed above the main focus of this work is the generation and labeling of HIDS datasets. Thus we mainly deal with text based log formats, but even with this limitation the proposed framework should still be able to handle and capture a wide variety of data sources and data formats. Furthermore the proposed framework should be able to parse unstructured data sources and covert them to well defined structured formats, which can be easily processed by automated systems. This structured data should then be made accessible to further dataset processing and labeling processes.

R6 Integration of facts: Gharib et al. [Gha+16] note the importance of including network and system configuration, attack scenario definitions and other metadata in datasets. They discuss that such information can largely contribute to the usability of datasets. For the purposes of this work we will refer to such information as *facts* (e.g., *testbed facts* or *scenario facts*). While Gharib et al. [Gha+16] only require the inclusion of such facts in the final datasets, we further require that *facts* are not only part of an artifact, but also available as sources of truth and data for the processing and labeling of dataset.

R7 Adaptable within pre-defined models: Landauer et al. [Lan+20b] describe a methodology for rapidly generating IDS datasets using a model-driven testbed approach. Their approach relies on defining testbed and attack scenario configurations as models with a few randomly assigned variables, to rapidly generate multiple different datasets. The dataset generation and labeling methodology we propose should be able to resist the changes between instances of such partially random generated testbeds and attack scenarios as described by Landauer et al. [Lan+20b]. That is, generating a new instance of a defined testbed and attack scenario model should not require re-configuration of our proposed framework. This means it should be possible to configure the framework in such a way that it is able to adapt to the changes between model instances automatically.

R8 Extensible Labeling Rules: Labeling techniques and approaches can vary vastly for different datasets and attack scenarios. For example, Ring et al. [Rin+17] use a four stage labeling approach for their network flow dataset, while Sharafaldin et al. [Sha+18] describe a two stage labeling process. Apart from the general approaches labeling techniques are usually also different depending on the attack scenario and testbed infrastructure. Ring et al. [Rin+17] label their dataset mainly by matching the source and destination IP addresses and ports with malicious activity as defined by their attack scenario model. In contrast Sharafaldin et al. [Sha+18], label their network flows based on the attack timings, i.e., network flows that occurred during the attack are labeled as such. The proposed methodology and framework should therefore be able to support a wide variety of labeling strategies and techniques,

for example, time based, similarity based and exact attribute matching. For this it should be possible to easily configure different approaches and utilize or easily implement various labeling techniques.

R9 Precise Labeling: The required precision of labels varies between use cases. For example, for a simple IDS benchmark dataset might be sufficient to provide two types of labels differentiation *malicious* and *normal* logs or traffic. For other use cases it might be beneficial to provide more precise labels that clearly indicate the specific attack that produced the logs or traffic. Such details can for example be useful for researchers to determine which attacks types can be correctly detected by their attack classification algorithms. As the proposed methodology and framework should be applicable to all use cases it must be possible to create very fine grained and precise labels with it.

5.1.2 System Architecture

Above we defined requirements for an IDS dataset generation and labeling methodology. Below we discuss our proposed framework Cyber Range Kyoushi. In this section we will discuss the framework's overall system architecture and how each of its components work together to create an IDS dataset generation and labeling methodology.

The generalized architecture of the *Cyber Range Kyoushi* framework, or short *CR Kyoushi*, is based on four system layers. Each layer represents a different stage of the dataset generation and labeling process supported by the CR Kyoushi frameworks components. Figure 3.2 depicts a visual representation of the four system layers and the interaction between them.

Figure 5.1: Cyber Range Kyoushi - System Layers

Model Layer

The model layer contains the datasets testbed and scenario configurations. That is, it formally defines and describes a datasets simulated environment (i.e., infrastructure and interactive agents). The layer is the single source of truth for the dataset. It contains all the configuration files and information necessary to reproduce (**R1**) and verify the dataset. Figure 5.1 shows three types of model configuration in particular, but depending on the dataset and simulated attack scenario definition there might be more.

- **User Model:** The user model defines the simulated normal user behavior (**R3**). Sharafaldin et al. [Sha+18] refer to this as *B-Profiles* (*Benign*) as it defines what and how the benign (i.e., normal) system interaction and traffic is generated. Note that our methodology does not define how user profiles are defined, i.e., how one can create realistic user profiles for various simulated interactions. This process depends on the applications and protocols present in the specific datasets testbed infrastructure, but generally it should be possible to extract realistic user models from traffic captures or system logs of real systems using statistical analysis as discussed by Sharafaldin et al. [Sha+18] and others.

- **Infrastructure Model:** The infrastructure model defines the cyber range testbed configuration for the dataset. That is, it defines both the networking and system configurations. The CR Kyoushi framework follows the methodology discussed by Landauer et al. [Lan+20b] and Leitner et al. [Lei+20] and uses state of the art infrastructure and software provisioning as code technologies to define the infrastructure model. Using these technologies makes it possible to automate most of the cyber range testbed deployment process, as they eliminate cumbersome manual software and network configuration steps. This also makes it possible to reliably orchestrate even complex network and system configurations as once defined the infrastructure model only needs to be applied (**R1 and R2**). Support for complex infrastructures is also a prerequisite for many different attack types that require specific network or system configurations (**R4**).

- **Attack Model:** While the user model is used to define the benign interactions within the testbed environment, the attack model defines all malicious activity. Sharafaldin et al. [Sha+18] also refer to this as *M-Profiles* (*Malicious*). The contents of the attack model highly depend on the use case of the dataset, i.e., what type or class of attacks should be simulated. Various attack models might also be used with the same user and infrastructure models. For example, in sections 3.1 and 3.2 we describe an infrastructure and user model. These two models are used in both of our attack scenarios discussed in chapter 4. Attack models can describe any type of attack and are only limited by capabilities of the software used to instrument them (**R4**).

Testbed Layer

The testbed layer is provided by both the cyber range testbed used to orchestrate the infrastructure model and the software used to simulate the defined user and attack models. The model and testbed layer are highly dependent on each other. As discussed by Landauer et al. [Lan+20b] and Leitner et al. [Lei+20], Cyber Range Kyoushi utilizes OpenStack[2], Terraform[3] and Ansible[4] as its cyber range testbed technology stack. There are many state of the art user or attacker simulation technologies available.

For example, Buchanan et al. [Buc+21] use the MITRE *CALDERA* framework's *human plugin* to simulate user traffic on attacker controlled devices within their datasets scenario. The CR Kyoushi framework implements a state machine based simulation library for this purpose. Using this state machine approach it is possible to model a wide variety of user and attacker behavior (**R3 and R4**).

Data Collection Layer

Generation of IDS datasets produces large amounts of data. This data is not limited to the traffic or system logs that will become part of the final dataset, but also includes facts about the cyber range testbed configuration (e.g., IP addresses, software configurations, etc.) and other network traffic and system logs of host machines or applications that are outside the scope of the dataset. Such machines might be required by the datasets scenario, e.g., for our two scenarios discussed in chapter 4 the final datasets only observe the simulated companies server machines.

All other machines (e.g., employee clients, external servers, etc.) interact with these servers in some form or the other, but would not be covered by an Intrusion Detection System (IDS). The logs and traffic for these systems are therefore not part of the labeled dataset. Nonetheless collecting and providing the facts and system logs of these machines as part of the dataset has some important benefits. As discussed with requirements **R5** and **R6** this additional information can contribute to the usability and verifiability of the dataset. Also it can be useful during dataset processing and labeling to make it possible to precisely label the dataset (**R9**).

Processing Layer

The processing layer addresses requirements **R5** to **R9**. On this layer the raw data (traffic captures, logs and facts) generated and collected by the previous layers is first converted into machine readable and processable formats and IDS relevant logs are then labeled.

The Cyber Range Kyoushi framework defines three components that enable this process:

[2] https://www.openstack.org/
[3] https://www.terraform.io/
[4] https://www.ansible.com/

- **Processor:** A three stage processing pipeline

  1. **Pre-Processing:** first prepares collected raw data for consumption (e.g., converting binary file formats to text formats).
  2. **Parsing:** Then parses unstructured text data into structured data formats, which are easier to utilize in further steps.
  3. **Post-Processing:** Finally, parsed data is utilized to extract additional information (e.g., execution timestamps of attack steps) and do additional dataset preparation steps (e.g., trimming logs to the datasets observation time frame).

- **Processing Database:** A schema-less database that is used to store information parsed by the *processor*. The database makes it possible to access and search for specific parsed information during the *Post-Processing* step and later when labeling the dataset.

- **Labeller:** A software component that can assign labels to log lines based on various types of *labeling rules*. *Labeling rules* are defined as configurable algorithms that access the *processing database* and apply some reasoning logic to decide if a label should be applied to a log line or not, for example, a simple database query verifying if the log entry matches certain criteria, such as containing the attacker machines IP address. The Cyber Range Kyoushi framework implements four types of labeling rules (discussed in section 5.5), but is designed so that more rule types can be easily added (**R8**). Note that labeling rule configurations are defined as templates on the model layer and then rendered using the information gathered by the data collection layer. This is done to ensure that labeling rules are automatically adopted to concrete testbed configurations (**R7**).

5.1.3 Components & Process Description

Above we discussed the system layers of the Cyber Range Kyoushi framework. Below we summarize the dataset generation and labeling process and the software components developed for the framework. For more detailed discussion on their implementations see sections 5.2 to 5.5.

Components

The following software components were developed as part of the Cyber Range Kyoushi framework:

- **Cyber Range Kyoushi Simulation**: A state machine execution library written in python. Using this library we implemented the user models and attacker model discussed in sections 3.2 and 4.1. It makes it possible to implement theoretical state machine models of users or attackers as executable code able to interact with a variety of systems and tools.

- **Cyber Range Kyoushi Gather:** An Ansible Role that is used to gather both files (e.g., log, configuration files, etc.) as well as system facts (e.g., network configuration, software package versions, etc.) from cyber range testbed systems.
- **Cyber Range Kyoushi Dataset:** A python CLI utility that implements both the *processor* and *labeller* functions discussed above. The CLI utility can consume simple YAML or JSON configuration files to define the processing pipeline and labeling rules. Internally it uses the log aggregation and parsing tool Logstash[5] to parse unstructured log data and the search and analytics engine Elasticsearch[6] as its *processing database*.

Process

Below we summarize the dataset generation and labeling process as defined by the Cyber Range Kyoushi framework:

1. Scenario definition
   - Testbed Model
   - User Models
   - Attack Models
2. Cyber range testbed provisioning
3. Dataset recording
   - User Simulation
   - Attack Simulation
4. Data collection
5. Dataset processing
   5.1. Pre-Processing
   5.2. Parsing
   5.3. Post-Processing
6. Dataset labeling
7. Dataset archiving & publishing

Note that steps 2 to 6 might be repeated multiple times for datasets that define multiple attack models that require a clean testbed state. For example, a dataset might define two distinct attack scenarios involving the same host, each of the scenarios might require that the host is in a certain state before the attack.

[5] https://www.elastic.co/logstash
[6] https://www.elastic.co/elasticsearch/

5.2 User & Attacker Simulation

The Cyber Range Kyoushi Simulation library provides a development API and Command Line (CLI) utilities for creating and running state machines. Developed state machines can be executed to automate the simulation of both attacker and normal user behavior in a cyber range, thus facilitating IDS dataset generation and other cyber range related activities.

5.2.1 API

The development API revolves around 6 major components:

- Statemachine
- Context
- State
- Transition
- TransitionFunction
- StatemachineFactory

Simulation state machines are defined by combining these 6 components. We discuss the API components below.

Statemachine

The Statemachine base class provides the high level state machine execution logic for state machines. In its most basic form it is initialized with an *initial state* and a list of states. After initialization a state machine can either run fully autonomously by executing `Statemachine.run()` or step-vise by calling `Statemachine.execute_step(...)`.

State machines will execute according to the *transitions* defined in its *states*. A state can have zero or more outgoing transitions, if a state has zero outgoing transitions it is referred to as a *final* or *end state*. Transitions are assigned a single target state. A special case exists where a transition does not have a target state, this is treated as the transition implicitly leading to an *end state*. Various state classes implement different strategies for selecting the next transition to execute. When using `Statemachine.run()` the library will keep executing state transitions until either an *end state* is reached or the *current state* has no outgoing *transitions*.

The Cyber Range Kyoushi Simulation library already includes various state machine base classes that allow the configuration of different execution behaviors. For example, the `WorkHoursStatemachine` allows the configuration of a work schedule and execution

start and stop times. That is, it is possible to define both work days and hours (e.g., Monday 8h-16h, Friday 9h-13h) for the state machine. The state machine will then only be active during the defined work schedule, outside of it the state machine will idle. This is particularly useful for modeling the behavior of employees of an company. Other special state machine execution behavior can easily be added by extending the functionality of existing state machine classes.

Context

The state machine *execution context* is used to pass information between the various *states* and *transitions*. A state machine *execution context* can be helpful when building a complex state machine for which some of the *states* or *transitions* depend on some shared information or objects. For example, when working with Selenium, to simulate web browser usage, one might want to store the Selenium[7] driver in the *context* so that *transitions* and *states* can all access the same Browser instance.

The library also integrates the *context* into the state machine life cycle through the `Statemachine.setup_context` and `Statemachine.destroy_context` methods. These methods are called at various execution stages (e.g., on execution start and stop), when running the state machine autonomously through `Statemachine.run()`. They can be used to initialize the *execution context* or otherwise prepare the state machine for execution. They could for example be used to start and stop a web browser instance.

States & Transitions

A *state* has zero or more outgoing *transitions* and implements a strategy for selecting the *transition* that should be executed next. *Transitions* have *transition functions* used to simulated various behaviors (e.g., open a link in the browser). The *transition* class also defines how these *transition functions* are executed.

The library provides various *state* and *transition* types, for example, a `ProbabilisticState` can be used to model probabilistic finite state machines or a `DelayedTransition` can be used to simulate random hesitation or pauses in between the various execution steps of an attacker or user model.

CLI & StatemachineFactories

The `StatemachineFactory` class is part of the Cyber Range Kyoushi library's plugin system. Using the plugin system it is possible to implement and define state machines as single python files or package plugins. The Cyber Range Kyoushi library's CLI utility can then detect these state machine plugins and execute them. Listing 5.1 shows such a factory definition, which declares a simple state machine with 3 sequentially connected states.

State machine execution is configured using two configuration files:

[7] `https://www.selenium.dev/`

- **CLI Settings:** The CLI settings are used to configure the CLI scripts behavior itself e.g., logging and the plugin system. Using this configuration file it is also possible to configure random seeds. This can be useful when trying to replicate an experiment as it ensures that randomized state machine behavior is identical between two runs. Note that the random seed only affects randomization within the python process, external tools executed by the state machine or operating system functions are not affected by it.
- **State machine config:** This configuration file is used to configure state machine specific options. For example, our state machine implementations for our user model discussed in section 3.2 declares a state machine configuration that allows us to configure the probabilities of state transitions and other model parameters.

```python
class StatemachineFactory(sm.StatemachineFactory):
  @property
  def name(self) -> str:
    return "GettingStartedStatemachineFactory"

  @property
  def config_class(self):
    return dict

def build(self, config: dict):
  # setup the states
  initial = states.SequentialState("initial", init_transition)
  example_state = states.SequentialState("example_state",
  ↪  example_transition)
  end = states.FinalState("end")

  # Initialize the state machine
  return GettingStartedStatemachine(
    "initial",
    [initial, example_state, end],
  )
```

Listing 5.1: Example State Machine Factory

5.2.2 Logging

As discussed in section 5.1.1 to ensure that it is possible to correctly and precisely label a dataset all available information should be used to its fullest extent. This also includes the activity logs of simulated actors (e.g., attackers or users). Such activity logs can be very

```
# bind selected transition and target to logger
log = log.bind(
  transition=self.current_transition.name,
  target=self.current_transition.target,
)
# execute transition
self.execute_transition(log)
```

Listing 5.2: Simulation Library - Log data binding

useful for retrieving execution timestamps of actions (e.g., attacker steps or user activity). Also depending on the user or attacker model definitions simulated behavior might be randomized, e.g., in our server take over scenario the order of commands executed by the attacker is randomized. More complex attack scenarios might even have completely randomized attack phases. In such cases execution data necessary for the labeling process cannot be extracted from the static attacker model, but rather must be extracted from the attacker simulation logs.

To ensure that all necessary information is available and can be easily processed, by both the processor and labeling components of the framework, the Cyber Range Kyoushi Simulation library heavily integrates structured logging principles within its state machine life cycle. For this we use the python structured logging library Structlog[8]. Using Structlog it is possible to create logging objects, and bind additional information (e.g., name of the currently executing transition) to them. The logging object can then be passed to the various state machine life cycle and transition functions. For example, listing 5.2 shows how the current transition and target state are bound to the log object before executing a transition.

Log messages and all bound information can then be easily output in the form of a line-separated JSON log. Since JSON is already a structured data format we can skip most of the parsing operations usually necessary when ingesting log information. Ensuring that simulation logs are designed to be in a structured formated also makes them easier to use during data extraction. For example, if we wish to retrieve the execution time of a specific simulated action we can simply search for the actions associated *state* and *transition*.

5.3 Dataset Collection

In context of the Cyber Range Kyoushi framework we differentiate between two types of data collection:

[8] https://www.structlog.org/en/stable/

- **Live capture:** The aggregation of log files is integrated into the testbeds system configuration. That is, the testbed systems are configured to use one of the many available log aggregation tools (e.g., Filebeats[9], Logstash[10], Syslog-ng[11]) to continuously send log entries to a centralized server.
- **Offline capture:** The data is collected as a single action, i.e., instead of the hosts continuously sending data to a central server. The data collector application collects the data in bulk from all the testbed systems at the same time. This is usually done at the end of a simulation. Note that facts collection might be done both before the simulation and once it ended. This can be useful when defined attack scenarios change system configurations as both the initial and the final state are recorded.

Both of these data collection types have their advantages and disadvantages. For example, the integrated nature of live capture means that it will always influence certain types of system or traffic logs. Depending on the configuration the data transfer, for example, will show up in logs of a Network and Host Intrusion Detection System (NHIDS) like Suricata[12] or other traffic captures. The operating system file read calls issued by the log aggregation application used might also show up in system logs (e.g., Auditd). Depending on the scenario this might be expected behavior anyways and thus not a problem, but in some cases, e.g., work stations usual are not configured to produce such traffic. One would have to take special care to prevent traces of the collection processes tainting the dataset.

In contrast offline capture does not introduce any additional logs into the dataset, since the data is collected as one single process outside the datasets observation time frame. It is also a lot simpler to use, e.g., any file transfer application is sufficient, but it also has its disadvantages. For example, log rows almost always have a timestamp, but some cases exist where one might observe a log row without a timestamp. Note that for our two scenarios we observed two such cases which are discussed in section 6.2. In a live capture scenario this would not be a problem as one could use the ingestion timestamp instead. The same is not possible when using offline capture, without an ingestion timestamp to fallback other techniques must be used to assign a somewhat accurate timestamp. In our testbed we, for example, used the timestamp of the previous log line.

5.3.1 Cyber Range Kyoushi Gather

Live data capture must be integrated with the testbed systems, thus it is not be possible to provide uniform configuration for all datasets. Always using the same log aggregation software for all datasets could bias machine learning algorithms. We therefore implemented Cyber Range Kyoushi Gather as an offline data capture utility. While it is an

[9] https://www.elastic.co/beats/filebeat
[10] https://www.elastic.co/logstash
[11] https://www.syslog-ng.com/
[12] https://suricata-ids.org/

offline data capture utility it is possible to utilize it alongside a live capture configuration, i.e., logs can first be aggregated using live capture and then retrieved from the central server using offline capture. This makes it possible to create datasets using a wide variety of software configurations, while also having a common tool used to retrieve data for dataset processing.

Cyber Range Kyoushi Gather is implemented as an Ansible role that can be used to retrieve both log and system files from all testbed systems. Using Ansible group and host vars, that make it possible to assign different configuration to groups of or specific hosts, it is possible to configure log and configuration file retrieval tailored to each host.

Cyber Range Kyoushi Gather also support the collection of system facts. Facts as defined by Ansible [Ans21], i.e., a fact is a variable containing information about a testbed system (e.g., IP addresses, list of installed packages, etc.). The CR Kyoushi Gather tool in fact uses Ansible fact modules and plugins to collect this information. In its default configuration the utility will use all standard Ansible fact modules and the external fact gathering tools Facter[13] and Ohai[14], if they are installed on the host. This makes it possible to retrieve a vast amount of facts about the system (e.g., network configuration, user accounts, installed software, etc.). Ansible's plugin architecture also makes it possible to add custom fact gathering modules should the default fact gathering configuration not be sufficient. The gathered fact information is saved by the Cyber Range Kyoushi Gather utility in a single JSON file per testbed host system. This makes it easy to access fact data during the dataset processing phase.

5.4 Dataset Processing

Dataset processing is an important part of the dataset generation and labeling process, as it bridges the two steps. That is, during dataset processing we transform, prepare and extract all the necessary information from the generated raw datasets so that we can label them. As briefly discussed in section 5.1.2 the Cyber Range Kyoushi framework defines dataset processing as a process with three stages. Below we first discuss this process in more detail, before discussing how the process is implemented within the CR Kyoushi framework.

5.4.1 Process

The framework defines three dataset processing stages: *Pre-Processing*, *Parsing* and *Post-Processing*. Each stage fulfills a certain role, we discuss them below.

Pre-Processing

The *pre-processing* is used to prepare a dataset for parsing. As discussed in section 5.1.2 the Cyber Range Kyoushi framework's processing and labeling components are designed

[13] `https://puppet.com/docs/puppet/7.6/facter.html`
[14] `https://docs.chef.io/ohai/`

to work with text based data formats. Due to this, one of the objectives of the pre-processing phase is the transformation of all non text based data to a text based format. For example, two common cases for this, we already encountered for our two scenarios (see discussion in section 6.2), are *PCAP Capture File Format* traffic captures and GZIP[15] compressed log files. Both of these two data formats have to be converted before they can be parsed by the Cyber Range Kyoushi parser component.

Apart from preparing data for consumption the pre-processing phase is also used to curate testbed facts and generate configuration files for further processing and labeling steps. Listing 5.3 shows a Jinja2[16] template of the Logstash parser input configuration. This template is rendered during the pre-processing phase. Cyber Range Kyoushi Dataset processors are used to first extract the necessary fact information (e.g., hostnames, log files and types, etc.) from the collected fact JSON files (see discussion in section 5.3.1) and then use the information to render the template. Similar configuration templates can also be used during pre-processing to curate various testbed facts into more compact data files to avoid having to load the large JSON fact files every time.

```
input {
{% for name, server in servers.items() %}
{% for log in server.logs %}
  file {
{%- if log.type is not none %}
    type => "{{ log.type }}"
{%- endif %}
    codec => "{{ log.codec }}"
{%- if log.path is string %}
    path => "{{ DATASET_DIR.joinpath('gather',name,'logs',log.path) }}"
{%- else %}
    path => [
{%- for _path in log.path %}
      "{{ DATASET_DIR.joinpath('gather',name,'logs',_path)}}",
{%- endfor %}
    ]
{%- endif %}
{%- if log.exclude | length > 0 %}
    exclude => {{ log.exclude | tojson }}
{%- endif %}
    mode => "read"
    exit_after_read => true
    file_completed_action => "log"
    file_completed_log_path => "{{ PARSER.completed_log }}"
    file_sort_by => "path"
    file_sort_direction => "{{ log.file_sort_direction }}"
{%- if log.file_chunk_size is not none %}
    file_chunk_size => {{ log.file_chunk_size }}
{%- endif %}
...
```

Listing 5.3: Pre-Processing - Parsing Input Config Template

[15] https://www.gnu.org/software/gzip/
[16] https://jinja.palletsprojects.com/en/2.11.x/

Parsing

The *parsing* phase has two objectives. First, the conversion of all unstructured text based log file formats into structured data formats for easier automated processing and labeling. Log formats are often meant to be human readable and not machine readable. Such log files must be transformed so that they can be utilized during the automated post-processing and labeling phases. The second objective of the parsing phase is to store the parsed structured data in a searchable schema-less database.

The Cyber Range Kyoushi Dataset tool implements this parsing and data storage process by relying on the external log aggregation tool Logstash. For the parsing phase static or templated (rendered during pre-processing) Logstash filters can be defined to parse a vast variety of log formats. For example, listing 5.4 shows the Grok[17] filter configuration used to parse DNSMASQ[18] log lines. The listing shows that four log line patterns are configured, each of the log line patterns defines a few sub patterns that match parts of the log message and assigned the matched content to a structured field. The first log line pattern, for example, saves the query type in the nested field `dns.question.type`.

```
13 grok {
14   match => { "message" => [
15     "%{SYSLOGBASE2} query\[%{DATA:[dns][question][type]}\]
       ↪  %{DATA:[dns][question][name]} from
       ↪  %{IP:[source][address]}",
16     "%{SYSLOGBASE2} %{DNSANSWERMODE:[dns][answers][mode]}
       ↪  %{DATA:[dns][answers][name]} is
       ↪  %{GREEDYDATA:[dns][answers][data]}",
17     "%{SYSLOGBASE2} forwarded %{DATA:[dns][question][name]} to
       ↪  %{IPORHOST:[destination][address]}",
18     "%{SYSLOGLINE}"
19    ]
20   }
21   pattern_definitions => {
22    "DNSANSWERMODE" => "(?:(reply)|(cached))"
23   }
24   overwrite => [ "message" ]
25 }
```

Listing 5.4: Parsing - DNSMASQ Grok Filter

The data parsed by Logstash is then stored in the search and analytics engine Elasticsearch. This makes it possible for us to easily query stored logs during the post-processing phase

[17] https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html

[18] https://thekelleys.org.uk/dnsmasq/doc.html

and use complex algorithms to query and reason based on the log events to assign labels. It is also possible to configure the Cyber Range Kyoushi framework to store parsed logs not only in the Elasticsearch database, but also on the file system as line-separated JSON files. Also providing log files in an already parsed format can be useful for those using the datasets to benchmark or train Intrusion Detection Systems, but raw log formats should always be provided as they usual implement their own parsing components. For example, the anomaly detection system discussed by Wurzenberger et al. [Wur+18] parses log files into sequences of tokens.

The Cyber Range Kyoushi framework also supports an alternative way to parse logs through Elasticsearch directly. Recent versions of Elasticsearch provide the possibility to use so called ingest pipelines for parsing log files. This is commonly used by the log aggregation tool Filebeat. Ingest pipelines have similar parsing capabilities as Logstash parsing filters, but are executed on the Elasticsearch node during data ingestion. The Cyber Range Kyoushi Dataset tool provides a processor to configure such ingest pipelines during the pre-processing phase. Note that since parsing happens on the Elasticsearch node when using ingest pipelines it is not possible to directly write parsed data to the file system using Logstash. This functionallity would have to be implemented through a post-processor that reads the Elasticsearch database and dumps the parsed entries into files.

Post-Processing

The final processing phase is the *post-processing* phase, it is mainly used to extract information from the parsed logs and render labeling rules (see section 5.5). For this purpose, Elasticsearch search queries can be used as part of configuration templates. Other than preparing the labeling rules, the post-processing phase can also be used for a few final modifications of the datasets.

For example, for our two scenarios we use the post-processing phase to remove all server logs which are not within the datasets observation time period. The datasets are trimmed after the parsing phase. Since the raw datasets contain many different timestamp formats, it is easier to filter out log lines with timestamps outside the observation time once this information can be queried through the dataset database. Also, as mentioned above, logs processed using ingest pipelines could also be saved to the file system in this phase.

5.4.2 Cyber Range Kyoushi Dataset - Processing

Above we discussed the Cyber Range Kyoushi processing process, below we discuss the implementation of this process using the Cyber Range Kyoushi Dataset CLI utility. Cyber Range Kyoushi Dataset's processing capabilities are implemented through so called *processors*, which work similar to Ansible modules, and the already discussed integration with the log parsing tool Logstash and the analytical engine Elasticsearch.

Configuration

Using the CLI utility dataset processing pipelines can be configured using a configuration file in YAML or JSON format. This configuration has three root configuration keys:

- **pre-processors:** A list of processor configurations which are read by the tool and executed in the defined order.
- **parser:** The parser configuration is used to configure the integration with Logstash, e.g., location of Logstash configuration files.
- **post-processors:** Same as the *pre-processors*, also a list of processor configurations executed in order.

Processors

Cyber Range Kyoushi Dataset *processors* are similar in design to Ansible modules. A processor defines input variables and can be used to execute specific actions. Listing 5.5 shows the configuration of a *template processor*. Each processor configuration has at least two mandatory attributes:

- **name:** A free text description of the action that is supposed to be achieved by this processor
- **type:** The type of processor that is being used. This attribute is necessary so that the CR Kyoushi dataset tool can execute the correct code.

Processors can also always be supplied with context variables. This can be done by either supplying them through configuration files (YAML and JSON are supported) or directly in the processor configuration. By using a context it becomes possible to supply parts of the processor configuration as Jinja2 templates. Every templated processor configuration is rendered before execution using the supplied context variables. The processor configuration in listing 5.5, for example, only partially configures the template context.

The Cyber Range Kyoushi Dataset tool already implements many useful processors, but new processors can be easily added by extending a base processor class, defining the input parameters and implementing the desired processing logic. Below we list the processors that are available by default:

- **TemplateProcessor:** A processor that can be used to render Jinja2 templates. This is useful for extracting fact information and curating it in compact configuration files or for other configuration files which can only be fully rendered at runtime.

```yaml
- name: Prepare server facts
  type: template
  context:
    var_files:
      groups: processing/config/groups.yaml
    vars:
     exclude_groups: ["Region.*", "envvars.*", "instance-.*",
      ↪  "meta-.*", "nova"]
     exclude_interfaces: ["lo", 'tun\d*']
  template_context:
    vars:
     servers: "{{ all }}"
  src: processing/templates/servers.json.j2
  dest: processing/config/servers.yaml
```

Listing 5.5: Template Processor Configuration

- **ForEachProccessor:** This is a special type of processor we refer to as *processor container*. The processor allows you to configure a list of items and a sub processor. The sub processor is then executed for each item. This is particularly useful in combination with using partial Jinja2 configurations for the sub processor as the sub processor will be rendered for each item separately.

- **CreateDirectoryProcessor:** A simple processor that creates a directory structure. This is useful for preparing the dataset directory for processing.

- **GzipProcessor:** This processor accepts either a file path or path glob, matching files are decompressed.

- **PcapElasticsearchProcessor:** This processor uses the external tool TShark[19] to convert *PCAP Capture File Format* traffic dumps into line-separated JSON log files.

- **TemplateCreateProcessor:** A processor for creating and configuring Elasticsearch index templates. This is useful for log files that potentially have many different fields (e.g., PCAP files). Usually Elasticsearch is configured to use dynamic field mapping, which adds field type mappings as it encounters new fields. For log files with many fields this can cause performance issues. Thus when working with such log files, it is often better to configure a static field type mapping using this processor.

[19] https://www.wireshark.org/docs/man-pages/tshark.html

- **IngestCreateProcessor:** This processor can be used to configure Elasticsearch ingest pipelines. As mentioned above such pipelines can used to parse log files using Elasticsearch instead of Logstash.
- **LogstashSetupProcessor:** This is a special processor used to configure the Logstash processing pipeline for the parsing phase. Among other things it can configure the Logstash input and output configurations and setup the necessary file system directory structure.
- **TrimProcessor:** A *post-processor* that can be used to trim specified log files to the datasets observation time period.

5.5 Labeling

The Cyber Range Kyoushi framework's approach is based on concepts from threat hunting, in fact, the framework's technology stack is very similar to a threat hunting setup discussed by Almohannadi et al. [Alm+18]. The ELK stack[20] of which Elasticsearch and Logstash are a major part of is also commonly used not only for log monitoring, but also for security operations. For this it provides so called detection rules that enable security operations teams to define various Elasticsearch queries used to identify potential incursions into the monitored systems. The Cyber Range Kyoushi framework's standard *labeling rule* types operate in a similar fashion [Gao+21; Alm+18].

In the context of the Cyber Range Kyoushi framework we define a labeling rule as a configured decision function that decides the problem of *assigning a label to log entries* based on information available in the dataset database (i.e., Elasticsearch) and the labeling rules configuration parameters. The framework is designed to make it possible to easily create and a new rule types, e.g., one could add a labeling rule type that uses trained classification algorithms to decide the problem of labeling. For the Cyber Range Kyoushi we implemented four different rule types, all of which are mainly based on Elasticsearch analytical query features.

Security operations teams only have limited information on potential attacks, e.g., they might know some behavioral signatures of specific attackers or attack types, but there are many variables unknown to them. This makes the process of threat hunting using analytical queries hard since some vital information needed to define efficient queries might be missing.

In the context of synthetic dataset generation and labeling we do not have this problem. According to the Cyber Range Kyoushi framework architecture the model layers contains the definition of all simulated user and attacker behavior, additionally we also process, parse and store logs of all simulated actors and testbed machines, even those which will not be part of the labeled dataset. This allows us to define labeling rules that can label server logs by relating them to defined or observed attacker behavior (e.g., attacker state

[20] https://www.elastic.co/what-is/elk-stack

machine or *PCAP Capture File Format* logs). We can also use labeling rules to assign negative labels, i.e., labels indicating that something is not attacker related, by creating rules that actively match and relate simulated user behavior. Such negative labels can be very helpful in cases where attack scenario actions are hard to distinguish from normal user behavior, e.g., attackers access the system through the same IP address or account as normal users.

5.5.1 Labeling Configuration

The Cyber Range Kyoushi Dataset CLI utility implements the labeling process. Similar to the processing pipeline labeling rules can be defined through either YAML or JSON configuration files. The CLI utility accepts a list of *rule directories* to search for these configuration files. Each file can contain a list of labeling rules. Rule files are applied in lexicographical order (as defined by the ASCII character table), labeling rule order within a single rule file is also respected.

By applying labeling rules one after each other it is possible to define multi stage labeling processes, where two or more rules cooperate to assign a single label. For example, in scenario 1 we first use a rule to label all log lines that relate to attacker HTTP traffic. We then apply a second rule that applies a label used to indicate the directory scan attack step only to those log lines that were previously labeled by our first rules and occurred within the observed directory attack time frame (also see section 6.3.1). In this example two rules are used to decide a very specific label. This example only involves two rules and is fairly simple, but more complex configurations are possible, e.g., rules applying previously mentioned negative labels can be used alongside rules applying positive labels.

The configuration format for a rule depends on its type, but there are a few common configuration fields shared by all rule types. These are listed below:

- **type:** The type field is used to indicate what kind of rule should be executed. It is possible to assign the id of any of the rule types loaded by the labeling system.
- **id:** All rules must be assigned a **globally** unique rule id. This id is used to store the label assignment in the dataset database. Label assignments are stored under their rule ids to ensure that it is possible to attribute label assignments to the originating rules.
- **labels:** A list of labels to assign to the log entries for which the rules decision function returns a positive response.
- **description:** The description field is optional, but is included in the base rule configuration to encourage documentation of labeling rules. Such documentation can be very useful for people using the dataset to understand the reasoning behind complex labeling rules. While it is possible to document labeling rules written in YAML format using comments, the same is not possible for JSON formatted rule files. We therefore provide this configuration field for rule documentation.

5.5.2 Labeling Rule Templates

In section 5.1.1 we discussed requirements of a dataset generation and labeling framework. One of the requirements (**R7**) was that the framework can handle pre-defined changes to the testbed, user models or attack scenarios, i.e., that there is no need to manually reconfigured the dataset processing pipeline and labeling rules. This requirement is especially important in the context of a model-driven simulation testbed, as discussed by Landauer et al. [Lan+20b]. Such a system can be used to rapidly generate multiple datasets by randomly assigning the models configuration parameters. Having to reconfigure labeling rules for each created dataset, would be a major bottleneck for the dataset generation process.

The Cyber Range Kyoushi framework, therefore allows the definition of labeling rules as templates on the model layer. Converting labeling rules into templates we can abstract variables that might change based on the testbed, user, or attacker models. The templated labeling rules are then rendered during the post-processing phase (see section 5.4), once all testbed and simulation information is available (i.e., testbed facts have been gathered and any required data has been extracted from the logs).

Listing 5.6 shows a templated labeling rule. This rule was used to label the DNS exfiltration traffic in scenario 2 and is a simple query that tries to match either the `dns.answer.name` or `dns.question.name` to the attacker controlled domain (by default for bool queries with should clauses at least one expression must apply). The rule uses two Jinja2 template directives to insert the attacker controlled domain into the regular expression query conditions. The domain is a variable part of the attacker model and must thus be dynamically extracted from the attacker model instance after all variables have been randomly assigned.

5.5.3 Rule Types

As mentioned above the Cyber Range Kyoushi Dataset CLI utility currently implements four types of labeling rules mostly based on Elasticsearch query logic. We discuss each of the rule types below:

Query Rule - elasticsearch.query

This is the most basic rule type implemented by the Cyber Range Kyoushi Dataset tool, it simply uses the Elasticsearch query DSL to search the database. All log entries matching the query are assigned the configured labels. This rule type can be used to configure most labeling rules that try to match specific attributes to a single log entry (e.g., specific source IP address). The Elasticsearch query DSL already provides many useful query functions that can be used to reason based on an entry's attributes (e.g., exact, regexp or fuzzy matching). For more complex query logic it is also possible to use so called script queries, which allow using painless scripts, a language based on Java, to query the database [Ela21b].

```yaml
- type: elasticsearch.query
  id: dnsteal.domain.match
  labels:
   - dnsteal
   - attacker
  index:
   - dnsmasq-inet-firewall
  query:
    bool:
      should:
       - regexp:
           dns.answers.name: '.*\.{{ attack.dnsteal.domain | regex_escape }}'
       - regexp:
           dns.question.name: '.*\.{{ attack.dnsteal.domain | regex_escape }}'
```

Listing 5.6: Template Processor Configuration

Listing 5.7 shows an example of a `elasticsearch.query` rule. The rule defines a simple Elasticsearch query that searches the `dnsmasq-inet-firewall` index; indices in Elasticsearch are similar to tables in SQL databases. The query shown in listing 5.7 is configured to filter out any log entries which are not within the specified time range (line 6 to 11). After filtering the query will match any log entry that has previously been matched by another rule (line 12 to 14). See the discussion on `dnsteal.domain.match` in section 6.3.1 for details on this. Note that it is also possible to define negative filters using the `exclude` configuration option.

Sequence Rule - elasticsearch.sequence

Sequence rules can be used to define event sequences, which are searched in the Elasticsearch database. Such event sequences can be a powerful tool to label log entries, which cannot be assigned a label on their own, but can be attributed to a series of malicious events (or non-malicious events when trying to decide negative labels). This rule type is implemented using Elasticsearches EQL query API [Ela21a].

Event sequences are defined using the following three configuration fields that reflect the EQL syntax:

- **by:** The *by* field accepts either a single field name or a list of field names. The given field names are used to join the log events, e.g., listing 5.8 shows an EQL query for which the *by* field is set to `url.full`. This means that all log events

```yaml
- type: elasticsearch.query
  id: dnsteal.domain.received
  labels: [dnsteal-received]
  index:
   - dnsmasq-inet-firewall
  filter:
    range:
      # dnsteal start up and stop time
      "@timestamp":
         gte: "2021-03-29T12:28:30.371Z"
         lte: "2021-03-31T23:30:00.281Z"
  query:
   match:
     kyoushi_labels.rules: dnsteal.domain.match
```

Listing 5.7: Query Rule - Example

of a single sequence must share the same value in the `url.full`. Note that it is also possible to define *by* directives as part of the sequence list (discussed below), should join fields not share the same name.

- **max_span:** The *max_span* configuration field allows us to define the maximum of time the full sequence can span. That is, the amount of time between the first and last event in a sequence.
- **sequences:** This field accepts a list of at least two event definitions in EQL query syntax. Similar to Elasticsearch DSL queries we can use various operators to match field values, but the EQL syntax imposes some limitations. For example, array comparisons are not supported.

Defined sequences are retrieved through Elasticsearch and labels are applied to all events using a second query. Note, due to current limitations of the Elasticsearch EQL API we had to apply a few special implementation measures to be able to use sequence rules properly. As of version 7.12 of Elasticsearch it is not possible to retrieve all results for an EQL query, i.e., there is no support for result pagination. To circumvent this issue we repeatably issue EQL queries and apply labels until we cannot find a single sequence that is not already labeled by the current rule. This is implemented by using a special query filter that exclude all events labeled with the rule from inclusion in sequences [Ela21a].

Listing 5.8 shows an example sequence rule configuration. As already mentioned above this rule joins two events based on their `url.full` field values. The rule shown in listing 5.8 is specifically used to match Apache access log entries to HTTP responses observed in the attackers *PCAP Capture File Format* traffic capture.

```yaml
- type: elasticsearch.sequence
  id: attacker.foothold.apache.access
  labels:
    - attacker_http
  index:
   - pcap-attacker_0
   - apache_access-intranet_server
  by: url.full
  max_span: 3m
  sequences:
   - '[ apache where event.action == "access" and
    ↪  source.address == "172.16.100.151" ]'
   - '[ http where source.ip == "172.16.0.217" and
    ↪  layers.http.http_http_response == true ]'
```

Listing 5.8: Sequence Rule - Example

Sub Query Rule - elasticsearch.sub_query

Sub query rules use a two stage query process to decide if a label should be applied to a log entry or not. First, log entries matching certain attributes are retrieved using the *main query*. The *main query* usually is used to search specific logs for which we already have done some reasoning (e.g., applied a label). Next, sub queries are executed for each result retrieved through the main query. All sub query results are then labeled.

The sub queries are defined as templates which can access the fields of the main query result, e.g., listing 5.9 line 32 shows a template directive accessing the main query results `url.full` field. While sequence rules only allow relating log entries in the context of log time series, sub query rules make it possible to relate log entries based on any type of attribute comparison.

Note that while sub queries can be powerful tools, they can also seriously impact the efficiency of the labeling process. This stems from the fact that sub query rules execute separate sub queries for each result of the main query.

Listing 5.9 shows a sub query rule used to label Apache access logs as attacker HTTP traffic. This rule covers a special case of log lines that are missed by the sequence rule shown in listing 5.8. Sometimes HTTP responses do not reach the attacker server, e.g., due to network issues. In such cases we have Apache access log lines that cannot be matched to a HTTP response in the attackers *PCAP Capture File Format* traffic capture.

The sub query rule shown in listing 5.9 is able to match these Apache access logs. For this the rule is defined to first retrieve all attacker HTTP requests within the attack time that have not yet been labeled as attacker HTTP (listing 5.9 line 7 to 26). Note that another sequence rule is used to pair labeled HTTP response with their respective

requests. Next sub queries trying to find matching Apache access log entries (i.e., same URL, IP, time frame) are executed and results are labeled (configured in listing 5.9 line 27 to 39).

```
1  - type: elasticsearch.sub_query
2    id: attacker.foothold.apache.access_dropped
3    labels:
4      - attacker_http
5    index:
6      - pcap-attacker_0
7    query:
8      term:
9        destination.ip: "172.16.0.217"
10   filter:
11    - term:
12        event.category: http
13    - term:
14        event.action: request
15    - range:
16        "@timestamp":
17          gte: "2021-03-23T20:32:09.309Z"
18          lte: "2021-03-23T21:12:52.152Z"
19    - bool:
20        must_not:
21          - script:
22              script:
23                id: test_dataset_kyoushi_label_filter
24              params:
25                labels: [attacker_http]
26   sub_query:
27     index:
28       - apache_access-intranet_server
29     query:
30       - term:
31           url.full: "{{ HIT.url.full }}"
32       - term:
33           source.address: "172.16.100.151"
34     filter:
35       - range:
36           "@timestamp":
37              gte: "{{ (HIT['@timestamp'] | as_datetime).replace(microsecond=0) }}"
38              lte: "{{ ( HIT['@timestamp'] | as_datetime).replace(microsecond=0)
                 ↪ + timedelta(seconds=1) }}"
```

Listing 5.9: Sub Query Rule - Example

Parent Query Rule - elasticsearch.partent_query

Parent query rules work in the inverse direction to sub query rules discussed above. For parent query rules we first retrieve a list of log entries we potentially want to label using a normal Elasticsearch DSL query. Then we execute *parent queries* for all retrieved results. These *parent queries* are used to find log entries that indicate that the original query result should be assigned the label, i.e., if we can find at least n matching parent log entries (n can be configured and defaults to 1) we assign the labels to the log entry. The

performance considerations for sub queries also apply to parent queries as they are very similar in implementation.

Listing 5.10 shows an example parent query used to label a special type of DNS exfiltration logs in scenario 2. When the dnsteal server shuts down before all file transfers for the exfiltration process have finished, the script will encounter DNS request timeouts. For our experiment discussed in section 6.3.2, we assign the label *dnsteal-dropped* to DNSMASQ logs for such timed out requests. We initially label them using a simple time based rule that matches all log entries assigned the *dnsteal* label that occur within the time period after the dnsteal server has shutdown and before the exfiltration script terminated. This rule is able to label most of the dropped logs, but misses a few requests that happen after the exfiltration service has shutdown. Such requests might occur if the system's DNS resolver automatically retries requests that have timed out.

To label the log lines for these request we can use a parent query rule. The rule shown in listing 5.10 first retrieves all log line labels as *dnsteal* that happen after the exfiltration service has stopped (line 6 to 21). Then it executes a parent query trying to find an identical log entry that has already been labeled as *dnsteal-dropped* (line 23 to 40). Note that we try to find an exact match since the unlabeled log lines should all be results of query retries, i.e., an identical timed out query executed while the exfiltration was still running must exist.

```yaml
- type: elasticsearch.parent_query
  id: dnsteal.domain.dropped-retry
  labels: [dnsteal-dropped]
  index:
   - dnsmasq-inet-firewall
  filter:
    range:
      "@timestamp":
        # exfiltration service stop
        gt: "2021-04-01T00:38:33.148Z"
        # we assume that query retries happen within
        # 3 minutes of the service shutting down
        lte: "2021-04-01T00:41:33.148Z"
  query:
   bool:
    must:
     - match:
         kyoushi_labels.rules: dnsteal.domain.match
    must_not:
     - match:
         kyoushi_labels.rules: dnsteal.domain.dropped

  parent_query:
   index:
    - dnsmasq-inet-firewall
   filter:
     range:
       "@timestamp":
         gt: "{{ ( HIT['@timestamp'] | as_datetime) -
          ↪  timedelta(minutes=3) }}"
         lt: "{{ HIT['@timestamp'] }}"
   query:
     bool:
       must:
        - term:
            dns.question.name: "{{ HIT.dns.question.name }}"
        - term:
            event.action: "{{ HIT.event.action }}"
        # and destination/source must also match depending on
         ↪  query type
        - term:
           "{{ 'source.ip' if HIT.event.action == 'query' else
            ↪  'destination.ip'}}":  "{{ HIT.source.ip if
            ↪  HIT.event.action == 'query' else
            ↪  HIT.destination.ip }}"
```

Listing 5.10: Parent Query Rule - Example

CHAPTER 6

Experiment

In this chapter we will discuss both of the two dataset generation experiments we conducted. We will first describe the general experiment parameters and setup, before then describing the used configurations of our labeling system for each of the experiments. We conducted these two attack scenario simulation experiments to test how well we are able to generate labeled datasets with our proposed approach. The expert evaluation of the final labels will be discussed in chapter 7.

6.1 Experiment Setup

To evaluate our framework we implemented the testbed, discussed in section 3.1, and the two attack scenarios, discussed in chapter 4. Each attack scenario was then simulated in two separate experiments, in an effort to generate two raw datasets to test our proposed framework. Other than the attacker configuration, both experiments used the same testbed configuration, i.e., the network topology and simulated user behavior were the same. It is to note though that while the topology was identical, the IP addresses of the network connected machines were not. The two scenarios were simulated during the following time periods:

- **Scenario 1 - Server Take Over:** from *2021-03-23T00:00:00.000Z* until *2021-03-28T00:00:00.000Z*
- **Scenario 2 - Data Exfiltration:** from *2021-03-30T00:00:00.000Z* until *2021-04-03T00:00:00.000Z*

Note that these time periods also mark the generated datasets observation time, i.e., the time periods which are considered for the final labeled datasets. All logs with timestamps outside of the observation period are not relevant for labeling and are not included as

part of the datasets, although they might be used as reference points or data sources during the processing or labeling phases.

At the end of each of the experiments simulations, we used the *Kyoushi Gather* Ansible role to collect both the logs and the facts from all testbed hosts. The raw datasets were then archived and backed up to a network drive, before we proceeded to process and labeled them using the *Kyoushi Dataset* tool we developed. Below we discuss the configuration for the processing and label of both datasets. Note that the processing configuration for both scenarios is mostly identical as they share the same underlying testbed configuration. As such, we will not make a distinction between the two. The label rules and labels depend not only on the testbed configuration, but mostly on the simulated attack scenario. Therefore we will discuss these separately for each of the scenarios.

6.2 Processing Configuration

We processed the two raw datasets collected from our experiments using the Kyoushi Dataset tools processing feature. For this we created a processing configuration file, a few configuration templates and logstash parser configuration files. Below we will give an overview of the processing setup for our two datasets. The complete processing and logstash parser configurations can be viewed in [Fra21a; Fra21b; Fra21c]. As mentioned in sections 5.1 and 5.4 within our framework dataset processing is split into three phases: *Pre-Processing*, *Parsing* and *Post-Processing*.

6.2.1 Pre-Processing

All processing that happens in the pre processing phase, for our two experiment datasets, can be categorized into three different categories: processors for compiling fact information, preparing the dataset for processing and those used to setup and configure the parser.

First, we have multiple processor configurations that have the purpose of extracting and curating the gathered testbed facts into structured configuration files. These can then be used as configuration context for other processors in the pre or post processing phases. For these pre processors we mostly use the template processor type and simply render the verbose fact information into a more compressed easier to use format. This makes the further processor configuration simpler as we only have to access the already curated configuration files instead of the verbose facts information that not only is complex, but also distributed across multiple files. Listing 6.1 shows one such pre processor configuration which is used to extract the most important facts (e.g., IP addresses, OS, etc.) about each of the testbeds virtual machines. This template processor uses the logs configuration and the facts files for each machine to render the servers template (excerpt shown in listing 6.2)

The second category of pre processors we use in our processing configuration involves pre processors which are used to prepare recorded logs for parsing. That is, some logs

```
17  - name: Prepare server facts
18    type: template
19    context:
20      var_files: processing/config/groups.yaml
21    template_context:
22      vars:
23        exclude_groups: ["Region.*", "envvars.*",
          ↪  "instance-.*", "meta-.*", "nova"]
24        exclude_interfaces: ["lo", 'tun\d*']
25        servers: "{{ all }}"
26      var_files: |
27        {
28        {% for server in all %}
29          "server_logs": "processing/config/logs.yaml",
30          "{{ server }}": "gather/{{ server }}/facts.json"{%
            ↪  if not loop.last %},{% endif %}
31        {% endfor %}
32        }
33    src: processing/templates/servers.json.j2
34    dest: processing/config/servers.yaml
```

Listing 6.1: Pre-Processing - Server Facts

are saved in a format which our parser cannot process directly so we have to convert them to a parseable format before the parsing phase. For our two experiment datasets we have two types of files that require such pre processing: log rotated compressed files and the attacker network traffic which is saved in the PCAP format. Note that while our parser could in theory read GZIP compressed files directly, leaving them in their compressed state would create problems for other stages in the processing pipeline. For example, we would not be able to trim compressed log files to the observation time without decompressing them. For consistency, we therefore always decompress early in the pre processing phase. The CR Kyoushi Datasets processing tool provides processors for both of these use cases. The configurations for these are shown in listings 6.3 and 6.4. Note that the PCAP processor uses the tool TShark[1] and some custom python code to convert PCAP files into logstash and elasticsearch friendly line based JSON log files.

Finally the third category of pre processors we use in our processing configuration are those pre processors used to configure the logstash and elasticsearch for the parsing process itself. For example depending the log files we might want to configure the index mapping manually instead of allowing dynamic mapping, or we might want to use ingest pipelines, which execute on the elasticsearch node, for parsing the log files instead of

[1] https://www.wireshark.org/docs/man-pages/tshark.html

```
 1  {
 2  {% for server in servers %}
...
20   "{{ server }}": {
21       "hostname": "{{ _server.ansible_facts.hostname }}",
22       "groups": {{ _server_groups }},
23  {% if "employee" is in _server_groups %}
24       "username": "{{ _server.employee_user.name }}",
25  {% endif %}
26  {% if "remote_employee" is in _server_groups %}
27       "openvpn_user": "{{ _server.openvpn_client_user }}",
28  {% endif %}
29       "distribution":
         ↪  "{{ _server.ansible_facts.distribution }}",
30       "distribution_release":
         ↪  "{{ _server.ansible_facts.distribution_release }}",
31       "distribution_version":
         ↪  "{{ _server.ansible_facts.distribution_version }}",
...
44  {%- if _server.dns is defined %}
45       "fqdns": {{ _server.dns.values()
         ↪  | selectattr("fqdn", "defined")
         ↪  | map(attribute="fqdn") | list | tojson}},
46  {%- endif %}
47       "logs": {{ _server_host_logs + _server_group_logs }},
48       "timezone": "{{ _server.ansible_facts.date_time.tz }}"
49   }{% if not loop.last %},{% endif %}
50  {% endfor %}
51  }
```

Listing 6.2: Pre-Processing - Servers Config Template

parsing them inside our logstash pipeline. For such cases we have to apply the necessary elasticsearch configuration during the pre processing phase. Listings 6.5 and 6.6 show two such pre processors, one is used to configure the index mapping for our attacker PCAP logs. We have to define this mapping manually as the network traffic dump can contain many different fields which can cause dynamic mapping to slow down our elasticsearch node significantly. The other processor configures an ingest pipeline for parsing auditd log files. This ingest pipeline was adapted from the ingested pipeline used by Filebeats Auditd Module[2]. We could have converted this ingest pipeline to a logstash pipeline, but

[2] https://www.elastic.co/guide/en/beats/filebeat/current/

```
57  - name: Decompress all GZIP logs
58    type: gzip
59    path: gather
60    glob: "*/logs/**/*.gz"
```

Listing 6.3: Pre-Processing - Log file decompression

```
62  - name: Convert attacker pcap to elasticsearch json
63    type: pcap.elasticsearch
64    pcap: gather/attacker_0/logs/traffic.pcap
65    dest: gather/attacker_0/logs/traffic.json
66    tls_keylog: gather/attacker_0/logs/premaster.txt
67    read_filter: "tcp or udp or icmp"
68    protocol_match_filter_parent: >-
69      tcp
```

Listing 6.4: Pre-Processing - PCAP to JSON conversion

```
74  - name: Add pcap index mapping
75    type: elasticsearch.template
76    template: processing/logstash/pcap-index-template.json
77    template_name: pcap
78    index_patterns: ["pcap-*"]
```

Listing 6.5: Pre-Processing - PCAP index mapping configuration

```
80  - name: Add auditd ingest pipeline to elasticsearch
81    type: elasticsearch.ingest
82    ingest_pipeline: processing/logstash/auditd-ingest.yml
83    ingest_pipeline_id: auditd-logs
```

Listing 6.6: Pre-Processing - Auditd Ingest Pipeline configuration

due to its complexity we choose not to do so.

6.2.2 Parsing

As already discussed in sections 5.1.3 and 5.4 we use logstash to parse log files. For this we have to configure logstash filters for the various log file types we recorded in our dataset. In [Fra21a] you can view the complete logstash parsing configuration used for both datasets. Most of the filters simply use standard logstash parsing practices and GROK patterns from the logstash patterns repository[3]. Below we will discuss special parsing configurations we used to make the logs easier to label.

Apache Logs

For the server take over scenario the Apache logs are one of the most important logs within the dataset. Therefore correctly parsing them and ensuring we have access to as much information as possible for labeling is vital. To this end we had to handle a few special cases as part of our parsing configuration. A few of our Apache logs labeling rules (discussed below) rely on matching attacker logs to Apache logs based on URI. For access logs this can very easily be done by simply combining the host name and path, but the same cannot be done so easily for Apache error logs. For scenario 1 we encountered two types of Apache error logs. Errors for standard Apache module, which report the file in which the error occurred, and PHP errors, which report the path and line number of the PHP script in which the error occurred. In both cases we can extract the paths using simple GROK patterns. The problem now is how we can infer the URI from the error files and scripts. For this we need to inject two pieces of information into our logstash parsing configuration. First we need the FQDN for each virtual host configured on the system. Next we also need all the web directories configured for these virtual hosts. With this information we can then replace the absolute paths of the error files and scripts with URIs starting with the FQDN. Note that some virtual hosts might use more than one web directory to serve content. This is for example the case for Wordpress which uses two special web directories for serving Javascript files. Listing 6.7 shows the logstash ruby filter configuration we used for this.

Another tricky special case we encountered with Apache error logs were log lines that do not conform to the Apache log format and do not have any timestamps at all. Such log lines can occur if the web service running on the Apache server writes custom error messages or otherwise unexpected errors to the Apache error log file. For our datasets we had two such cases. The OwnCloud web services would write a generic error without timestamp to the error log if it was configured to mount a Samba share, but did not have write permissions to `/var/run/samba/msg.lock`. While the missing file permissions do not impact OwnClouds support for mounting Samba shares, they do

`filebeat-module-auditd.html`

[3] `https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns/ecs-v1`

```
original = event.get("[apache][error][path]")
fqdn = event.get("[host][fqdn]")
event.get("[@metadata][kyoushi][httpd_dirs]")
 ↪  .each_slice(2) { |key, sub|
    original = original.gsub(key, sub)
}
event.set("[url][full]", fqdn+original)
```

Listing 6.7: Parsing - Apache Error Paths

produce many log lines. The other type of log line without a timestamp we encountered was a `Bad File Descriptor` error created due to the stream redirection of the reverse shell command used in scenario 1. For both cases we decided to artificially add timestamps to the log lines. We did this keeping track of the last observed timestamp and setting it as the timestamp for all log lines for which we could not extract a timestamp. Listing 6.8 shows the logstash filter configuration we used for this. Note that this type of timestamp issue is only a problem if we gather our logs statically, i.e., once after the simulation has ended. In a live capture scenario, i.e., we capture the log files as they are written, we could always use the time we observed the logs as fallback.

```
if "_grokparsefailure" in [tags] {
  ruby {
    code => '
      event.set("@timestamp", @@fallback_timestamp[event
 ↪  .get("[host][name]")+event.get("type")])
    '
  }
} else {
  ruby {
    code => '
      @@fallback_timestamp[event.get("[host][name]")+event
 ↪  .get("type")] = event.get("@timestamp")
    '
  }
}
```

Listing 6.8: Parsing - Handle Missing Timestamps

Syslog and Auth Log

Another case where we had to add special parsing configuration were syslog and auth log files. The default behavior for Linux syslog and auth logs is that the timestamps do not contain the year. This means that we have to manually add the year to the recorded timestamps to be able to properly label the log lines as our labeling process processes the log lines as a time series. For our two datasets we add the year to log timestamps by simply using the year value from our observation time. The ruby filter code for this is shown in listing 6.9. This approach only works as long the observation period does not cross the year boundary, i.e., the experiment starts and ends within the same year. Should this not be the case a more complex ruby filter, that also can handle the change in year, would have to be used. Note that similar to the Apache error log case above we could simply rely on the ingest timestamp, if we were to collect the dataset logs live.

```
103 year = @@observe_start.time.strftime("%Y")
104 event.set("timestamp",  year+" "+event.get("timestamp"))
```

Listing 6.9: Parsing - Handle Missing Timestamps

6.2.3 Post-Processing

During our post processing phase for both datasets we have three different objectives. First we want to trim the log files for the final dataset to only include timestamps within the observation time. For this we use the *dataset.trim* processor provided by the CR Kyoushi Dataset tool. The configuration for the processor is shown in listing 6.10. Note that we only trim the log files for hosts in the server group. We do this as these hosts are the only hosts that are actually part of the final labeled dataset, i.e., used to train or evaluate IDS algorithms. The other log files, e.g., attacker logs, logs of internet hosts, etc. are only used by us during processing and labeling.

Second we want to extract information from the now parsed log data and save it as configuration files. This is done mostly to get information about the execution of the attack scenario which is first known at runtime. Such information can therefore not be extracted from the static configuration files during the pre processing phase. Examples for such information would be the execution timestamps for the various attack steps within the scenarios or network port numbers assigned to various connections. Such information can be very useful during labeling to, e.g., to restrict the time range in which we try to apply a rule. To create the configuration files containing this information we again use template processors, but in contrast to the pre processing phase we now also can use elasticsearch queries from within our templates to access the parsed information. Listing 6.11 shows one such query used to extract the timestamp for the first VPN connection within scenario 1. The listing shows the YAML configuration for an elasticsearch DSL query. The query is used to select the log entry right before the attacker state machine initiates the VPN

```
93  - name: Trim server logs to observation time
94    type: dataset.trim
95    context:
96      var_files:
97        groups: processing/config/groups.yaml
98    # we only want to trim the logs of servers
99    # that will be part of the IDS dataset
100   indices: |
101     [
102     {% for server in groups["servers"] %}
103       "*-{{ server }}"{% if not loop.last %},{% endif %}
104     {% endfor %}
105     ]
```

Listing 6.10: Post-Processing - Log Trimming

connection. This is done by matching the log line based on the current state machine state, transition and the log message using *term* queries.

```
2   vpn_connect:
3     start:
4       bool:
5         must:
6           - term:
7               current_state: initial
8           - term:
9               transition: vpn_connect
10          - term:
11              message_keyword: connecting_to_vpn
```

Listing 6.11: Post-Processing - VPN timestamp query

Finally we use the dynamic configuration files created during both the pre and post processing phases to render our label rules templates. The configuration files are used as context for the template render process and complete the template rules with information specific to the current instantiation and execution of the testbed and attack scenario (e.g., IP addresses, execution timings, etc.). Listing 6.12 shows the scenario 2 post processor configuration for this. For scenario 2 we have two label rules templates files for audit and dnsmasq logs, which have to be rendered before we can apply them.

```
- name: Render label rules
  type: foreach
  items:
    - src: audit.yaml.j2
      dest: audit.yaml
    - src: dnsmasq.yaml.j2
      dest: dnsmasq.yaml
  processor:
    name: Render labeling rule template {{ item.src }}
    type: template
    context:
      var_files:
        attacker: processing/config/attacker/attacker.yaml
        servers: processing/config/servers.yaml
    src: "processing/templates/rules/{{ item.src }}"
    dest: "rules/{{ item.dest }}"
```

Listing 6.12: Post-Processing - Scenario 2 Labeling Rules Render

6.3 Labeling Rules & Labels

Below we discuss the labeling rules and labels used for each of our two experiment scenarios. The finalized, i.e., fully rendered, labeling rules are shown in [Fra21b; Fra21c].

Note that for our two datasets we choose to label *all* log lines resulting from *any* actions by the attacker. This means that even log lines resulting from what could be considered normal user behavior, e.g., simply visiting the homepage of a website, will be assigned a label. We have chosen this extensive labeling strategy, because it allows us to create a very verbose final labeled dataset. This means that it will be easier for interested parties to use the resulting datasets for various purposes, e.g., people wishing to test or evaluate very precise algorithms that detect actions of malicious actors rather than malicious activity will be able to do so. For use cases where only malicious activity is relevant the labels can simply be reduced to whatever is considered malicious within the datasets parameters. If our datasets only had labels for malicious activity alone the first example use case would not be possible. For this reason we chose this approach, as it is always possible to reduce label space, but it is not always possible or very hard to introduce more fine grained and verbose labels after the dataset has been published.

6.3.1 Scenario 1 - Server Take Over

For scenario 1 we have defined a total of eleven labels, each used to indicate specific steps or phases in the attack scenario. We apply these labels with 22 labeling rules for four different log file types. Note that for this experiment we decided to limit the

hosts, log files and scenario phases we intend to label. Specifically we only label the logs of the intranet and VPN servers. Also we disregard any attacker action before the directory scan. We accept these limitations in order to be able to more throughly explore the remaining log files and types for this initial evaluation of the framework. As such, we removed most of the broad network and service scans in favor of the often more interesting "manual" exploitation steps.

Labeling Rules

As mentioned above for this scenario we have defined 22 labeling rules. Below we will discuss these labeling rules. For the sake of brevity we do not include code listings for these rules in this section, they instead are shown in *rules/* [Fra21b]. Since we have defined many rules for this scenario we will discuss the rules in separate paragraphs for each log file type. That is we will first discuss the rules used to label the VPN log files, then the rules for the Apache logs and so on.

OpenVPN Logs. With the scenario the attacker establishes a VPN connection during each of the two attack phases. For this they use VPN credentials obtained from a simulated company employee. As mentioned in section 4.1 the process of obtaining the credentials is out of scope for this iteration of the attack scenario. The labeling rules below therefore are only used to label attacker connection related VPN logs.

To label the OpenVPN server logs we have defined four labeling rules a pair of two for each attack phase. Both of the pairs are basically identically except for the time frames they are applied to. This means we effectively have created two labeling rules that can be used to label attacker VPN connections and used them twice in slightly different configurations for both of our attack phases. It is to note that in theory we could have applied the two rules to the time periods for both attack phases at the same time, but choose not to do this as one of the rules uses a sequence query which performs better when its applied to smaller amounts of logs. The below list describes the two rules we defined:

- **attacker.foothold.vpn.ip and attacker.escalate.vpn.ip:** This rule applies the *attacker_vpn* label to all logs that contain the attacker server as source IP. With the exception of a few special log entries most OpenVPN logs always include at least the remote IP address. Since we parse all of our logs into a structured format during the dataset processing step we can very easily match the attacker IP to these log entries with a simple query. Listing 6.13 shows a few example OpenVPN log lines containing the attacker IP address `192.42.0.255`.
- **attacker.foothold.vpn.duplicate and attacker.escalate.vpn.duplicate:** This rule labels the OpenVPN log rows related to duplicate connections, i.e., the connections where the attacker connects while the legitimate user is already connected (see the label summary below for details on how we treat this special case). Some

```
2021-03-23 20:39:04 192.42.0.255:42062 VERIFY EKU OK
2021-03-23 20:39:04 192.42.0.255:42062 VERIFY OK: depth=0,
↪ CN=ffeichtner
2021-03-23 20:39:04 192.42.0.255:42062 peer info: IV_VER=2.4.4
2021-03-23 20:39:04 192.42.0.255:42062 peer info:
↪ IV_PLAT=linux
```

Listing 6.13: OpenVPN Log Excerpt

of the log entries written as a result of such a connection do not contain the remote IP address information. As such, they will not be matched by the IP rule described above. Listing 6.14 shows such log rows resulting from such a duplicate connection. Nonetheless using the CR Kyoushi sequence rule type we can still easily attribute such log lines to the attacker. For this we simply start the sequence at one of the connection log lines that contains the remote IP address information and then simply allows us to match consecutive lines even if they are missing this identifier.

Apache Logs. Out of the four log file types we have labeled for this scenario the Apache logs contain the most number of labeled log lines. Also more than half of the labeling rules defined are used for this log type. This can be explained by the fact that for one the Apache logs contain the traces of the directory scan which by far produces the most log lines and second by the fact that Wordpress instance is also the central access point for further exploitation.

For many of the Apache logs labeling rules discussed below we rely on matching the log lines to attacker traffic recorded via tcpdump. To be able to do this we had to ensure that the full URI (excluding the protocol scheme) is available as data field for both log types. This was done during parsing of the dataset.

- **attacker.foothold.apache.access and attacker.foothold.apache.error:** Both of these rules work very similar and the only real difference between the two is the log files they are applied to (Apache access and error logs). The rules apply the *attacker_http* label to all Apache access/error log rows that can be matched to a HTTP response recorded in the attackers traffic log. For this we use a sequence rule pairing the two log lines based on the `url.full` field containing the full URI as mentioned above. We further narrow down the logs we apply this rule to by requiring that the source address in the Apache log must be the VPN server and the source address in the PCAP HTTP response must be intranet server. Remember as discussed in section 3.1 traffic from machines connected to the VPN will appear to originate from the VPN servers IP address due to the use of DNAT. With these two rules we are able to label almost all of the attackers HTTP traffic. Only a few special cases require additional rules. Note that we use the HTTP response

```
1  2021-03-25 20:36:00 192.42.0.255:48833 [ffeichtner] Peer
   ↪ Connection Initiated with [AF_INET]192.42.0.255:48833
2  2021-03-25 20:36:00 MULTI: new connection by client
   ↪ 'ffeichtner' will cause previous active sessions by this
   ↪ client to be dropped.  Remember to use the --duplicate-cn
   ↪ option if you want multiple clients using the same
   ↪ certificate or username to concurrently connect.
3  2021-03-25 20:36:00 MULTI_sva: pool returned IPv4=10.9.0.10,
   ↪ IPv6=(Not enabled)
4  2021-03-25 20:36:00 MULTI: Learn: 10.9.0.10 ->
   ↪ ffeichtner/192.42.0.255:48833
5  2021-03-25 20:36:00 MULTI: primary virtual IP for
   ↪ ffeichtner/192.42.0.255:48833: 10.9.0.10
6  2021-03-25 20:36:00 Outgoing Data Channel: Cipher
   ↪ 'AES-256-CBC' initialized with 256 bit key
7  2021-03-25 20:36:00 Outgoing Data Channel: Using 160 bit
   ↪ message hash 'SHA1' for HMAC authentication
8  2021-03-25 20:36:00 Incoming Data Channel: Cipher
   ↪ 'AES-256-CBC' initialized with 256 bit key
9  2021-03-25 20:36:00 Incoming Data Channel: Using 160 bit
   ↪ message hash 'SHA1' for HMAC authentication
10 2021-03-25 20:36:01 ffeichtner/192.42.0.255:48833 PUSH:
   ↪ Received control message: 'PUSH_REQUEST'
```

Listing 6.14: OpenVPN Log - Duplicate Connection

for this pairing due to the fact that the Apache access logs in our testbed use timestamps with second precisions. This means that a request might appear after the corresponding access log in the time series due to the missing milliseconds. By pairing the logs using the response we can avoid this problem as the response must be after the access log entry anyways.

- **attacker.foothold.pcap.requests:** This rule is a special case since its purpose is not really to label attacker related server logs, but is rather used to mark attacker HTTP requests for which we already labeled the corresponding access log entries. This is done to reduce the number of HTTP requests we have to consider for the *attacker.foothold.apache.access_dropped* rule. After this rule has been applied only the attacker HTTP request, for which no response was observed (e.g., due to network issues or timeout), should be unlabeled.
- **attacker.foothold.apache.access_dropped:** This is used to label a special case of Apache access logs. Specifically those for which we have an access log entry, a HTTP request, but no HTTP response. As already mentioned above this can

happen due a variety of network or web application related errors. Using the *attacker.foothold.pcap.requests* we have labeled all request and response pairs. This means we can now apply the rule to all unlabeled requests knowing that they do not have a matching response. For this we use a *sub_query* rule, meaning we use two consecutive queries. With the first query we retrieve all unlabeled HTTP requests. Then we issue a sub query for each result. The sub query is used to find an Apache access log entry that matches the requests URI, source address (i.e., the VPN) and occurred within a one second window of the request.

- **attacker.foothold.apache.access_error:** This rule matches already identified access log lines to matching error log lines by use of a sequence rule. By doing this we can easily label error log rows that have been previously missed, but for which we were able to label the corresponding access log entry.

- **attacker.foothold.apache.error_index:** As mentioned in section 6.2.2 for Apache error logs we extract the corresponding URI from the file or script the error occurred in. This approach gives us the correct URI for most files, but there exist some special cases were the parsed URI does not match the actual requested URI. One of those cases involves requests to directories, i.e., those ending with "/". For many web applications such requests are often mapped to special index files, e.g., *index.php*. Such index mappings also exist for the Wordpress application running on the intranet server. As such we had to define this rule to cover this special case. For this we use a *sub_query* labeling rules with the main query retrieving all access log entries ending in "/" that have been previously marked as *attacker_http*. Sub queries is then used to also apply the *attacker_http* label to all error log entries matching the directory request URI appended with *index.php* that are within a 3 second time window of the access log entry.

- **attacker.foothold.apache.error_access:** This rule follows an inverse approach to the *attacker.foothold.apache.error_index* rule. It uses a *parent_query* labeling rule trying to match unlabeled error log entries to already labeled access log entries. The rule works by first retrieving a list of all unlabeled error log entries within the attack time frame. Then we use a parent query to find a matching labeled access log entry. Should such an entry exist then the original error log entry is labeled as *attacker_http*.

- **attacker.foothold.apache.php_warns:** This rule tries to apply missing labels to error lines that result from multiple PHP errors/warnings in a single request. First all PHP errors that are potentially from the attacker (based on the source address) are retrieved and then we try to find a labeled parent by matching *pid*, *port*, *url.full* and the approximate error time. We need this rule to since some requests will produce more than one PHP error/warning. These extra error log lines will be missed by the *attacker.foothold.apache.error* sequence rule as it will only apply the label to the first matching error log line.

- **attacker.dirb.time:** This rule applies the *dirb* label to attacker HTTP traffic resulting from the dirb directory scan. For this we use a simple query rule that matches all access and error logs marked with *attacker_http* and also occurs within the dirb execution time period. This rule assumes that the attacker will only execute the directory scan during this time period and not issue any other HTTP requests.

- **attacker.webshell.upload.seq:** This rule labels the web shell upload step by matching the 3 step sequence within the foothold phase. The rule requires us to extract the Wordpress instances base path, chosen post and the URI for the admin ajax endpoint from the attacker configuration and state machine logs.

- **attacker.escalate.webshell.cmd.http:** This rule matches the web shell web requests via the recorded web shell URI and web shell request params. For this rule we extract the web shell URI and the command HTTP request params from the attacker state machine log during the post processing phase. Using this information we can then render a simple query matching access log lines for the web shell URI that have the VPN server as source address and contain at least one of the web shell commands requests params. Note that we must also include the request params in our query rules as normal user behavior can also result in requests to the web shell URI. This can for example happen if a simulated user that has the wordpress editor activity visits the media menu. A request for the web shell will be issued since the web shell is present in the list of uploaded image files.

- **attacker.escalate.shell.descriptor:** This rule applies the *reverse_shell* label to "Bad file descriptor" error messages. The reverse shell command used does not do redirection nicely so we get such an error. For this we use a simple query rule that matches the phrase "Bad file descriptor".

Auth Logs. The intranet servers auth logs contain two types of attacker related log lines. First, those that relate to the attacker escalating from the web service user *www-data* to the local system user for which they were able to crack the password. Second, those that are the result of the user issue privileged commands within the elevated user session. The labeling rules used to label these logs are discussed in the list below:

- **attacker.escalate.su.login:** This sequence rule labels auth log rows resulting from the attacker escalating to a different user. This is done by relating the event sequence using the target and source user. Further we also match the source and target users of the change user events with the web service user, based on the intranet servers configuration facts, and the escalation user according to the attacker state machine log.

- **attacker.escalate.sudo.command:** This sequence rule labels auth log rows resulting from the attacker invoking a sudo command through the hijacked user. We

label this by relating an already labeled change user event to the sudo events based the TTY and user of the log events.

- **attacker.escalate.sudo.open:** This rule labels auth log rows resulting from the attacker invoking a sudo command that opens a pam session through the hijacked user. We label this by relating it to an already labeled sudo command based on proximity, i.e., the next sudo open and close session events after the command event will be labeled. This rule is only applied to pam session open and close events which occur close to each other. We limit us to such events as attribution of pam these events to their respective processes, becomes almost impossible if we observe multiple events in a short timeframe. The reason for this is, the fact that the log entries for these particular pam events do not contain any information that can be used to attribute them to any specific process or session.

Audit Logs. Similar to the auth logs the audit log also contains logs for the user escalation and privileged commands issued by the attacker. Note that some of our audit log labeling rules rely on already labeled auth log entries. Therefore we ensure that the auth log rules are applied before the audit log rules by prefixing the auth log rule file with "*0_*". This ensures the execution order since the CR Kyoushi Dataset labeling function applies rule files in lexicographically order. The audit log labeling rules are discussed in the list below:

- **attacker.escalate.audit.su.login:** Similar to its auth log counter part this rule labels the user escalation events created by the attacker calling /bin/su and changing to the hijacked user. For this we use a sequence rule that matches the su change user events starting from the user ID of the web service user to the cracked user. Note that for this rule we do not only need the user names of both the web service user and the escalation user, but also their user ids. All this information can again be easily extracted from the intranet servers configuration facts and the attacker state machine log.

- **attacker.escalate.audit.sudo.command.start:** This rule matches auditd ran-command events for sudo commands with their respective auth log lines which are already labeled. To do this we also limit the audit log events which potentially can be a part of such a sequence, to events for commands actually executed by the attacker. The commands executed can be retrieved from the attacker state machine log.

- **attacker.escalate.audit.sudo.command.events:** This rule labels all expected audit events for a sudo command (e.g., *CRED_REFR*) to an attacker command that was labeled by the *attacker.escalate.audit.sudo.command.start* rule. Since we can join the events directly via the process id we can be fairly certain that resulting labels are correct as long as the parent rule is correct.

Labels

Below we summarize and describe all labels for scenario 1.

- **foothold:** This label is applied to all logs resulting from attacker actions taken in the foothold phase of the scenario.
- **escalate:** Similar to the foothold label, this label is assigned to all attacker related logs in the escalation phase.
- **attacker_vpn:** This label is applied to all OpenVPN log lines resulting from the attacker using the hijacked VPN credentials to connect to the company network. Note that the OpenVPN server is configured to only allow a single VPN connection for each user. This means that if the user, whose VPN credentials have been stolen, and the attacker connect at the same time the person who connected first will be disconnected. Such a duplicate connection and forced disconnect also produce specific log lines. For these we only labeled log lines as *attacker_vpn* if the legitimate user was the one to be disconnected, i.e., the attackers connection removed the user from the companies network. Cases where the user connecting to the VPN causes the attacker to loose the connection are specifically not labeled as *attacker_vpn*.
- **attacker_http:** This label is used for all logs which are the direct result of attacker HTTP traffic, e.g., the directory scan or web shell related exploitation steps.
- **dirb:** This label is used for all logs resulting from the directory scan executed with the dirb[4] web scanner. As this is all HTTP traffic, the logs that have this label are a subset of the logs labeled with *attacker_http*.
- **webshell_upload:** This label marks the attacker actions executed in order to upload a web shell by exploiting CVE-2020-24186 (an arbitrary file upload vulnerability).
- **webshell_cmd:** This label marks all logs resulting from commands sent to the already uploaded web shell.
- **reverse_shell:** This label is used to mark logs resulting from the creation of a reverse shell connection from the target host to the attacker server.
- **attacker_change_user:** This label is applied to log entries related to the attacker escalating their privileges from the web service user to a local user for which the credentials have been cracked.
- **escalated_command:** This label marks all logs which are a direct result of commands executed in the escalated user session.

[4] https://tools.kali.org/web-applications/dirb

- **escalated_sudo_session:** This label is applied to all sudo session log entries which are the result of the attacker executing a sudo command during the escalation phase.

Section 6.3.1 shows the distribution of labels among the various log files and hosts. The table shows that the vast majority of labeled log lines are labeled as *dirb*. This is an expected result as the automated directory scan is a very noisy attack step. For this scenario we have labeled a total of 408038 log entries, of which only 368 are not attributed to the directory scan. Note that the final table row contains the total some of all labeled log lines for each file. Also note that this total value cannot simply be calculated by summing all rows, since some of the labels overlap, i.e., are applied to the same log lines, or are sub or super sets of each other.

	vpn/openvpn.log	intranet/wordpress-access.log.4[5]	intranet/wordpress-error.log.2[5]	intranet/error.log.2	intranet/auth.log	intranet/audit.log	Total
foothold	308	406402	1268				407978
escalate	28			1	12	19	60
attacker_vpn	336						336
attacker_http		406402	1268				407670
dirb		406392	1268				407660
webshell_upload		3					3
webshell_cmd		7					7
reverse_shell				1			1
attacker_change_user					5	9	14
escalated_command					8	10	18
escalated_sudo_session					6		6
	336	406402	1268	1	12	19	408038

Table 6.1: Scenario 1 - Labels per log file

[5]The servers FQDN "intranet.company.cyberrange.at" was replaced with the string "wordpress"

6.3.2 Scenario 2 - Data Exfiltration

Labeling Rules

Compared to the server take over scenario discussed above the data exfiltration scenario is less complex. This difference in complexity can also be observed in the amount of labeling rules we require to label the resulting datasets. As discussed above for scenario 1 we created a total of 22 labeling rules to be able to label the dataset. For scenario 2 we only have five labeling rules all four of which are simple query rules. The list below summarizes the four labeling rules, also see the corresponding rendered rules files are shown in *rules/* [Fra21c].

- **dnsteal.domain.match:** This rule simply matches all DNS queries and answers that end with the sub domain used in the dnsteal exfiltration. The configured sub domain is extracted from the attacker servers facts during the pre processing phase. Now this rule only works if we assume that users will not query our malicious sub-domain which should be a reasonable assumption. Given a scenario that includes a simulated that might also query the sub domain, e.g., a simulated SOC team investigating the unusual DNS traffic, we would have to do some more filtering. For example we could use sequence queries to match requests to dnsteal server log lines or do negative matching based on the simulated user logs.
- **dnsteal.domain.received:** This rule applies the *dnsteal-received* label to all logs, which already have the *dnsteal* label and occurred during the time period the dnsteal server was active. This is done by simply filtering based on the observed start and stop timestamps and the presence of the *dnsteal* label.
- **dnsteal.domain.dropped:** This rule applies the *dnsteal-dropped* label to all log lines, which are marked as *dnsteal* and occured while the dnsteal server was inactive. Note that we define the dnsteal servers inactive time period to be from the moment the dnsteal server stopped until the exfiltration service stopped. Both of these time stamps can easily be extracted from our parsed log files before rendering the rules.
- **dnsteal.domain.dropped-retry:** This rule applies the *dnsteal-dropped* label to dnsteal marked log lines that occurred when both the dnsteal server and the exfiltration script where inactive. Specifically we try to correctly label DNS query retries that occurred after the time frame considered for the *dnsteal.domain.dropped* rule. Such logs can occur if the system resolver retries queries even if the requesting process has already shut down. To make sure that we do not match unrelated DNS queries traffic we use a parent query rule, which first gets all remaining logs not labeled as either *dnsteal-received* or *dnsteal-dropped*. Next the rule checks if we can find a matching identical parent request already labeled as *dnsteal-dropped*. If we find such a parent we apply the label.
- **exfil.service:** The exfiltration service does not produce any notable logs in the current scenario and testbed definition. As such this rule simply captures the

services process events (e.g., start and stop). As the start time probably precedes the observation time we will only match the stop event with this.

Labels

Below we summarize and describe all labels for scenario 2.

- **dnsteal:** This label indicates DNS exfiltration traffic by the attacker dnsteal server, i.e., all log lines resulting from data transfers via DNS. For this the attacker uses a modified version of the tool dnsteal[6].
- **dnsteal-received:** This label indicates DNS exfiltration traffic that was correctly received by the attacker dnsteal server.
- **dnsteal-dropped:** This label indicates DNS exfiltration traffic that was not received by the attacker dnsteal server. Such traffic occurs when the attacker server shuts down or otherwise becomes unreachable meaning DNS queries can be sent by the exfiltration script, but will not receive a response.
- **exfiltration-service:** This label indicates activity of the data exfiltration script on the infected machine (e.g. in the audit or syslog).

Section 6.3.2 shows the distribution of labels across the various log files. Note that as the *dnsteal* label is a superset for all other labels, its total count is also the total count of all labeled log lines. The table shows that a total of 43303 log lines have been labeled for this scenario. All labels with the exception of a single log line, an audit log entry for the exfiltration service shutdown, are contained in the company firewalls dns log. Given the modeled scenario this is an unsurprising result.

	firewall/dnsmasq.log	share/audit.log	Total
dnsteal-received	36536		36536
dnsteal-dropped	6766		6766
exfiltration-service		1	1
dnsteal	43302	1	43303

Table 6.2: Scenario 2 - Labels per log file

[6] https://github.com/m57/dnsteal

CHAPTER 7

Evaluation

In the previous chapter we described two dataset generation experiments we conducted. For these experiments we used our proposed methodology and proof of concept tool implementations to generate labeled datasets for the two attack scenarios discussed in chapter 4. Below we discuss the expert evaluation we conducted to verify the correctness of the resulting labels. First we discuss the sampling strategy we used for extracting log lines to use in the expert survey. Last we discuss the results of the expert survey.

7.1 Dataset Sampling

As we already discussed in sections 6.3.1 and 6.3.2 we have labeled a total of 408038 and 43303 log lines for scenario 1 and 2 respectively. Given this considerable amount of labeled log lines alone it is not feasible to verify the correctness of all assigned labels and log lines which have not been assigned a label. Thus we need to carefully select which logs and labels we include in our evaluation process. Additionally we also have to decide how many log lines are to be sampled from the chosen files and labels. Below we first discuss our choice of labels and log lines for the evaluation. Then we will further discuss the sampling sizes and strategy.

7.1.1 Sampling Sources

The scenario 1 dataset has been labeled using a total of 11 different labels, each used to indicate a specific phase or step within the attack scenario. Many of the assigned labels can be considered as super or sub sets of each other. For example, the *foothold* label is applied to all labeled log entries with a timestamp within the foothold phase execution time. Similarly the *attacker_http* label simply marks attacker HTTP traffic related logs, as such it is a clear super set of the *dirb*, *webshell_upload* and *webshell_cmd* labels.

Keeping the relationship between our labels in mind we chose 6 out of the 11 scenario 1 labels for our expert survey. Namely the following scenario 1 labels were selected:

- *dirb*
- *webshell_upload*
- *webshell_cmd*
- *reverse_shell*
- *attacker_change_user*
- *escalated_command*

These labels were selected, because they are, with the exception of the *escalated_command*, not releated to a more specific label. That is, there is no label that contains a subset of the log lines marked by the label. Choosing the most specific labels has two benefits. First, we avoid accidentally sampling the same log lines multiple times as the labels mark distinct sets of log lines. Second, each of these labels also indicates a clearly definable step within the attack scenario. For example, *webshell_cmd* is only assigned to log lines related to the commands executed via the web shell during the foothold phase. Having such a narrow definition reduces the factors one has to consider when making a decision about the correctness of a label. Thus we can reduce the cognitive strain for our expert reviewers. In contrast would we have chosen one of the very general labels, such as the *foothold* label reviewers would have to consider many factors at once making their decision process much harder.

Following a similar reasoning we also excluded the *attacker_vpn* label from the evaluation process. While this label is specific and also does not have any label that forms a sub set of its labeled log line we decided against including it. The reasoning behind this is that its inclusion would have added a lot of additional complexity for our expert reviewers. While all of the 6 labels we have chose are used to label log files on the intranet server, the *attacker_vpn* label is used to label the OpenVPN server log on the VPN server. This means that if we had included this label our expert reviewers would have had to familiarize themselves with an additional testbed server and log format.

For scenario 2 we only have four different labels, thus selecting labels to sample from was not really a problem. Nonetheless there were still other factors we had to consider when deciding how to use the labels for the evaluation. Specifically as was discussed in section 6.3.2 for scenario 2 we have the following labels: *dnsteal*, *dnsteal-received*, *dnsteal-dropped* and *exfiltration-service*. The *dnsteal* label is the super set that marks all log lines related to the attacker, i.e., it contains all log entries that have been labeled with any of the other labels. Thus we decided to draw the samples *dnsteal-received*, *dnsteal-dropped* and *exfiltration-service* labels.

While we decided to sample based on these three labels we decided to not reveal the *dnsteal-received* and *dnsteal-dropped* labels to the expert reviewers. Instead we indicate sample log lines from these two labels with the general *dnsteal* label. We do this because recognizing if a label should be marked as *dnsteal-received* or *dnsteal-dropped* is very hard for a person not intimately familiar with the scenario and label definition.

For example, listing 7.1 shows an excerpt of the dnsmasq log for scenario 2. Log lines 1 to 3 in this log excerpt should be labeled as *dnsteal-received* and lines 4 to 6 should be labeled *dnsteal-dropped*. On first and second glance all 6 log lines look almost identical. The only real noticeable differences are the queried domains and log line type (i.e., query, forwarded and reply). Also it is to note that within these log lines only two unique domains occur, one domain shared by log lines 1 to 3 and another domain shared by log lines 4 to 6. The reason why we know log lines 1 to 3 should be labeled as *dnsteal-received* stem from the definition of both labels. The label *dnsteal-received* is only applied to data exfiltration DNS queries that have been received by the dnsteal server. This means that on the company DNS server there must be one log line for each of the following steps:

1. The company DNS server receives the query
2. Then forwards it to the upstream DNS server for resolution
3. Lastly the company DNS server receives a reply to the forwarded query

As depicted in listing 7.1 log lines 1 to 3 match these steps exactly. Log lines 4 to 6 on the other hand do not. Log line 4 and 6 both are logs for the same query and there is no reply log line. This indicates that these lines should be labeled as *dnsteal-dropped*. As this example illustrates the difference between *dnsteal-received* and *dnsteal-dropped* labeled log lines is hard to distinguish and it would be very easy for a person unfamiliar with the attack scenario to make a mistake. Having log lines that cannot be reliably decided by an expert as part of our verification process, would hamper our ability to compare the collective assessment of our reviewers with the automatically assigned labels. Therefore while we sample based on these two labels, we only ask the experts, participating in the survey, to decide if a log line is correctly labeled as *dnsteal* or not.

7.1.2 Sample Sizes & Strategy

Our decisions in regards to sample sizes for our expert evaluation are driven by two factors.

- The total amounts of logs for each of the selected labels.
- The amount of log lines participating experts will be able and willing to review.

```
1 Mar 31 23:29:50 dnsmasq[3438]: query[A] 3x6-.12-.0a6OLJorWypr
  ↪ k8wccUQM21fddC3/jqGGTfVIxhB25CjNDVh-.XtXPlMub9W3cl/lHtMvX
  ↪ 55IpS*rM*XGzsDek*K8XJ6nOU*q-.M*zrQ0HQjbEbynquXkE1cLjtHRg9
  ↪ JLDbL1enQOMjYKFzfjk-.yslQzF6yRfFHYhhUrzGL0aqR*kkLGhdfbZhN
  ↪ 6Vcr7UKVHS7-.KelleyPlc.docx.wtulqzvb.at from
  ↪ 172.16.0.187
2 Mar 31 23:29:50 dnsmasq[3438]: forwarded 3x6-.12-.0a6OLJorWyp
  ↪ rk8wccUQM21fddC3/jqGGTfVIxhB25CjNDVh-.XtXPlMub9W3cl/lHtMv
  ↪ X55IpS*rM*XGzsDek*K8XJ6nOU*q-.M*zrQ0HQjbEbynquXkE1cLjtHRg
  ↪ 9JLDbL1enQOMjYKFzfjk-.yslQzF6yRfFHYhhUrzGL0aqR*kkLGhdfbZh
  ↪ N6Vcr7UKVHS7-.KelleyPlc.docx.wtulqzvb.at to
  ↪ 192.42.52.58
3 Mar 31 23:29:50 dnsmasq[3438]: reply 3x6-.12-.0a6OLJorWyprk8w
  ↪ ccUQM21fddC3/jqGGTfVIxhB25CjNDVh-.XtXPlMub9W3cl/lHtMvX55I
  ↪ pS*rM*XGzsDek*K8XJ6nOU*q-.M*zrQ0HQjbEbynquXkE1cLjtHRg9JLD
  ↪ bL1enQOMjYKFzfjk-.yslQzF6yRfFHYhhUrzGL0aqR*kkLGhdfbZhN6Vc
  ↪ r7UKVHS7-.KelleyPlc.docx.wtulqzvb.at is
  ↪ 195.128.42.1
4 Mar 31 23:30:00 dnsmasq[3438]: query[A] 3x6-.13-.rJ0ewD9ex*qn
  ↪ plPERUC73Q1Qxmzgt6NsKfC0O59uXT1mPP0-.TBtcHpG3t74wl59N1qz5
  ↪ ubwr2IMluIZbAyW0X0kAmlXvnOE-.0TJUevQqKVZ4qjkCWhnWYQvCWgg*
  ↪ cJCkNjrEsVmkZnlPEDh-.RL89jhTomspiX*9GpkT4mPrwAbe1gZpclA*q
  ↪ fEG9oLKwdUU-.KelleyPlc.docx.wtulqzvb.at from
  ↪ 172.16.0.187
5 Mar 31 23:30:00 dnsmasq[3438]: forwarded 3x6-.13-.rJ0ewD9ex*q
  ↪ nplPERUC73Q1Qxmzgt6NsKfC0O59uXT1mPP0-.TBtcHpG3t74wl59N1qz
  ↪ 5ubwr2IMluIZbAyW0X0kAmlXvnOE-.0TJUevQqKVZ4qjkCWhnWYQvCWgg
  ↪ *cJCkNjrEsVmkZnlPEDh-.RL89jhTomspiX*9GpkT4mPrwAbe1gZpclA*
  ↪ qfEG9oLKwdUU-.KelleyPlc.docx.wtulqzvb.at to
  ↪ 192.42.52.58
6 Mar 31 23:30:01 dnsmasq[3438]: query[A] 3x6-.13-.rJ0ewD9ex*qn
  ↪ plPERUC73Q1Qxmzgt6NsKfC0O59uXT1mPP0-.TBtcHpG3t74wl59N1qz5
  ↪ ubwr2IMluIZbAyW0X0kAmlXvnOE-.0TJUevQqKVZ4qjkCWhnWYQvCWgg*
  ↪ cJCkNjrEsVmkZnlPEDh-.RL89jhTomspiX*9GpkT4mPrwAbe1gZpclA*q
  ↪ fEG9oLKwdUU-.KelleyPlc.docx.wtulqzvb.at from
  ↪ 172.16.0.187
```

Listing 7.1: Scenario 2 - DNSMASQ Log Excerpt

Nowadays labeling of massive datasets for machine learning or similar applications often utilizes so called crowdsourcing. Brabham [Bra13, p.xix] defines „crowdsourcing as an online, distributed problem-solving and production model that leverages the collective

intelligence of online communities...“. In practice crowdsourcing is often used to solve simple problems (e.g., label images) by leveraging large numbers of laypeople. Using such an approach it is possible to solve these problems even for very large datasets at very low cost.

The problem we are trying to solve are trying to solve is the verification of the correctness of our assigned labels. This is a problem that requires experts trained in the field of IT security. There also exist crowdsourcing models and approaches that work under the assumption that the to be solved problem requires a community of experts rather than a community of laypeople. The crowdsourcing strategy proposed by Retelny et al. [Ret+14] involves hiring so called flash teams of problem domain experts. Thus would we use such an approach for our expert evaluation we would have to hire a number of IT security experts. Due to resource limitations this is not feasible for this work [Bra13; KOS13; Ret+14].

Thus we are only left with the option of relying on a small number of IT security expert volunteers, that are willing to assist us in our evaluation without receiving compensation. Such volunteers are hard to find and as they are volunteers they are also more likely to be deterred by large numbers of log lines to review. Therefore it is imperative for our evaluation to limit the amount of log lines to be reviewed by our experts as much as possible.

Looking at the raw numbers we have 408038 labeled log lines for scenario 1 and 43303 for scenario 2. For scenario 1 407660 of the logs are labeled as *dirb*, this means that we only have 378 scenario 1 log lines which have been assigned to the other labels. Out of these only 42 are assigned labels we have selected for the evaluation. The remaining 336 log lines have all been assigned the label *attacker_vpn*, which we previously excluded from our evaluation process. Therefore it is reasonable for us to sample only a few lines for those labels.

In case of the *dirb* and *dnsteal* labels we have a large amount of labeled log lines, but it can be argued that these log lines are very uniform. For example, *dirb* labeled log lines are mostly HTTP 404 - Not Found log entries in the Apache access or error log. There are only a few exceptions to this. For the *dnsteal* the labled log lines are even less varied, there are only 3 types of log lines which are assigned this label. These are queries, forwards and replies for the attacker controlled sub domain (see listing 7.1). Thus we argue that while we cannot sufficiently cover these two labels in terms of size a small sample size should still be a reasonably represented set of labeled log lines.

Keeping the above mentioned factors in mind we decided to sample 8 log lines per selected label. Half of the sampled lines per label were sampled from the set of labeled log lines and the other half were sampled from unlabeled log lines from the same log files as the labeled lines. Note that some labels have only been assigned to a single log line (e.g., *reverse_shell* and *exfiltration-service*). For these we sample a total of three lines: the labeled line and 2 unlabeled lines. Furthermore, we skip the *Incorrect Labeled* (IL) question category (see section 7.2 for details) for these labels, since we only have one

labeled log line. With this sampling strategy we have a total of 54 log lines to review, 35 for scenario 1 and 19 for scenario 2.

7.2 Expert Evaluation Format

As discussed above we have sampled a total of 54 log lines for our expert evaluation. Half of the log lines were sampled from labeled log lines and the other half were sampled from unlabeled log lines. The unlabeled log lines are assigned the artificial label *normal*. For our expert evaluation we further split our sampled log lines into four question categories:

- **Correct Labeled (CL):** These are labeled log line samples that have the label assigned by our rules.
- **Incorrect Labeled (IL):** These are labeled log line samples for which we remove the label and instead assign the artificial *normal* label we assume to be incorrect.
- **Correct Unlabeled (CU):** These are unlabeled log line samples we do not modify, i.e., the assigned artificial *normal* remains unchanged.
- **Incorrect Unlabeled (IU):** These are unlabeled log line samples for which we do not assign the artificial *normal* label, but instead incorrectly assign one of the sampled labels (e.g., *dnsteal*).

Categorization is done based on the sampling groups, i.e., unlabeled logs sampled for the *dnsteal* labels will be assigned the incorrect label *dnsteal*, if they are selected for the *IU* category. Regardless of question category the question asked of the experts will always be the same: "Is this log line labeled correctly?". For questions categories *CL* and *CU* we expect the reviewers to answer positively and for *IL* and *IU* questions we expect them to answer the question negatively.

The four categories were deliberately and sampling chosen to ensure that there is no bias towards one answer type (i.e., agreement vs disagreement). This way questions are equally distributed across the two different sampling source types, labeled log entries and unlabeled entries, and the expected answers.

For each question reviewers have 7 answer options with 3 varying levels of agreement or disagreement (e.g., Strongly Agree or Somewhat Disagree) and the "No Answer" option. Note that these levels were deliberately chosen so that a reviewer must either agree or disagree with a label. The "No Answer" option is interpreted as the reviewer not being able to decide the question. This might happen if a reviewer feels that they do not have enough information for their decision or are simply unsure.

To conduct the expert evaluation we utilize the survey platform LimeSurvey[1]. We configured LimeSurvey to group the questions into two groups one for each scenario.

[1] https://www.limesurvey.org/

Questions within a group and the order of the question groups themselves are fully randomized. For each question we provide the reviewers with the log file, line, assigned label and the 5 log lines before and after the line being reviewed. Additionally we also reference other possibly relevant logs on the same host system. These references also include the line numbers of the log entries with the timestamps closest to the timestamp of the log line that is being reviewed. Figure 7.1 shows an evaluation question. The figure shows the text noting the line to be reviewed and its assigned label at the top. This is followed by the actual log line and its preceding and subsequent lines. Lastly the question shows a list of log files with associated line numbers that are likely relevant to the reviewer of this survey question.

3 The following are log lines taken from **scenario_1_rce/servers/intranet_server/logs/apache2/intranet.company.cyberrange.at-access.log.4** line **68165** (marked in red) has been labeled as **dirb** please choose if this is correct or not:

```
68161 - 172.16.100.151 - - [23/Mar/2021:20:40:03 +0000] "GET /wp-content/themes/embed HTTP/1.1" 404 363 "-" "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.1)"
68162 - 172.16.100.151 - - [23/Mar/2021:20:40:03 +0000] "GET /wp-content/themes/embedd HTTP/1.1" 404 363 "-" "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.1)"
68163 - 172.16.100.151 - - [23/Mar/2021:20:40:03 +0000] "GET /wp-content/themes/embedded HTTP/1.1" 404 363 "-" "Mozilla/4.0 (compat-
ible; MSIE 6.0; Windows NT 5.1)"
68164 - 172.16.100.151 - - [23/Mar/2021:20:40:03 +0000] "GET /wp-content/themes/emea HTTP/1.1" 404 363 "-" "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.1)"
68165 - 172.16.100.151 - - [23/Mar/2021:20:40:03 +0000] "GET /wp-content/themes/emergency HTTP/1.1" 404 363 "-" "Mozilla/4.0 (compat-
ible; MSIE 6.0; Windows NT 5.1)"
68166 - 172.16.100.151 - - [23/Mar/2021:20:40:03 +0000] "GET /wp-content/themes/emoticons HTTP/1.1" 404 363 "-" "Mozilla/4.0 (compat-
ible; MSIE 6.0; Windows NT 5.1)"
68167 - 172.16.100.151 - - [23/Mar/2021:20:40:03 +0000] "GET /wp-content/themes/employee HTTP/1.1" 404 363 "-" "Mozilla/4.0 (compat-
ible; MSIE 6.0; Windows NT 5.1)"
68168 - 172.16.100.151 - - [23/Mar/2021:20:40:03 +0000] "GET /wp-content/themes/employees HTTP/1.1" 404 363 "-" "Mozilla/4.0 (compat-
ible; MSIE 6.0; Windows NT 5.1)"
68169 - 172.16.100.151 - - [23/Mar/2021:20:40:03 +0000] "GET /wp-content/themes/employers HTTP/1.1" 404 363 "-" "Mozilla/4.0 (compat-
ible; MSIE 6.0; Windows NT 5.1)"
```

Other log files from this host:

- **scenario_1_rce/servers/intranet_server/logs/apache2/intranet.company.cyberrange.at-error.log.2** with timestamp **2021-03-23T20:40:00.539Z** on line **295**
- **scenario_1_rce/servers/intranet_server/logs/auth.log** with timestamp **2021-03-23T20:39:01.000Z** on line **129**
- **scenario_1_rce/servers/intranet_server/logs/syslog.4** with timestamp **2021-03-23T20:39:22.000Z** on line **174**
- **scenario_1_rce/servers/intranet_server/logs/audit/audit.log** with timestamp **2021-03-23T20:39:22.711Z** on line **481**

	Strongly Disagree	Disagree	Somewhat Disagree	Somewhat Agree	Agree	Strongly Agree	No answer
Line 68165 is labeled as "dirb" is this label correct?	○	○	○	○	○	○	●

Figure 7.1: Example Evaluation Question

In addition to the information provided as part of the question we also provide detailed information about the attack scenarios at the start of each question group. We also provide a dataset archive containing all log files and testbed fact information for both scenarios. Additionally we also provide summarized attacker and server fact information in an effort to make it easier for reviewers to find the most important information. Listing 7.2 shows part of one of these information files. Within this file the reviewer can view the attack timeline and most important host facts at a glance.

```yaml
attacker_0:
  hostname: attacker-0
  distribution: Ubuntu
  distribution_release: bionic
  distribution_version: '18.04'
  ipv4_address: 192.42.1.190
  ipv6_address: fe80::f816:3eff:fe3d:82aa
  timezone: UTC
  # the domain used during exfiltration
  attack_domain: "wtulqzvb.at"
# attack timeline
timeline:
  dnsteal:
      # time the attacker server and exfiltration process first started
      # this is before the server logs observation time frame
      start: "2021-03-29T12:28:30.371Z"
      # time the attacker server stopped accepting DNS queries
      stop: "2021-03-31T23:30:00.281Z"

  # exfiltration service name and time it stopped
  # due to lost connection to the attacker server
  exfiltration_service:
    name: "wallflower "
    stop: "2021-04-01T00:38:33.148Z"
inet-firewall:
  hostname: inet-firewall
  distribution: Ubuntu
  distribution_release: bionic
  distribution_version: '18.04'
  default_ipv4_address: 192.42.165.180
  default_ipv6_address: fe80::f816:3eff:febb:c933
  ipv4_addresses:
    - '192.42.165.180' # internet port
    - '172.16.0.1' # intranet port
    - '172.16.100.1' # DMZ port
  ipv6_addresses:
    - fe80::f816:3eff:febb:c933 # internet ipv6
    - fe80::f816:3eff:fe51:7536 # intranet ipv6
    - fe80::f816:3eff:feca:14fd # DMZ ipv6
```

Listing 7.2: Scenario 2 - Attacker Information

7.3 Results

The expert survey was made available to a group of volunteering cyber security specialists using the survey platform LimeSurvey[2]. Responses of the expert evaluation were collected from 2021.04.19 until 2021.05.05.

7.3.1 Participation

For the expert survey we contacted some cyber security specialists, e.g., security analysts and penetration testers from leading cyber security companies directly. We also shared our survey in the group chat of a notable Austrian Capture the Flag (CTF) team. Experts contacted directly by us were also encouraged to share the survey with others in the cyber security community.

Until the final day of the expert survey we received a total of 16 participant responses through LimeSurvey. This number includes anyone that visited the survey page and proceeded past the initial welcome page, which was used to explain the survey format. Out of the total 16 responses 10 reviewers completed the survey and 6 did not submit their final responses. Note that surveys are considered completed if the reviewer pressed the final *submit* button.

To ensure the quality of responses considered for the evaluation, we decided to only include responses from participants that skipped less than 25% of the questions. For this expert survey a skipped question can mean one of two things. The reviewer either opted to not answer the question, because they felt that they did not have enough information to make a decision. Another reason could be that the participant did not have enough time available to finish the survey and thus skipped a series of questions completely. Out of the total 16 recorded participants 8 participants meet this requirement.

At the beginning of the survey we asked participants to select a role description that best fits their current occupation. The 8 participants that fulfilled our selection criteria have assigned themselves to three roles. The majority (5) selected the role *security analyst*, two identified as *penetration testers* and one as *cyber security research engineer*. Since our participant pool is quiet small we will refrain from referencing the selected roles throughout the result discussion.

7.3.2 Consensus

For our reduced survey results we have received an average of 7.828571 and 7.894737 responses per question in scenario 1 and 2 respectively. As we only consider 8 participants, this indicates that the remaining 8 participants only skipped very few questions, in fact, they skipped only $1,85\%$ on average. The response data also shows that every question has received at least 7 responses, meaning that no two or more participants skipped the same question. This might indicate that the participants were all very confident in their

[2] [**https://www.limesurvey.org/**]

ability to judge if labels were correctly assigned and that there was no single question that was hard to decide for everyone.

For further evaluation of our survey result we created box plots of the answers for each question. Figures 7.2 to 7.9 show these box plots grouped by scenario and by the four question classes we defined. Note that for *CL* and *CU* agreement to the assigned label is expected, while disagreement is expected for the *IL* and *IU* classes for which we purposefully inverted the labels assigned by the Cyber Range Kyoushi framework. In addition to the box plots the figures also show individual responses visualized by the blue x-marks.

For the purpose of the box plots we converted the 6 decision options to the following numerical values:

- **Strongly Agree:** 3
- **Agree:** 2
- **Somewhat Agree:** 1
- **Somewhat Disagree:** -1
- **Disagree:** -2
- **Strongly Disagree:** -3

The plots can be interpreted as follows: Boxes that span only either agreement or disagreement options indicate a general consensus among the participating cyber security experts. This consensus is stronger the further the box leans towards either the top (agreement) or the bottom (disagreement). Note that differences in personality between participants can influence this, e.g., participants that are inclined to be more cautious tend to select answer options closer to the middle. X-marks which are outside the boxes and their whiskers are considered outliers, i.e., they divert from the consensus formed by the other participants.

Consensus formed by the experts matches the labels assigned by our framework, if the consensus agrees with the assigned labels for *CL* and *CU* type questions or disagrees for *IL* and *IU* type questions. Figures 7.2 to 7.9 show that first there is a consensus for most of the survey questions and second that the expert consensus matches the decisions of the labeling rules that were defined and applied with the Cyber Range Kyoushi framework. Consensus on this is especially strong for scenario 2. For example, fig. 7.8 shows the box plots for the incorrectly labeled (IL) log lines of scenario 2. With the exception of three outliers all responses *strongly disagree* with the presented labeling, i.e., the expected result as *IL* questions show log lines for which assigned attack labels were purposefully replaced with the artificial normal label.

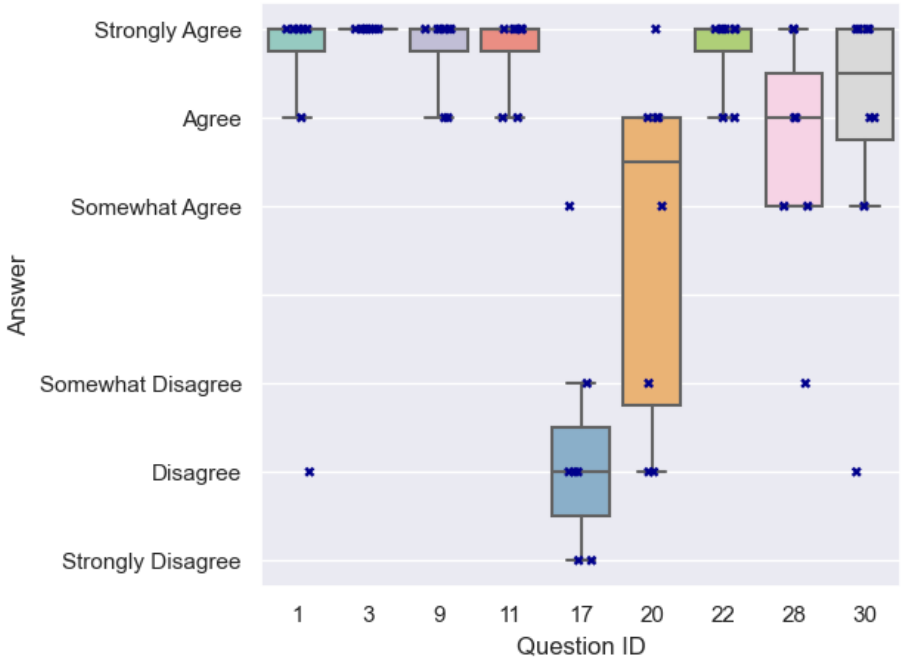

Figure 7.2: Scenario 1 - Correct Label (CL) Questions

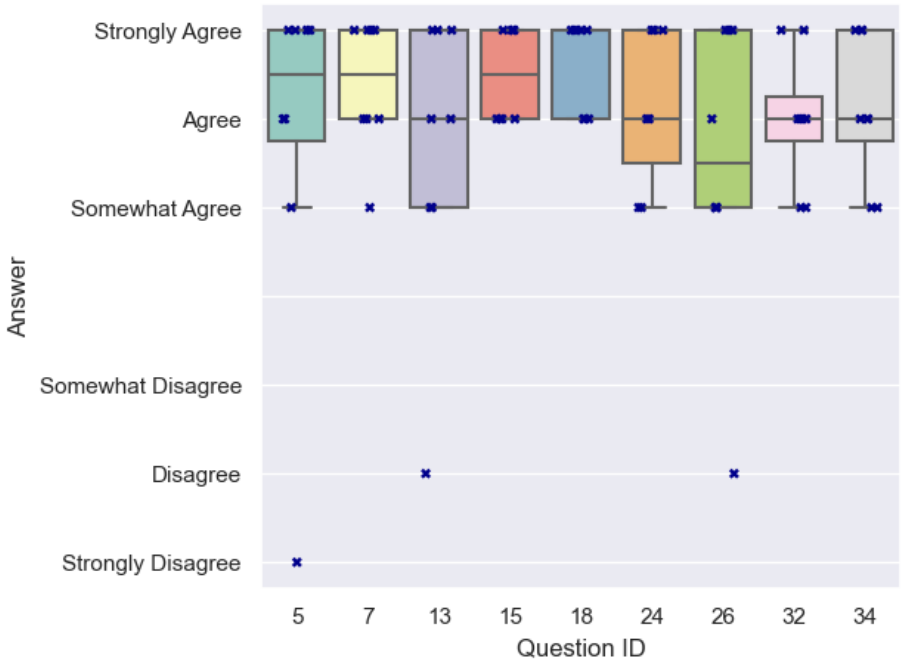

Figure 7.3: Scenario 1 - Correct Unlabeled (CU) Questions

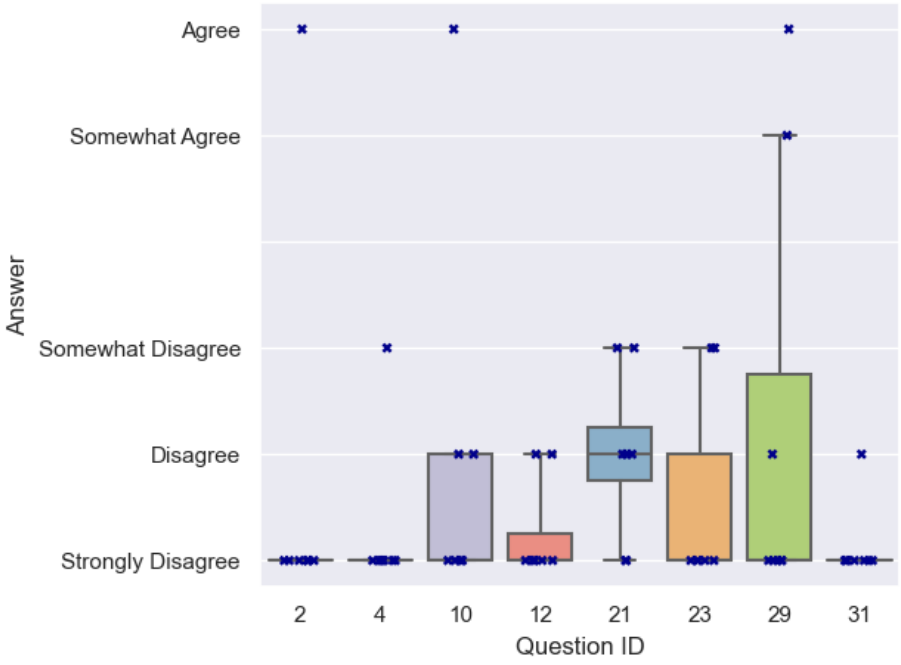

Figure 7.4: Scenario 1 - Incorrect Label (IL) Questions

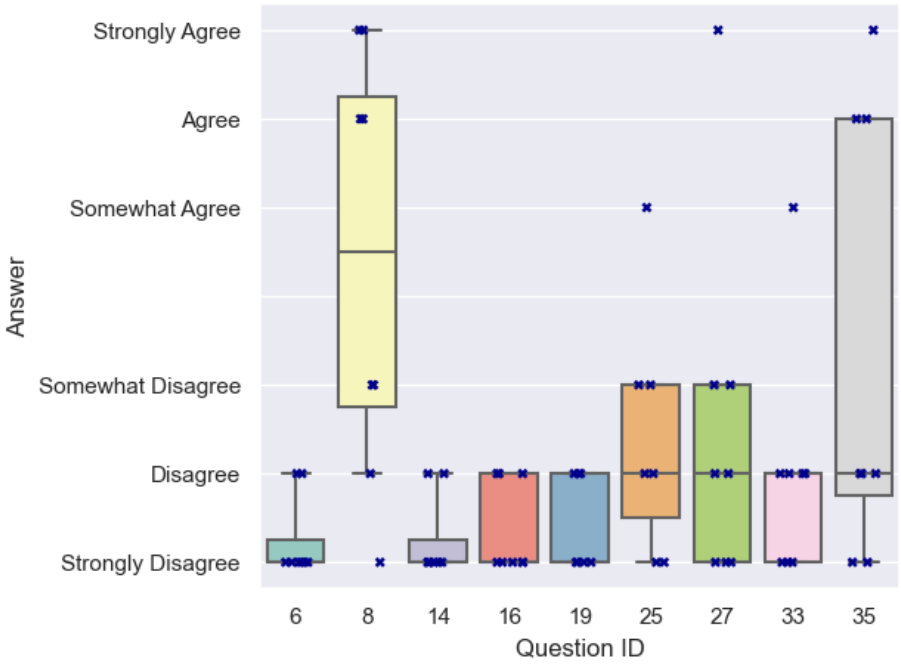

Figure 7.5: Scenario 1 - Incorrect Unlabeled (IU) Questions

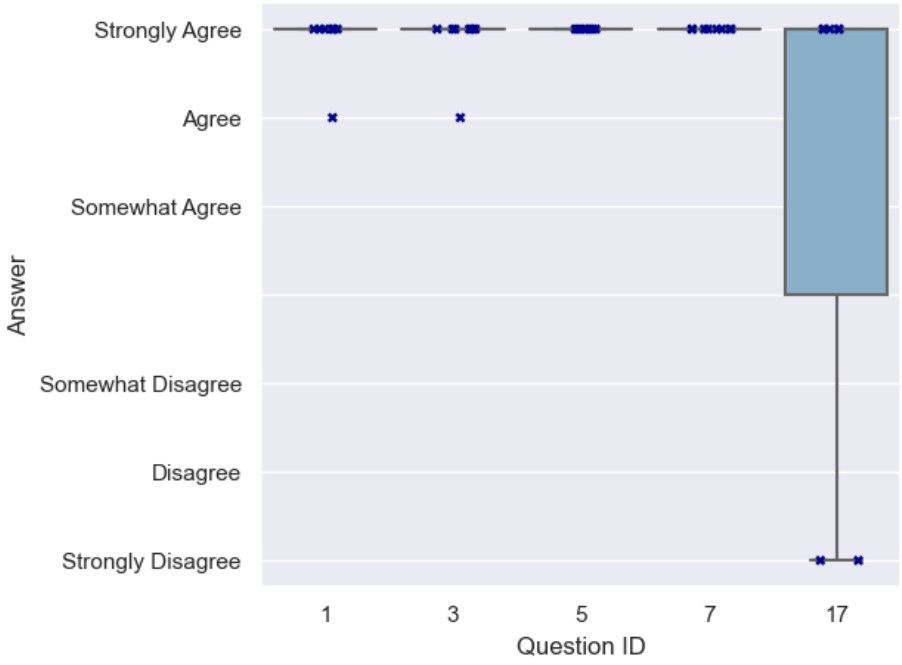

Figure 7.6: Scenario 2 - Correct Label (CL) Questions

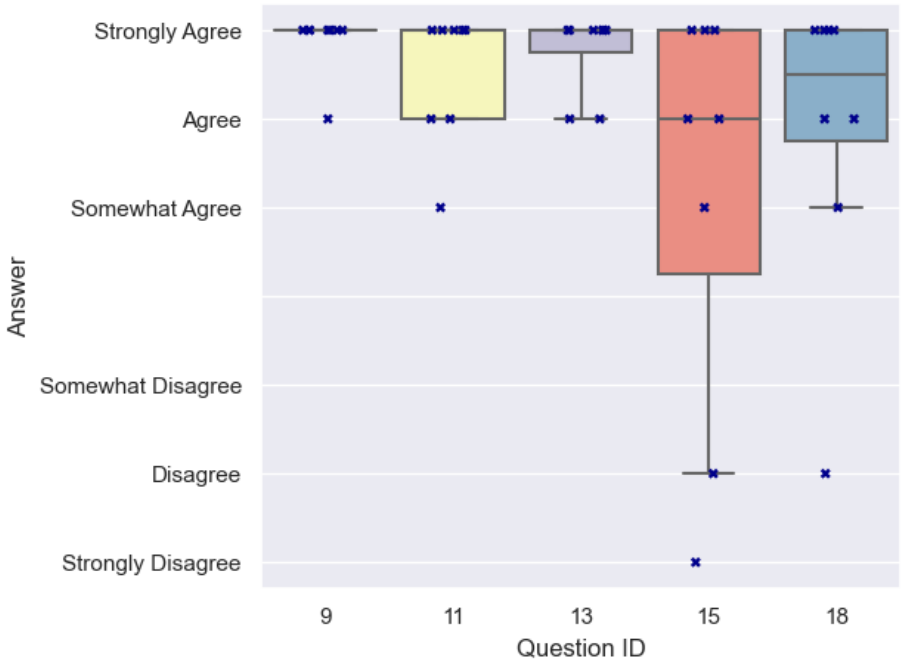

Figure 7.7: Scenario 2 - Correct Unlabeled (CU) Questions

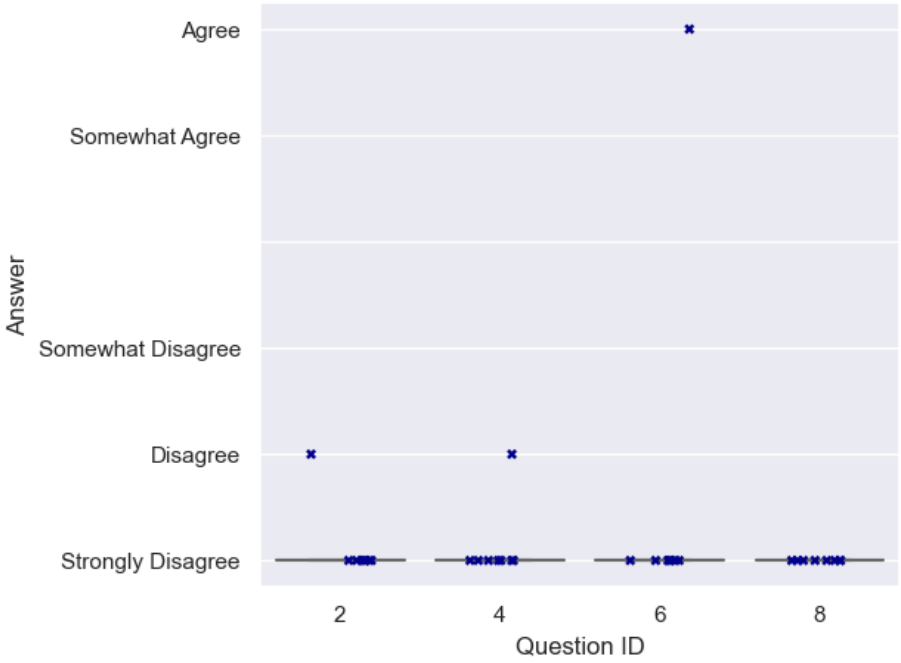

Figure 7.8: Scenario 2 - Incorrect Label (IL) Questions

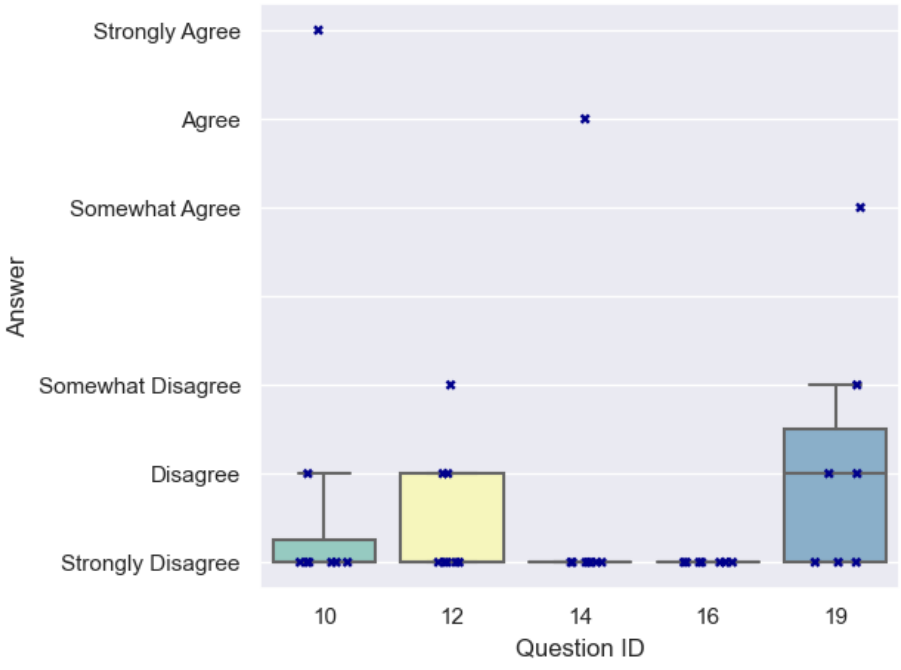

Figure 7.9: Scenario 2 - Incorrect Unlabeled (IU) Questions

7.3.3 Outliers

Below we further discuss survey questions for which no consensus was formed by the experts, i.e., the box plots span across both agreement and disagreement options, and questions for which the labels assigned using the Cyber Range Kyoushi framework do not match the expert consensus. Note that question order was fully randomized for each expert reviewers, so question ids do not indicate any order.

Scenario 1

Question 17 (CL). The labeling rules defined for scenario 1 assigned the *reverse_shell* label to line 3 shown in listing 7.3. Since this is a *CL* question we would expect the consensus to agree with the label assignment presented by the questions, but as can be seen in fig. 7.2 expert consensus disagrees with the label assigned using the Cyber Range Kyoushi framework.

A thorough manual investigation of the attack scenario and logs shows that in this case labeling rules have in fact applied the correct label. The log line is the direct result of executing the reverse shell command through the web shell. The used command uses a particular type of bash redirection to redirect *stdout* and *stdin* through the reverse shell connection resulting in this `Bad file descriptor` error. We suspect that the expert consensus disagrees with the label for two reasons. One, the log line does not have a time stamp and thus cannot be easily attributed to the attack timings. Surrounding log lines only indicate that the log line was written within the same day as the escalation phase. Second, this is not a very common error message and could also easily be caused by a incorrectly configured file reference within the web application.

```
1 [Thu Mar 25 06:25:06.459512 2021] [mpm_prefork:notice] [pid 21629] AH00163: Apache
      /2.4.29 (Ubuntu) OpenSSL/1.1.1 configured -- resuming normal operations
2 [Thu Mar 25 06:25:06.459563 2021] [core:notice] [pid 21629] AH00094: Command line: '/
      usr/sbin/apache2'
3 bash:  196:  Bad file descriptor
4 [Fri Mar 26 06:25:04.228393 2021] [mpm_prefork:notice] [pid 21629] AH00171: Graceful
      restart requested, doing restart
```

Listing 7.3: Question 17 - Log Snippet

Question 20 (CL). This question only has a weak agreement consensus, i.e., the majority of responses *Agree* with the presented label, but only one participants *Strongly Agrees* and three participant indicated that they disagree (2) or somewhat disagree (1).

A reason for this result might be that unparsed audit logs are generally harder to read than other log types as the log format is designed to be machine readable rather than human readable, e.g., timestamps are in unix epoch format. Also as shown in listing 7.4 the log snippet shown for this question contains only two indicators for malicious activity, the first two lines (1608 and 1609) showing that the change user events originate from the Apache web service user *www-data* (uid 33). This makes this question very hard to decide without referencing the additional attack scenario and testbed information provided.

```
1608 type=USER_ACCT msg=audit(1616704612.054:1710): pid=7916 uid=33 auid=4294967295 ses
         =4294967295 msg='op=PAM:accounting acct="msperl" exe="/bin/su" hostname=? addr=?
         terminal=/dev/pts/1 res=success'
1609 type=CRED_ACQ msg=audit(1616704612.074:1711): pid=7916 uid=33 auid=4294967295 ses
         =4294967295 msg='op=PAM:setcred acct="msperl" exe="/bin/su" hostname=? addr=?
         terminal=/dev/pts/1 res=success'
1610 type=USER_ACCT msg=audit(1616704612.118:1712): pid=7917 uid=0 auid=4294967295 ses
         =4294967295 msg='op=PAM:accounting acct="msperl" exe="/lib/systemd/systemd"
         hostname=? addr=? terminal=? res=success'
1611 type=CRED_ACQ msg=audit(1616704612.126:1713): pid=7917 uid=0 auid=4294967295 ses
         =4294967295 msg='op=PAM:setcred acct="msperl" exe="/lib/systemd/systemd" hostname
         =? addr=? terminal=? res=success'
1612 type=LOGIN msg=audit(1616704612.126:1714):  pid=7917 uid=0 old-auid=4294967295
         auid=1001 tty=(none) old-ses=4294967295 ses=247 res=1
1613 type=USER_START msg=audit(1616704612.126:1715): pid=7917 uid=0 auid=1001 ses=247 msg='
         op=PAM:session_open acct="msperl" exe="/lib/systemd/systemd" hostname=? addr=?
         terminal=? res=success'
1614 type=USER_END msg=audit(1616704612.150:1716): pid=6845 uid=0 auid=1002 ses=227 msg='op=
         PAM:session_close acct="ffeichtner" exe="/usr/sbin/sshd" hostname=172.16.100.151
         addr=172.16.100.151 terminal=ssh res=success'
1615 type=CRED_DISP msg=audit(1616704612.150:1717): pid=6845 uid=0 auid=1002 ses=227 msg='op
         =PAM:setcred acct="ffeichtner" exe="/usr/sbin/sshd" hostname=172.16.100.151 addr
         =172.16.100.151 terminal=ssh res=success'
1616 type=USER_END msg=audit(1616704612.178:1718): pid=6989 uid=0 auid=1002 ses=227 msg='op=
         PAM:session_close acct="root" exe="/usr/bin/sudo" hostname=? addr=? terminal=/dev/
         pts/0 res=success'
```

Listing 7.4: Question 20 - Log Snippet

Question 8 (IU). The log line for this question was sampled from the space of unlabeled log lines and presented with the incorrect label *dirb* (*IU* question). The boxplot shown in fig. 7.5 indicates that question 8 does not have a clear consensus. We theorize that this is due to the nature of this specific question.

The incorrectly assigned *dirb* label is applied to all log lines resulting from the directory scan phase. Directory scans usually generate large amounts of file not found errors and access violations. The question's log line, shown in listing 7.5, contains a PHP warning caused by a missing directory. While the timestamp for this log line is outside directory scan's attack time it would still be very easy to incorrectly attribute this warning to a request sent during the attack in the spur of the moment.

```
1 [Thu Mar 25 08:26:36.164484 2021] [php7:warn] [pid 5416] [client 172.16.0.217:35122]
      PHP Warning:  scandir(/var/www/intranet.company.cyberrange.at/wp-content/uploads/
      wpdiscuz/cache/gravatars/):  failed to open dir:  No such file or directory in
      /var/www/intranet.company.cyberrange.at/wp-content/plugins/wpdiscuz/utils/
      class.WpdiscuzCache.php on line 190
2 [Thu Mar 25 08:26:36.164707 2021] [php7:warn] [pid 5416] [client 172.16.0.217:35122]
      PHP Warning:  scandir(): (errno 2): No such file or directory in /var/www/intranet
      .company.cyberrange.at/wp-content/plugins/wpdiscuz/utils/class.WpdiscuzCache.php
      on line 190
```

Listing 7.5: Question 8 - Log Snippet

Question 35 (IU). For this question we only have a slim majority (5 out of 8) consensus that disagrees with the label presented in the question. Since this is an *IU* question this means that the slim majority agrees with the label assignment provided by the Cyber Range Kyoushi framework.

For this question we actually expected that experts might either not be able to form a consensus or that the formed consensus would be contra to the label assignment. Our reasoning for this was that the particular log sample is of a section of the auth log that is very hard to decide without inspecting the entire log file provided in the dataset.

Listing 7.6 shows the log snippet provided to reviewers as part of the survey question. The log line in question (line 6) is the result of the SSH session of user *ffeichtner* terminating and thus closing the *systemd-logind* session for the user. The only other log lines shown, related to this SSH session, are lines 4 and 5. All other log lines are results of the attacker escalating from the web service user to the local system account *msperl* and executing privileged commands. This lack of information in the log snippet and the log lines proximity to a lot of log lines labeled as malicious makes this question very hard to answer correctly. Since this question is an incorrect unlabeled *IU* question, i.e., the log line was assigned an incorrect malicious label on purpose, it is especially easy for an expert reviewer to assume that the presented label is correct as most other log lines shown in the snippet are indeed malicious.

```
1  Mar 25 20:36:52 intranet-server su[7916]: + /dev/pts/1 www-data:msperl
2  Mar 25 20:36:52 intranet-server su[7916]: pam_unix(su:session): session opened for user
       msperl by (uid=33)
3  Mar 25 20:36:52 intranet-server systemd: pam_unix(systemd-user:session): session opened
       for user msperl by (uid=0)
4  Mar 25 20:36:52 intranet-server sshd[6845]: pam_unix(sshd:session): session closed for
       user ffeichtner
5  Mar 25 20:36:52 intranet-server sudo: pam_unix(sudo:session): session closed for user
       root
6  Mar 25 20:36:52 intranet-server systemd-logind[1044]:  Removed session 227.
7  Mar 25 20:37:00 intranet-server sudo:   msperl : TTY=pts/1 ; PWD=/var/www/intranet.
       company.cyberrange.at/wp-content/uploads/2021/03 ; USER=root ; COMMAND=list
8  Mar 25 20:37:04 intranet-server sudo:   msperl : TTY=pts/1 ; PWD=/var/www/intranet.
       company.cyberrange.at/wp-content/uploads/2021/03 ; USER=root ; COMMAND=/bin/ls -
       laR /root/
9  Mar 25 20:37:04 intranet-server sudo: pam_unix(sudo:session): session opened for user
       root by (uid=0)
10 Mar 25 20:37:04 intranet-server sudo: pam_unix(sudo:session): session closed for user
       root
11 Mar 25 20:37:05 intranet-server sudo:   msperl : TTY=pts/1 ; PWD=/var/www/intranet.
       company.cyberrange.at/wp-content/uploads/2021/03 ; USER=root ; COMMAND=/bin/cat /
       etc/shadow
```

Listing 7.6: Question 35 - Log Snippet

CHAPTER 8

Conclusion and Future Work

This thesis introduced a list of requirements for frameworks using cyber range technologies to automatically generate and label datasets for Intrusion Detection Systems. Based on these requirements we further introduced Cyber Range Kyoushi, a framework and methodology for generating and labeling log datasets as commonly used by Intrusion Detection System (IDS) (*RQ3*). The proposed framework builds on top of previous work by Landauer et al. [Lan+20b] on the topic of model-driven testbed simulation for generating IDS log datasets. The log dataset generation methodology proposed by Landauer et al. [Lan+20b] enables rapid generation of datasets by using parameterized cyber range testbed and attack scenario models. However, labeling of generated datasets is a major bottleneck of this methodology. The Cyber Range Kyoushi framework extends the methodology discussed by Landauer et al. [Lan+20b] and defines and implements a labeling approach that is easy to use and is able to automatically adapt to parameter changes in the testbed and attack models.

The Cyber Range Kyoushi framework was proposed and implemented as a system working on four different layers: *Model*, *Testbed*, *Data Collection* and *Processing*. Each of the layers is responsible for specific aspects of the dataset generation and labeling process. Within the context of the framework we define two types of simulation artifacts: *testbed facts* and *logs*. The Cyber Range Kyoushi dataset generation and labeling methodology utilizes extensive logging of simulation agents (i.e., attackers and benign users) and thorough cyber range testbed fact gathering to build a knowledge base that can be used as input for the labeling process (*RQ2*). The raw dataset is also parsed and stored in a database that supports a wide range of analytical queries. Labeling is then performed in an extensible multi stage process, with each stage having access to the results of all prior stages. The labeling process uses labeling rules; special configurable decision functions that can apply labels to the dataset by reasoning on the knowledge base and parsed dataset. By supporting configuration and labeling rule templates at the model layer

the framework is also able to cope with statically and dynamically instantiated testbed, attack and user models.

To demonstrate the capabilities of the framework a cyber range testbed and two distinct attack scenario models were designed. Both scenarios were then implemented and configured using the Cyber Range Kyoushi framework's methodology and software components. Two labeled datasets were then created by simulating the implemented scenarios and applying the Cyber Range Kyoushi labeling process. Using the Cyber Range Kyoushi framework we were able to create very precise and detailed labels, i.e., the applied labels mapped to specific steps of the attack scenario. We further showed that labeling for both scenarios could be achieved using only a few templated labeling rules defined as part of the attack model. For this a total of only four labeling rule types were used. All four rule types only used simple analytical query techniques for their reasoning, showing that complex attack scenarios can be labeled using the proposed methodology. More complex labeling rule types, e.g., anomaly detection classifiers or complex statistical models, could be added in future versions of the framework to provide even more flexibility.

Furthermore the quality of the labels for two generated datasets was validated through a cyber security expert survey. For the survey discussed in evaluation chapter of this work log entries for both datasets were sampled and questions about the correctness of labels assigned to the sampled logs were posed to survey participants. For this we defined a sampling strategy based on the assigned labels and attack scenario definitions. Samples were drawn from both the sets of labeled and unlabeled log lines equally. The sample size was carefully chosen by evaluating the properties of labeled log lines. Sampled log lines were further split into four question categories to avoid answer biases (*RQ5*). The evaluation of the survey responses showed that cyber security experts agree with the label assignment for these two datasets provided by the Cyber Range Kyoushi framework, thus showing that the proposed framework is not only able to generate and label log datasets for Intrusion Detection Systems, but is also able to do so correctly. That is, by correctly defining templated labeling rules during the model definition it is possible to produce correct and precise labels (*RQ1, RQ4*).

For the expert survey we recorded a total of 16 volunteer participants of which only 8 answered enough survey questions so that their answers could be included in the evaluation process. While each participants was an expert in the field of cyber security and thus provided highly valuable input, low participation rates can have negative effects even on qualitative studies such as the one that was conducted. As such, for future work we might want to consider alternative evaluation approaches. One simple option would be moving from volunteers to paid expert reviewers to ensure a certain level of participation, but this is only feasible for works with significant resources as cyber security professionals are usually highly paid. Instead of using paid experts changes could be made to the way volunteers are motivated to participate in the evaluation process. For example, gamification could be used by designing the expert review in the form a Capture the Flag (CTF) like event or by having visitors of an industry conference answer a few survey

questions in exchange for a chance to win a price.

Funding. This thesis has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 833456, as part of the Guard project.

List of Figures

List of Tables

List of Listings

Acronyms

API Application Programming Interface. 46, 60, 61

APT Advanced Presistent Threat. 35

CIDR Classless inter-domain routing. 33

CLI Command Line. 45–48, 54, 55, 58, 59

CPS Cyber-Physical Systems. 1

CTF Capture the Flag. 96, 106

CVE Common Vulnerabilities and Exposures. 34

DMZ Demilitarized Zone. 13, 33

DNAT Dynamic Network Address Translation. 15, 78

DNS Domain Name System. 14, 15, 33, 35, 36

DOM Domain Object Model. 22, 24, 25

DSL Domain-Specific Language. 59, 61, 63, 74

FQDN Fully Qualified Domain Name. 72, 84

HIDS Host Intrusion Detection System. 1, 3, 4, 10, 31, 37–39

HSTS HTTP Strict Transport Security. 25

HTTP Hyper Text Transfer Protocol. 15, 22, 25, 58, 61, 62, 78–81, 83, 87, 91

HTTPS Hyper Text Transfer Protocol Secure. 15, 25

ICT Information and communications technology. 2

IDS Intrusion Detection System. ix, xi, 1, 4, 5, 7–13, 27, 31, 32, 34, 35, 37–40, 43, 46, 54, 74, 105, 106

IMAP Internet Message Access Protocol. 15

IoT Internet of Things. 1

IPS Intrusion Prevention System. 7

NAT Network Address Translation. 15

NHIDS Network and Host Intrusion Detection System. 10, 38, 50

NIDS Network Intrusion Detection System. 1, 10, 12, 38

NIST National Institute of Standards and Technology. 8, 9

OT Operational Technology. 2

SMTP Simple Mail Transfer Protocol. 15

SQL Structured Query Language. 60

SSH Secure Shell. 18, 22, 28, 34, 104

UI User Interface. 22

URI Uniform Resource Identifier. 72, 78, 80, 81

URL Uniform Resource Locator. 24, 25, 63

VPN Virtual Private Network. 15, 16, 31, 33, 34, 74, 77, 78, 81, 83, 88

Bibliography

[Ahm20] Tabrez Ahmad. „Corona virus (covid-19) pandemic and work from home: Challenges of cybercrimes and cybersecurity“. In: *Available at SSRN 3568830* (2020).

[Ahm+20] J. Ahmed et al. „Monitoring Enterprise DNS Queries for Detecting Data Exfiltration From Internal Hosts“. In: *IEEE Transactions on Network and Service Management* 17.1 (2020), pp. 265–279. DOI: 10.1109/TNSM.2019.2940735.

[Alm+17] Mohammad Almseidin et al. „Evaluation of machine learning algorithms for intrusion detection system“. In: *2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY)*. IEEE. 2017, pp. 000277–000282.

[Alm+18] Hamad Almohannadi et al. „Cyber Threat Intelligence from Honeypot Data Using Elasticsearch“. In: *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*. 2018, pp. 900–906. DOI: 10.1109/AINA.2018.00132.

[Ans21] Ansible. *Discovering variables: facts and magic variables — Ansible Documentation*. publisher: Ansible. 2021. URL: https://docs.ansible.com/ansible/latest/user_guide/playbooks_vars_facts.html (visited on 05/01/2021).

[Ark+21] Jari Arkko et al. *Report from the IAB COVID-19 Network Impacts Workshop 2020*. Internet-Draft draft-iab-covid19-workshop-01. Work in Progress. Internet Engineering Task Force, Feb. 2021. 23 pp. URL: https://datatracker.ietf.org/doc/html/draft-iab-covid19-workshop-01.

[BM01] Rebecca Bace and Peter Mell. *NIST special publication on intrusion detection systems*. Tech. rep. BOOZ-ALLEN and HAMILTON INC MCLEAN VA, 2001.

[Bra13] Daren C Brabham. *Crowdsourcing*. Mit Press, 2013.

[Buc+21] Molly Buchanan et al. *On Generating and Labeling Network Traffic with Realistic, Self-Propagating Malware*. 2021. arXiv: 2104.10034 [cs.CR].

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. „Anomaly detection: A survey“. In: *ACM computing surveys (CSUR)* 41.3 (2009), pp. 1–58.

[Cic+12] Paul Cichonski et al. *Computer Security Incident Handling Guide : Recommendations of the National Institute of Standards and Technology*. Tech. rep. NIST SP 800-61r2. National Institute of Standards and Technology, Aug. 2012, NIST SP 800–61r2. DOI: 10.6028/NIST.SP.800-61r2. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-61r2.pdf (visited on 04/20/2021).

[Col12] Eric Cole. *Advanced persistent threat: understanding the danger and how to protect your organization*. Newnes, 2012.

[Cve] *CVE-2020-24186*. Available from MITRE, CVE-ID CVE-2020-24186. Aug. 2020. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-24186 (visited on 03/30/2021).

[DCY16] Christian J D'Orazio, Kim-Kwang Raymond Choo, and Laurence T Yang. „Data exfiltration from Internet of Things devices: iOS devices as case studies". In: *IEEE Internet of Things Journal* 4.2 (2016), pp. 524–535.

[DL17] Ivan Dolnák and Ján Litvik. „Introduction to HTTP security headers and implementation of HTTP strict transport security (HSTS) header for HTTPS enforcing". In: *2017 15th International Conference on Emerging eLearning Technologies and Applications (ICETA)*. IEEE. 2017, pp. 1–4.

[Ela21a] Elastic. *EQL search*. Accessed on 29.04.2021. 2021. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/eql.html.

[Ela21b] Elastic. *Query DSL*. Accessed on 29.04.2021. 2021. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html.

[Emb20] Bryan Embrey. „The top three factors driving zero trust adoption". In: *Computer Fraud & Security* 2020.9 (Sept. 2020), pp. 13–15. DOI: 10.1016/s1361-3723(20)30097-x. URL: https://doi.org/10.1016/s1361-3723(20)30097-x.

[FLP17] Maximilian Frank, Maria Leitner, and Timea Pahi. „Design Considerations for Cyber Security Testbeds: A Case Study on a Cyber Security Testbed for Education". In: *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*. 2017, pp. 38–46. DOI: 10.1109/DASC-PICom-DataCom-CyberSciTec.2017.23.

[Fra21a] Maximilian Frank. *Cyber Range Kyoushi Thesis Experiment - Logstash Configuration*. May 2021. DOI: 10.5281/zenodo.4750917. URL: https://doi.org/10.5281/zenodo.4750917.

[Fra21b] Maximilian Frank. *Cyber Range Kyoushi Thesis Scenario 1 - Processing Config and Labeling Rules*. May 2021. DOI: 10.5281/zenodo.4750944. URL: https://doi.org/10.5281/zenodo.4750944.

[Fra21c] Maximilian Frank. *Cyber Range Kyoushi Thesis Scenario 2 - Processing Config and Labeling Rules*. May 2021. DOI: 10.5281/zenodo.4750959. URL: https://doi.org/10.5281/zenodo.4750959.

[Gao+21] Peng Gao et al. *Enabling Efficient Cyber Threat Hunting With Cyber Threat Intelligence*. 2021. arXiv: 2010.13637 [cs.CR].

[Gha+16] Amirhossein Gharib et al. „An Evaluation Framework for Intrusion Detection Dataset“. In: *2016 International Conference on Information Science and Security (ICISS)*. 2016, pp. 1–6. DOI: 10.1109/ICISSEC.2016.7885840.

[Gri+19] Martin Grimmer et al. „A modern and sophisticated host based intrusion detection data set“. In: *IT-Sicherheit als Voraussetzung für eine erfolgreiche Digitalisierung* (2019), pp. 135–145.

[HJB12] Jeff Hodges, Collin Jackson, and Adam Barth. „Http strict transport security (hsts)“. In: *URL: http://tools. ietf. org/html/draft-ietf-websec-strict-transport-sec-04* (2012).

[HR20] Guy Harris and Michael Richardson. *PCAP Capture File Format*. Internet-Draft draft-gharris-opsawg-pcap-01. https://www.ietf.org/archive/id/draft-gharris-opsawg-pcap-01.txt. IETF Secretariat, Dec. 2020. URL: https://www.ietf.org/archive/id/draft-gharris-opsawg-pcap-01.txt.

[Hst] URL: https://hstspreload.org/.

[Kie+14] Peter M. Kielar et al. „Concurrent Hierarchical Finite State Machines for Modeling Pedestrian Behavioral Tendencies“. In: *Transportation Research Procedia* 2 (2014). The Conference on Pedestrian and Evacuation Dynamics 2014 (PED 2014), 22-24 October 2014, Delft, The Netherlands, pp. 576–584. ISSN: 2352-1465. DOI: https://doi.org/10.1016/j.trpro.2014.09.098. URL: https://www.sciencedirect.com/science/article/pii/S2352146514001343.

[KK04] Hyang-Ah Kim and Brad Karp. „Autograph: Toward Automated, Distributed Worm Signature Detection.“ In: *USENIX security symposium*. Vol. 286. San Diego, CA. 2004.

[KL05] Oleg Kolesnikov and Wenke Lee. *Advanced polymorphic worms: Evading ids by blending in with normal traffic*. Tech. rep. Georgia Institute of Technology, 2005.

[KOS13] David R. Karger, Sewoong Oh, and Devavrat Shah. „Efficient Crowdsourcing for Multi-Class Labeling“. In: *SIGMETRICS Perform. Eval. Rev.* 41.1 (June 2013), 81–92. ISSN: 0163-5999. DOI: 10.1145/2494232.2465761. URL: https://doi.org/10.1145/2494232.2465761.

[Lan18] Max Landauer. *Dynamic log file analysis: an unsupervised cluster evolution approach for anomaly detection*. eng. Wien, 2018. URL: https://resolver.obvsg.at/urn:nbn:at:at-ubtuw:1-108603.

[Lan+18] Max Landauer et al. „Dynamic log file analysis: An unsupervised cluster evolution approach for anomaly detection“. In: *computers & security* 79 (2018), pp. 94–116.

[Lan+20a] Max Landauer et al. *AIT Log Data Set V1.0*. Version v1_0. Additionally funded by the FFG projects INDICAETING (868306) and DECEPT (873980). Mar. 2020. DOI: 10.5281/zenodo.3723083. URL: https://doi.org/10.5281/zenodo.3723083.

[Lan+20b] Max Landauer et al. „Have It Your Way: Generating Customized Log Data Sets with a Model-driven Simulation Testbed“. In: *IEEE Transactions on Reliability* (2020). Additionally funded by the FFG projects INDICAETING (868306) and DECEPT (873980).

[Lei+20] Maria Leitner et al. „AIT Cyber Range: Flexible Cyber Security Environment for Exercises, Training and Research“. In: *European Interdisciplinary Cybersecurity Conference (EICC)*. 2020, pp. 18–19.

[Lia+13] Hung-Jen Liao et al. „Intrusion detection system: A comprehensive review“. In: *Journal of Network and Computer Applications* 36.1 (2013), pp. 16–24. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2012.09.004. URL: https://www.sciencedirect.com/science/article/pii/S1084804512001944.

[Lia+16] Gaoqi Liang et al. „The 2015 ukraine blackout: Implications for false data injection attacks“. In: *IEEE Transactions on Power Systems* 32.4 (2016), pp. 3317–3318.

[MA07] Iyatiti Mokube and Michele Adams. „Honeypots: Concepts, Approaches, and Challenges“. In: *Proceedings of the 45th Annual Southeast Regional Conference*. ACM-SE 45. Winston-Salem, North Carolina: Association for Computing Machinery, 2007, 321–326. ISBN: 9781595936295. DOI: 10.1145/1233341.1233399. URL: https://doi.org/10.1145/1233341.1233399.

[Mir+14] Seyed M Mirtaheri et al. „A brief history of web crawlers“. In: *arXiv preprint arXiv:1405.0749* (2014).

[MIT17] MITRE. *CALDERA*. https://github.com/mitre/caldera, commit 981431200e64539ec858207a56cacdad403853db. 2017.

[Moz] *Top 500 Most Popular Websites*. URL: https://moz.com/top500.

[Nai+19] Nitin Naik et al. „Cyberthreat hunting-part 2: Tracking ransomware threat actors using fuzzy hashing and fuzzy c-means clustering“. In: *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE. 2019, pp. 1–6.

[NAS19] Asaf Nadler, Avi Aminov, and Asaf Shabtai. „Detection of malicious and low throughput data exfiltration over the DNS protocol“. In: *Computers & Security* 80 (Jan. 2019), pp. 36–53. DOI: 10.1016/j.cose.2018.09.006. URL: https://doi.org/10.1016/j.cose.2018.09.006.

[NK19] Petteri Nevavuori and Tero Kokkonen. „Requirements for Training and Evaluation Dataset of Network and Host Intrusion Detection System“. In: *New Knowledge in Information Systems and Technologies*. Ed. by Álvaro Rocha et al. Cham: Springer International Publishing, 2019, pp. 534–546. ISBN: 978-3-030-16184-2.

[Oge+15] A Ogee et al. „The 2015 Report on National and International Cyber Security Exercises“. In: *Technical Report*. European Network and Information Security Agency, 2015.

[Owa] *Unrestricted File Upload | OWASP*. en. URL: https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload (visited on 03/30/2021).

[Ray+20] Joshua Ray et al. *Cyber Threatscape Report*. Tech. rep. Accenture, 2020. URL: https://www.accenture.com/_acnmedia/PDF-137/Accenture-2020-Cyber-Threatscape-Report.pdf.

[Ret+14] Daniela Retelny et al. „Expert Crowdsourcing with Flash Teams“. In: *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*. UIST '14. Honolulu, Hawaii, USA: Association for Computing Machinery, 2014, 75–85. ISBN: 9781450330695. DOI: 10.1145/2642918.2647409. URL: https://doi.org/10.1145/2642918.2647409.

[Rin+17] Markus Ring et al. „Flow-based benchmark data sets for intrusion detection“. In: *Proceedings of the 16th European conference on cyber warfare and security*. 2017, pp. 361–369.

[Rin+19] Markus Ring et al. „A survey of network-based intrusion detection data sets“. In: *Computers and Security* 86 (2019), pp. 147 –167. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2019.06.005. URL: http://www.sciencedirect.com/science/article/pii/S016740481930118X.

[Ros+20] Scott Rose et al. *Zero Trust Architecture*. Tech. rep. Aug. 2020. DOI: 10.6028/nist.sp.800-207. URL: https://doi.org/10.6028/nist.sp.800-207.

[SG15] Shruti Sharma and Parul Gupta. „The anatomy of web crawlers“. In: *International Conference on Computing, Communication & Automation*. IEEE. 2015, pp. 849–853.

[Sha+18] Iman Sharafaldin et al. „Towards a reliable intrusion detection benchmark dataset“. In: *Software Networking* 2018.1 (2018), pp. 177–200.

[SM07] Karen Scarfone and Peter Mell. „Guide to intrusion detection and prevention systems (idps)“. In: *NIST special publication* 800.2007 (2007), p. 94.

[SM12] Karen Scarfone and Peter Mell. *Guide to intrusion detection and prevention systems (idps)*. Tech. rep. National Institute of Standards and Technology, 2012.

[Son+11] Jungsuk Song et al. „Statistical Analysis of Honeypot Data and Building of Kyoto 2006+ Dataset for NIDS Evaluation“. In: *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*. BADGERS '11. Salzburg, Austria: Association for Computing Machinery, 2011, 29–36. ISBN: 9781450307680. DOI: 10.1145/1978672.1978676. URL: https://doi.org/10.1145/1978672.1978676.

[Ull+18] Faheem Ullah et al. „Data exfiltration: A review of external attack vectors and countermeasures“. In: *Journal of Network and Computer Applications* 101 (Jan. 2018), pp. 18–54. DOI: 10.1016/j.jnca.2017.10.016. URL: https://doi.org/10.1016/j.jnca.2017.10.016.

[VAB18] Saurabh Vaidya, Prashant Ambad, and Santosh Bhosle. „Industry 4.0 – A Glimpse“. In: *Procedia Manufacturing* 20 (2018). 2nd International Conference on Materials, Manufacturing and Design Engineering (iCMMD2017), 11-12 December 2017, MIT Aurangabad, Maharashtra, INDIA, pp. 233–238. ISSN: 2351-9789. DOI: https://doi.org/10.1016/j.promfg.2018.02.034. URL: https://www.sciencedirect.com/science/article/pii/S2351978918300672.

[Vin+19] R. Vinayakumar et al. „Deep Learning Approach for Intelligent Intrusion Detection System“. In: *IEEE Access* 7 (2019), pp. 41525–41550. DOI: 10.1109/ACCESS.2019.2895334.

[WL13] Wenye Wang and Zhuo Lu. „Cyber security in the Smart Grid: Survey and challenges“. In: *Computer Networks* 57.5 (2013), pp. 1344–1371. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2012.12.017. URL: https://www.sciencedirect.com/science/article/pii/S1389128613000042.

[Wur+18] Markus Wurzenberger et al. „AECID: A Self-learning Anomaly Detection Approach based on Light-weight Log Parser Models.“ In: *ICISSP*. 2018, pp. 386–397.

[YKG20] Muhammad Mudassar Yamin, Basel Katt, and Vasileios Gkioulos. „Cyber ranges and security testbeds: Scenarios, functions, tools and architecture“. In: *Computers & Security* 88 (2020), p. 101636. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2019.101636. URL: https://www.sciencedirect.com/science/article/pii/S0167404819301804.

APPENDIX A

Extra Figures

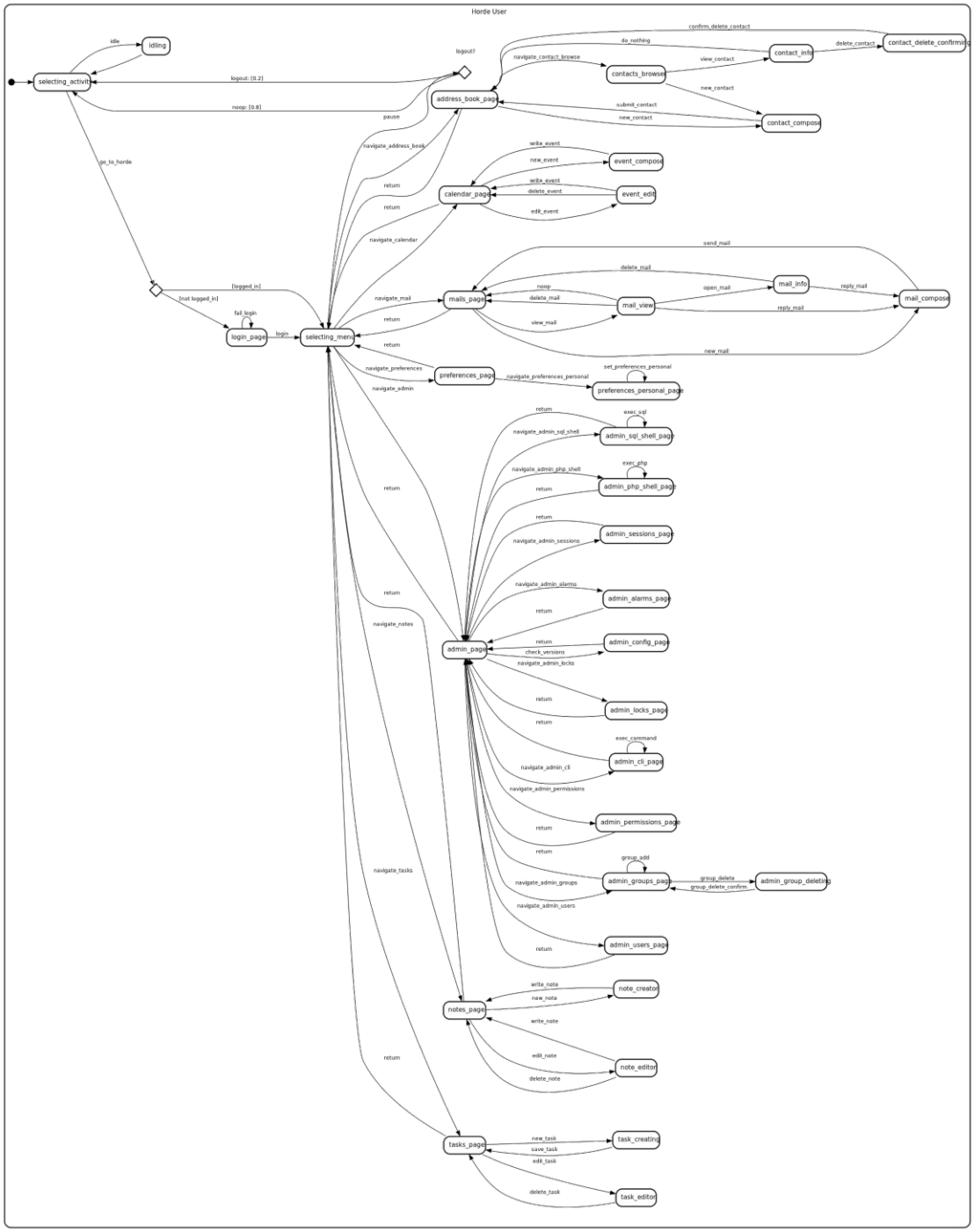

Figure A.1: Horde Webmail Activity