KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Computer Science

Second cycle, 30 credits

Automated Profiling of Cyber Attacks Based on MITRE ATT&CK

BENGTH PAPPILA

Stockholm, Sweden, 2024

Automated Profiling of Cyber Attacks Based on MITRE ATT&CK

BENGTH PAPPILA

Master’s Programme, Computer Science, 120 credits
Date: September 6, 2024

Supervisor: Kim Hammar
Examiner: Prof. Rolf Stadler
School of Electrical Engineering and Computer Science
Swedish title: Automatiserad Profilering av Cyberattacker Baserat på MITRE ATT&CK

Abstract

This Master thesis presents a framework for automated profiling of cyber attacks based on MITRE ATT&CK®. The framework includes two components: (1) a component for automated mapping of sequences of attacker actions to the corresponding tactics and techniques in MITRE ATT&CK®; and (2) a component for probabilistic profiling of attacker actions based on testbed measurements. The latter component models the relation between attacker actions and testbed measurements using a hidden Markov model, which allows to estimate the most likely attack sequence using probabilistic inference. The experimental part of this thesis includes extensive profiling of emulated attacks in the Cyber Security Learning Environment (CSLE), which is a platform for emulating attacks and defenses in virtualized IT environments. Our experimental results show that our framework is able to automatically map attacker actions in CSLE to MITRE ATT&CK® and that it can accurately estimate the start time of an attack based on testbed measurements.

Keywords

Attack emulation, Attack profiling, Autonomous network security, Cyber security, Hidden Markov Model, Mitre Att&ck, The Cyber Security Learning Environment (CSLE)

Sammanfattning

Denna masteruppsats presenterar ett ramverk för automatisk profilering av cyberattacker baserat på MITRE ATT&CK®. Ramverket inkluderar två komponenter: (1) en komponent för automatisk kartläggning av attacksekvenser till motsvarande taktiker i MITRE ATT&CK®; och (2) en komponent för probabilistisk profilering av attackaktioner baserat på mätdata från en testbädd. Den senare komponenten modellerar relationen mellan attackaktioner och mätdata från testbädden genom dolda Markovmodeller, vilket möjliggör estimering av den mest sannolika attacksekvensen med hjälp av probabilistisk inferens. Den experimentella delen av den här uppsatsen inkluderar omfattande profilering av emulerade cyberattacker i “the Cyber Security Learning Environment (CSLE)", vilket är en plattform för att emulera cyberangrepp och försvar i en virtuell IT-miljö. Resultaten visar att vårt ramverk automatiskt kan kartlägga attackaktioner i CSLE baserat på MITRE ATT&CK® och att den kan estimera starttiden för en attack med hög träffsäkerhet baserat på mätdata från testbädden.

Nyckelord

Attack emulering, Attack profilering, Autonom nätverkssäkerhet, Cybersäkerhet, Dold Markovmodell, Mitre Att&ck, The Cyber Security Learning Environment (CSLE)

Acknowledgments

I would like to express gratitude to my supervisor, Kim Hammar, for his mentorship and guidance throughout this thesis. His expertise and constructive feedback have been fundamental in shaping the content and direction of my work.

Stockholm, September 2024
Bength Pappila

Contents

List of Figures

List of Tables

List of acronyms and abbreviations

CAPEC	Common Attack Pattern Enumeration and Classification
CSLE	The Cyber Security Learning Environment
HMM	Hidden Markov Model
IDS	Intrusion Detection System
KLD	Kullback-Leibler divergence
TTP	Tactics, techniques and procedure

Chapter 1

Introduction

The ubiquity of cyber attacks has never been more apparent, and their far-reaching impacts on society demonstrate the critical need for automated security solutions. For example, Tietoevry data centers were recently attacked by Akira ransomware, which affected Swedish government agencies. Many institutions rely on manual configurations and domain experts to respond to incidents. While this approach can provide basic security for an organization's IT infrastructure, large IT infrastructures possess many attack vectors that are difficult and expensive for domain experts to analyze. Consequently, the demand for *automated* security solutions is increasing.

This thesis addresses the need described above and presents a novel framework for automated *profiling* of cyber attacks. With *attack profiling*, we mean the process of identifying and categorizing the characteristics and patterns of cyber-attacks.

Our framework for attack profiling involves two components. The first component is dedicated to classifying attack actions (i.e., network commands executed by an attacker) using MITRE ATT&CK®*, which is a comprehensive knowledge base that describes attacker behavior. The second component focuses on probabilistic inference of attack actions using a hidden Markov model. Both components are evaluated experimentally based on attacks emulated using The Cyber Security Learning Environment (CSLE) [1].

1.1 Background

The work presented in this thesis is part of a larger research project for automated security, whereby the problem of finding effective security policies

*https://attack.mitre.org

for an IT infrastructure is formulated as an optimization problem [2][3][4]. A key part of this research is the development of CSLE, a platform that emulates large-scale IT infrastructures and cyber attacks. (With emulation, we mean the creation of a software or hardware environment that behaves like the original system.) Through such emulation, we can collect data and compute effective security policies. The attack profiling methods presented in this thesis are integrated into CSLE to enable automated profiling of cyber attacks.

1.2 Problem

In this thesis, we study how to profile attacks in CSLE based on the MITRE ATT&CK knowledge base. Here, with "attack," we mean a sequence of network commands (actions) executed by an attacker in CSLE.

The following questions are examined:

- How can we model different types of attacks?
- How can we automatically profile a given sequence of network commands (attacker actions) using the model?
- How can we automatically profile an attack when the sequence of the attacker's network commands is unknown, and the only available information is a sequence of system measurements (e.g., log files and alerts)?

1.3 Approach

We model an attack in CSLE as a sequence of *attack actions*, i.e., a sequence of network commands executed by an attacker. To profile an attack, we map each command to data from MITRE ATT&CK (e.g., attack tactics and techniques). To automate this profiling, we leverage open-source APIS to map network commands in CSLE to data from MITRE ATT&CK. A challenge with this approach is that a network command in CSLE often maps to a lot of irrelevant data from MITRE ATT&CK. To address this issue, we design an algorithm that takes as input an *attack graph* that encodes the structure of an attack based on domain knowledge and then uses that structure to prune the data from MITRE ATT&CK. We show that this pruning leads to more meaningful attacker profiling. Finally, to profile attacks when the attacker's network commands are *unavailable*, we use a hidden Markov model to estimate the most likely sequence of network commands based on system measurements (e.g., log files and alerts).

1.4 Delimitations

This thesis focuses on specific types of attacks in CSLE. As a consequence, the thesis is delimited to the subset of attack techniques and tactics from MITRE ATT&CK® that are appropriate for the attacks that are studied. Furthermore, the project is using the specific framework 'MITRE ATT&CK® for Enterprise' and does not consider other parts of MITRE ATT&CK, e.g., MITRE ATT&CK for industrial control systems.

1.5 Structure of the thesis

The remainder of this thesis is structured as follows. Chapter 2 presents relevant background information and related works. Chapter 3 presents the methodology. Chapter 4 presents our model and solution framework. Chapter 5 presents the results of the implemented framework, followed by a discussion of the results. Lastly, Chapter 6 presents this thesis's conclusion and suggestions for future work.

Chapter 2

Background

This chapter provides background information about MITRE ATT&CK®, Kullback-Leibler divergence, and Hidden Markov Models. The chapter also discusses related work.

2.1 MITRE ATT&CK® for Enterprise

MITRE ATT&CK® is a knowledge base and model of attacker behavior. It consists of three matrices, each functioning as a framework to organize and present various aspects of attack behaviors. The three matrices are organized based on domains where attacks might occur: Enterprise, Mobile, and Industrial Control System (ICS). Each matrix has three core components: tactics, techniques, and sub-techniques.

2.1.1 Tactics

Tactics represent an attacker's reason for performing an action. A tactic is a contextual category for individual techniques. It contains information on what an attacker does during a specific phase of the attack. The Enterprise Matrix contains 14 tactics; we describe some of them below.

- `Reconnaissance`. The objective of this tactic is to gather information to be used to plan future operations. It contains techniques to actively or passively gather information about the victim organization, infrastructure, or staff. For example, *Active Scanning* is a technique within the "Reconnaissance" tactic that models an attacker that examines the victims' infrastructure via network traffic.

- `Resource Development`. The objective of this tactic is for the attacker to establish resources to support further operations. For example, the *Compromise Accounts* technique within the "Resource Development" tactic represents an adversary that compromises existing accounts – such as email accounts – to support phishing attacks as a part of gaining initial access.

- `Initial Access`. This tactic aims to get an initial foothold within the network. The initial foothold could be gained by using *Content Injection*, a technique within the "Initial Access" tactic by injecting malicious content through online network traffic where the attacker can manipulate the traffic and inject their content.

- `Execution`. The objective of the attacker in this tactic is to run malicious code after gaining initial access to the target system. For example, *Command and Scripting Interpreter* is a technique within the "Execution" tactic where the attacker abuses command interpreters to execute commands, scripts, or binaries.

2.1.2 Techniques and Sub-techniques

Techniques and sub-techniques in MITRE ATT&CK® represent how the attacker achieves the objectives of attack tactics. Sub-techniques are specific methods an attacker may use to implement a particular technique. One example technique is *Command and Scripting Interpreter*, which contains sub-techniques such as *Python* and *Powershell*.

A technique includes several attributes, e.g.,

- `Tactics`. The tactics under which the technique is categorized.

- `Platform`. The platforms on which the technique is used.

- `Sub-techniques`. The sub-techniques that belong to a technique.

- `Mitigations`. Configurations, tools, or processes that prevent the (sub-)technique from working.

- `Data Source`. Source of information collected by a sensor or logging system. It can be utilized to identify the attacker action being performed.

In total, MITRE ATT&CK® for Enterprise contains 201 techniques and 424 sub-techniques.

2.2 Kullback-Leibler divergence

In this thesis, we use Kullback-Leibler divergence (KLD) for feature selection. In particular, we use it to quantify the difference between probability distributions of infrastructure metrics during an attack and normal operation, which in turn allows to identify the features (metrics) that provides the most information for detecting an attack. Let $P = \{p_1, p_2, ..., p_n\}$ and $Q = \{q_1, q_2, ..., q_n\}$ represent two discrete distributions. Then, the Kullback-Leibler divergence is defined as follows:

$$D_{KL} = \sum_i p_i \cdot \log_2 \left(\frac{p_i}{q_i}\right).$$

The KLD is not symmetric i.e., $D_{KL}(P||Q) \neq D_{KL}(Q||P)$. In case the i^{th} element is missing in either distribution, the p_i or q_i is evaluated as 0, which makes the value of the equation undefined. The constant back-off smoothing algorithm can be applied to overcome this issue [5].

2.3 Hidden Markov Model (HMM)

A Hidden Markov Model (HMM) is a statistical model based on hidden states and observations describing a Markov process. The model captures the relationship between the observations and the hidden states. The model λ is described as follows:

$\lambda = (\mathcal{A}, \mathcal{B}, \pi)$	HMM model	(2.1a)
$x \in \{1, \dots, N\}$	States	(2.1b)
$k \in \{1, \dots, K\}$	Observations	(2.1c)
$\mathcal{A} = \{\mathcal{A}_{ij} \mid 1 \leq i, j \leq N\}$	State-transition matrix	(2.1d)
$\mathcal{A}_{ij} = P(x_t = j \mid x_{t-1} = i)$	Transition probability	(2.1e)
$\mathcal{B} = \{\mathcal{B}_{ik} \mid 1 \leq i \leq N, 1 \leq k \leq K\}$	Observation matrix	(2.1f)
$\mathcal{B}_{ik} = P(o_t = k \mid x_t = i)$	Observation probability	(2.1g)
$\pi = \{\pi_i \mid 1 \leq i \leq N\}$	Initial state distribution	(2.1h)
$\pi_i = P(x_1 = i)$	Initial state probability	(2.1i)

Here, $\mathcal{A}$ and $\mathcal{B}$ are row stochastic matrices, meaning that all rows sum up to 1. Similarly, π is a probability distribution, i.e., the entries of π sum up to 1. It follows from the first-order Markov assumption that:

$$P(X_t = j | X_{t-1} = i) = \mathcal{A}_{ij}.$$

This assumption implies that the probability of transitioning to state j at time t depends only on the state at time $t-1$. Similarly, the observation probability $P(O_t = j | X_t = i)$ is determined by the current state i.

Training a model λ involves estimating the parameters in (2.1). In this thesis, we estimate these parameters with empirical probability distributions computed based on measurements from our testbed (see §4.3.1).

2.4 Related work

In the following subsections, we describe prior work on emulating cyber attacks and automated attack profiling.

2.4.1 Attack Emulation

There is a lot of research on attack emulation in the cybersecurity domain. Applebaum *et al.* [6] propose a framework for automated red team emulation. They focus on a red team's activities after gaining access to a system. In particular, they developed CALDERA, an attacker emulation tool that uses an automated planner to predict future actions of the attacker based on MITRE ATT&CK®. In a follow-up work, the same authors develop a simulation testbed and compare different attack strategies. They find that using an automated planner leads to better attack modeling performance than not using an automated planner.

In a separate line of work, NASimEmu is a research project by Janisch *et al.* [7] that aims to develop a framework that trains an attacker in simulation using the Network Attack Simulator (NASim)[8] and an associated emulator. Similar to NASimEmu, Standen *et al.* [9] introduce CyBORG, a platform for simulating cyber attacks, which is specifically designed to enable the training of autonomous defense agents. Other environments with similar characteristics as NASimEmu and CyBORG include: PenGym [10], ASAP Chowdhary *et al.* [11], CLAP Yang and Liu [12], and Cygil Li *et al.* [13].

2.4.2 Attack Profiling

Automated attack profiling is an active area of research that studies how to leverage measurement data to categorize cyber attacks automatically. Like this thesis, Rodríguez *et al.* [14] analyze runtime events from systems to profile malicious behavior according to the tactics in MITRE ATT&CK®. Their work shows promising results of using raw data and process mining tools to identify the characteristics of an attacker. In a similar line of work, Wu *et al.* [15] present GroupTracer, a framework aimed at extracting Tactics, techniques and procedures (TTPs) profiles from log data collected on IoT devices. Lastly, the work by Wang and Stadler [16] and Holgado *et al.* [17] use statistical learning methods, e.g., HMMs to predict attacks. [16] uses the same testbed (CSLE) used in this thesis for their research. Other frameworks for automated attack profiling include MAMBA[18], Holmes [19], and RapSheet [20]. These frameworks use execution traces and log files to automatically classify cyber attacks and map them to MITRE ATT&CK®. Lastly, Miehling *et al.* [21] introduce a formal model for real time network protection. Their work demonstrates how Bayesian attack graphs can model attacker behavior and be used for defense strategies in real time.

Among the references listed above, the most similar to this thesis are the works described in Rodríguez *et al.* [14], Wang and Stadler [16], and Applebaum *et al.* [6]. This thesis differs from these works in the following ways. First, the difference between Rodríguez *et al.* [14] and this thesis is that we use collected metrics to train a model and then probabilistically identify malicious activity, whereas Rodríguez *et al.* [14] assumes that malicious events are already labeled. Second, Wang and Stadler [16] pre-process the observation space and do not consider the MITRE ATT&CK knowledge base. By contrast, we do not perform such preprocessing, and our method is centered around MITRE ATT&CK. Lastly, CALDERA by [6] maps a single technique to an attacker action — by contrast, our approach allows us to map multiple techniques to an attacker action.

Chapter 3

Methodology

The research method consists of the following steps:

Step 1 Create a model of an attacker action based on MITRE ATT&CK® for Enterprise.

Step 2 Define and implement an attack profiler that maps network commands in CSLE to the model.

Step 3 Define and implement an attack profiler that maps *sequences* of network commands in CSLE to the model.

Step 4 Define and implement an attack profiler that estimates the most likely sequences of network commands in CSLE based on system measurements.

Step 5 Evaluate the implemented attack profilers based on cyber attacks emulated in CSLE.

Step 1 involves theoretical modeling, which is needed for understanding the structure of the components in MITRE ATT&CK® that are relevant to the CSLE platform. In Step 2, the model developed in Step 1 is utilized to map network commands in CSLE to MITRE ATT&CK®, this provides a systematic way to profile actions. In Step 3, the framework is extended by incorporating temporal aspects. It explores how a sequence of actions can be mapped to the model. Step 4 investigates how probabilistic methods can be used to assess the likelihood of specific attack actions being executed based on data from the system. Lastly, in Step 5, we evaluate the implemented attack profilers based on data from the CSLE testbed, described below.

3.1 Testbed

The testbed consists of machines that run the CSLE emulation system. The emulation system runs a virtualization layer provided by Docker containers and virtual links. The system implements network isolation and traffic shaping using network namespaces and the netem module in the Linux kernel. Resource allocation to containers, e.g., CPU and memory, are enforced using cgroups.

The network topology of the emulated infrastructure is shown in Fig. 3.1. The emulation system includes the clients, the attacker, the defender, network connectivity, and 31 devices of the target infrastructure (e.g., application servers and the gateway). The software functions on the emulation system replicate important components of the target infrastructure, such as web servers, databases, and the Snort IDS, which is deployed using Snort's community ruleset v2.9.17.1.

Connections between servers are emulated as full-duplex lossless connections of 1 Gbit/s capacity in both directions. Connections between the gateway and the external client population are emulated as full-duplex connections of 100 Mbit/s capacity and 0.1% packet loss with random bursts of 1% packet loss. (These numbers are based on measurements on enterprise and wide-area networks.)

3.2 Data collection

The data used for the experimental part of this thesis is collected from the CSLE testbed. We collect $25,000$ measurements of the number of intrusion detection alerts generated by Snort both during normal operation and during intrusions.

3.3 Goal of experiments

The goal of the experiments is to evaluate the implemented attack profilers. The implementations are evaluated and assessed based on generality and efficiency. The generality is evaluated based on the ability to handle different types of attacks on the CSLE testbed. The efficiency of the implementations is measured based on their computational performance and scalability.

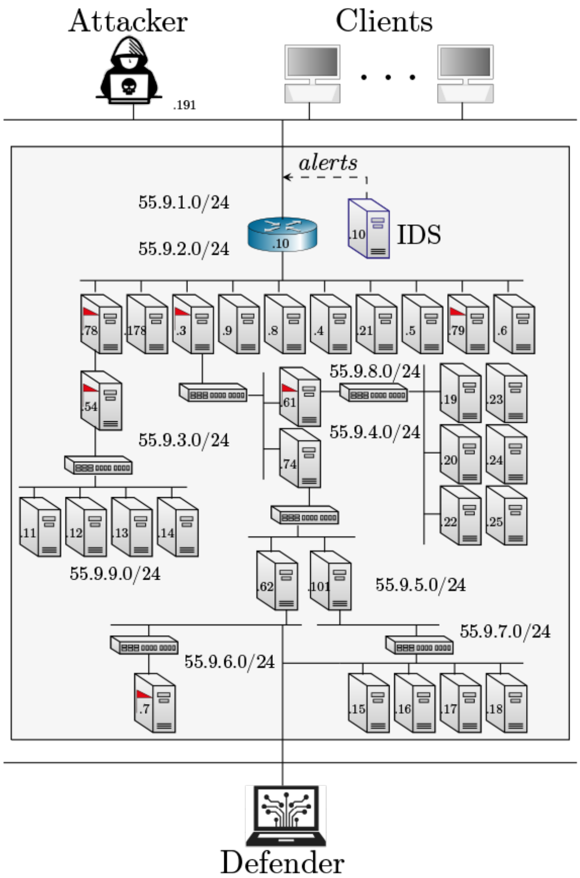

Figure 3.1: The CSLE testbed used for data collection.

Chapter 4

Attack profiler

In this chapter, we present our framework for attacker profiling, which comprises two main components. The first component maps network commands in CSLE to data from ATT&CK ENTERPRISE, while the second component estimates network commands in CSLE based on measurement data. By an attacker action we mean an network command issued by an emulated attacker in the CSLE platform, e.g a TCP SYN SCAN. By "attack profiler", we mean a tool designed to estimate sequences of network commands in CSLE from measurement data and map them to data from ATT&CK ENTERPRISE. We use $\mathcal{C}$ to denote the set of network commands implemented in CSLE.

The ATT&CK ENTERPRISE framework provides several sets of data: attack tactics $\mathcal{T}$, attack techniques T, sub-techniques $\mathcal{S}$, attack mitigations $\mathcal{M}$, and data sources $\mathcal{D}$. These sets are related through specific correspondences:

$$f_{\mathcal{T},\mathbf{t}} : \mathsf{T} \rightarrow 2^{\mathcal{T}} \qquad \text{technique to tactics} \tag{4.1}$$

$$f_{\mathcal{T},\mathbf{s}} : \mathsf{T} \rightarrow 2^{\mathcal{S}} \qquad \text{technique to sub-techniques} \tag{4.2}$$

$$f_{\mathcal{T},\mathbf{m}} : \mathsf{T} \rightarrow 2^{\mathcal{M}} \qquad \text{technique to mitigations} \tag{4.3}$$

$$f_{\mathcal{T},\mathbf{d}} : \mathsf{T} \rightarrow 2^{\mathcal{D}} \qquad \text{technique to data sources.} \tag{4.4}$$

These correspondences are implemented in open-source APIS that can be invoked from our attack profiler.

Based on the data available in ATT&CK ENTERPRISE, we define a profiled network command (attacker action) to consist of the associated tactics, techniques, sub-techniques, mitigations, and data sources in ATT&CK ENTERPRISE, as defined below.

Definition 1 (Profiled attacker action)**.** *A profiled attacker action a is a tuple*

$$a \triangleq \langle \mathbf{t}, \boldsymbol{\tau}, \mathbf{s}, \mathbf{m}, \mathbf{d} \rangle, \tag{4.5}$$

where

$$\begin{aligned}
&\boldsymbol{\tau} \subseteq \mathsf{T} & & \textit{techniques} \quad (4.6)\\
&\mathbf{t} \subseteq \mathcal{T} & \forall t \in \mathbf{t}\, \exists \tau \in \boldsymbol{\tau} : t \in f_{\tau,\mathbf{t}}(\tau) & \quad \textit{tactics} \quad (4.7)\\
&\mathbf{s} \subseteq \mathcal{S} & \forall s \in \mathbf{s}\, \exists \tau \in \boldsymbol{\tau} : s \in f_{\tau,\mathbf{s}}(\tau) & \quad \textit{sub-techniques} \quad (4.8)\\
&\mathbf{m} \subseteq \mathcal{M} & \forall m \in \mathbf{m}\, \exists \tau \in \boldsymbol{\tau} : m \in f_{\tau,\mathbf{s}}(\tau) & \quad \textit{mitigations} \quad (4.9)\\
&\mathbf{d} \subseteq \mathcal{D} & \forall d \in \mathbf{d}\, \exists \tau \in \boldsymbol{\tau} : d \in f_{\tau,\mathbf{s}}(\tau) & \quad \textit{data sources.} \quad (4.10)
\end{aligned}$$

Equations (4.6)–(4.10) are constraints that ensure that the components of the tuple in (4.5) are consistent, e.g., that each tactic is consistent with at least one technique, etc. More specifically, Eq. (4.6) states that the set of techniques is a subset of the set of techniques provided by ATT&CK ENTERPRISE; (4.7)–(4.10) state two things: a) that the sets of tactics, sub-techniques, mitigations, and data sources belong to ATT&CK ENTERPRISE, and b) that each tactic, sub-technique, mitigation, and data source is related to a technique. We provide an example of a profiled attacker action below.

Example 1. Consider a network command associated with the technique ACTIVE SCANNING. By using the correspondences in (4.1)—(4.4), we can automatically associate the network command with the tactic RECONNAISSANCE, the sub-technique VULNERABILITY SCANNING, the mitigation PRE–COMPROMISE, and the data source NETWORK TRAFFIC.

4.1 Profiling a single attack

Given the definition of a profiled attacker action, the task of the attack profiler is to map a network command c to a tuple a that satisfies Def. 1. More specifically, the attack profiling problem can be defined as follows.

Problem 1 (Attack profiling)**.** *Implement the mapping*

$$\varphi : \mathcal{C} \to \mathcal{A}, \tag{4.11}$$

where $\mathcal{C}$ is the set of network commands that should be profiled and $\mathcal{A}$ is the set of actions that satisfy Def. 1.

Network command	Techniques
TCP SYN SCAN	ACTIVE SCANNING GATHER VICTIM HOST INFORMATION NETWORK SERVICE DISCOVERY
SSH BACKDOOR	COMPROMISE CLIENT SOFTWARE BINARY CREATE ACCOUNT
SAMBACRY EXPLOIT	EXPLOIT PUBLIC FACING APPLICATION REMOTE SERVICES EXPLOITATION OF REMOTE SERVICE NATIVE API
CVE 2015-1427 EXPLOIT	EXPLOIT PUBLIC FACING APPLICATION EXPLOITATION OF REMOTE SERVICE COMMAND AND SCRIPTING INTERPRETER FALLBACK CHANNELS

Table 4.1: Network commands and corresponding techniques

We implement the mapping of Prob. 1 by manually labeling network commands with attack techniques in ATT&CK ENTERPRISE. Examples of network commands and the associated techniques are listed in Tab. 4.1. Based on this manual labeling, we can then automatically associate network commands with tactics, sub-techniques, mitigations, and data sources by invoking the correspondences in (4.1)—(4.4). The following example illustrates these steps. (Further details of our implementation can be found in Sec. 4.4.)

Example 2. Consider the network command TCP SYN SCAN in CSLE. Based on domain knowledge, we have manually mapped this command to three techniques: ACTIVE SCANNING, GATHERING VICTIM HOST INFORMATION, and NETWORK SERVICE DISCOVERY. Using the correspondences (4.1)—(4.4) provided by ATT&CK ENTERPRISE, we can infer that the network command is associated with the DISCOVERY and RECONNAISSANCE tactics, as well as *three* mitigations and *four* data sources.

4.2 Profiling attacker sequences

Problem 1 captures the task of profiling an *individual* attack command based on ATT&CK ENTERPRISE. However, it does not capture the task of profiling *sequences* of attack commands. We formulate this task as follows.

Problem 2 (Attack profiling a sequence of commands)**.** *Given a set of network commands $\mathcal{C}$ and a maximum sequence length N, implement the mapping*

$$\vartheta : \mathcal{C}^N \rightarrow \mathcal{A}^N, \tag{4.12}$$

where $\mathcal{C}$ is the set of network commands that should be profiled and $\mathcal{A}$ is the set of actions that satisfy Def. 1.

One naive implementation of the mapping in Prob. 2 is to apply a profiler that solves Prob 1 for each network command in the sequence. This implementation associates all possible techniques and tactics with each command, even though some techniques and tactics may be irrelevant due to the temporal structure of the sequence of commands. To address this problem, we propose using an attack graph incorporating domain knowledge about the attacker. This graph encodes the attack's temporal structure, allowing us to prune irrelevant tactics and techniques.

Example 3. Consider the sequence of network commands TCP SYN SCAN $\rightarrow$ SSH DICTIONARY ATTACK $\rightarrow$ NETWORK SERVICE LOGIN. These commands are associated with seven techniques and eight tactics in total. TCP SYN SCAN is mapped to the tactics 'RECONNAISSANCE' and 'DISCOVERY'. SSH DICTIONARY ATTACK is mapped to 'CREDENTIAL ACCESS', 'DEFENSE EVASION', 'PERSISTENCE', 'PRIVILEGE ESCALATION', and 'INITIAL ACCESS'. SERVICE LOGIN is mapped to 'INITIAL ACCESS' and 'LATERAL MOVEMENT'. The commands are also mapped to mitigations, data sources, and sub-techniques as defined in Def. 1.

We can deduce from the temporal structure of the command sequence in Ex. 3 that the first tactic must be 'RECONAISSANCE'. This follows because the technique 'DISCOVERY' is a tactic used by an attacker post-compromise, and hence, it cannot be the first tactic in the sequence. 'LATERAL MOVEMENT', is also a post-compromise tactic and can be deduced from the third command. As a consequence, the naïve attack profiler that repeatedly profiles each command without taking into account the temporal structure of the sequence is not precise as it associates both 'RECONNAISSANCE' and 'DISCOVERY' with the first command in the sequence, and 'INITIAL ACCESS' and 'LATERAL MOVEMENT' with the third command.

To make the profiling of sequences of network commands more precise, we construct and leverage an *attack graph* based on domain knowledge to represent the temporal structure of a typical attack. We define a node in the graph to be a tactic in MITRE ATT&CK® and we define edges to represent possible sequences of tactics that the attacker may follow. Since the attacker may reuse the same tactic several times throughout an attack sequence, a tactic may be associated with several nodes in the graph. Each node has a unique identifier to distinguish between nodes. Formally,

Definition 2 (Attack graph)**.** *Given the set of attack tactics $\mathcal{T}$ an attack graph $\mathcal{G}$ is a directed graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, where each node corresponds to a tactic in $\mathcal{T}$ and each edge represent a possible change of attacker tactic during an attack sequence.*

To solve Prob. 2 we introduce an algorithm utilizing the attack graph to prune tactics and techniques from a naïvely profiled sequence. The pseudocode of the algorithm is listed below.

Algorithm 1 Attack profiling sequence of commands

Input: attack graph $\mathcal{G}$ Def. 2, naive profiled attack sequence Ac
Output: pruned profiled attack sequence Ac

$Ac[1].tactics = Ac[1].tactics \cap \mathcal{G}.root$
$s = \mathcal{G}.root$
for i in 1, ..., len(Ac) **do**
  $s = \bigcup_{j \in s} \mathcal{G}.ch(j)$
  $Ac[i].tactics = Ac[i].tactics \cap s$
  if $|Ac[i].tactics| == 1$ **then**
    $s = Ac[i].tactics$
  end if
end for
return Ac

Algorithm 1 prunes the input sequence of attack actions (Def. 1) using an attack graph $\mathcal{G}$ (Def. 2). It starts by initializing the state to the root node of $\mathcal{G}$. It then iterates through the attack sequence. Each action in the sequence updates the state to be the set of child nodes (tactics) of the current state (set of tactics). After updating the state, the algorithm prunes the set of tactics of the current attack action by removing all tactics that are not included in the current state. If the set of tactics of the current attack action is a singleton set, the state is updated to that node. The same procedure continues until each action in the input sequence has been processed.

Since the number of child nodes of each node is upper bounded by $|\mathcal{T}|$ (Def. 4.7), it follows that the time complexity of Algorithm 1 is $\mathcal{O}(N|\mathcal{T}|)$, where N is the length of the attack sequence. The number of tactics that are pruned by Algorithm 1 depends on the attack graph $\mathcal{G}$ (Def. 2) as well as the input sequence of attack actions.

4.3 Probabilistic profiling of attacker sequences

Both of the attack profiling problems defined above (Probs. 1–2) require that the commands of the attacker are known. While the attacker's commands may be available after an attack has been discovered, the commands are generally not available during the attack, which means that profilers that solve Probs. 1–2 can not be used in real time. To address this limitation, we formulate a generalization of Prob. 2 where the attack commands are unknown and the only information available is system metrics that can be measured in real time (e.g., Intrusion Detection System (IDS) alerts and log files).

Problem 3 (Probabilistic profiling of an attack sequence)**.** *Given a sequence of infrastructure metric observations $o_1, o_2, \dots, o_N$, and a probability distribution $P(o \mid c)$, where $o \in \mathcal{O}$ is an infrastructure metric observation, and $c \in \mathcal{C}$ is a network command of the attacker. Implement the mapping*

$$\lambda : \mathcal{O}^N \to \mathcal{A}^N, \tag{4.13}$$

where $\mathcal{A}$ is the set of actions that satisfy Def. 1.

4.3.1 Empirical distributions of infrastructure metrics

Before presenting our solution to Prob. 3 we analyze the metrics $o_1, o_2, \dots, o_N$. 88 unique metrics are collected at periodic intervals from our testbed (see Section 3.1). We restrict our attention to metrics that have more than 10 unique values. We analyze the metric distributions under conditions of both intrusion (i.e., when the attacker executes a command) and non-intrusion.

We use the KLD to quantify the difference between the distributions. We compute the KLD values using a back-off smoothing algorithm, presented in Appendix A.6. To gain insight into the distributions of the metrics, we use quantiles. We can understand the spread of the different distributions by looking at the quantiles. For example, a metric could have high KLD values for distributions associated with certain network commands. We use the KLD for feature selection to find the most relevant metrics that contribute to distinguishing between an intrusion and non-intrusion. We use KLD because it is a measurement sensitive to differences between distributions, and we can effectively find a divergence between intrusion and non-intrusion events.

Figures 4.1–4.2 show the KLD of the distributions under the conditions of intrusion and non-intrusion for different metrics based on measurements from our testbed. Figures 4.3–4.5 show the empirical probability distributions of three selected metrics.

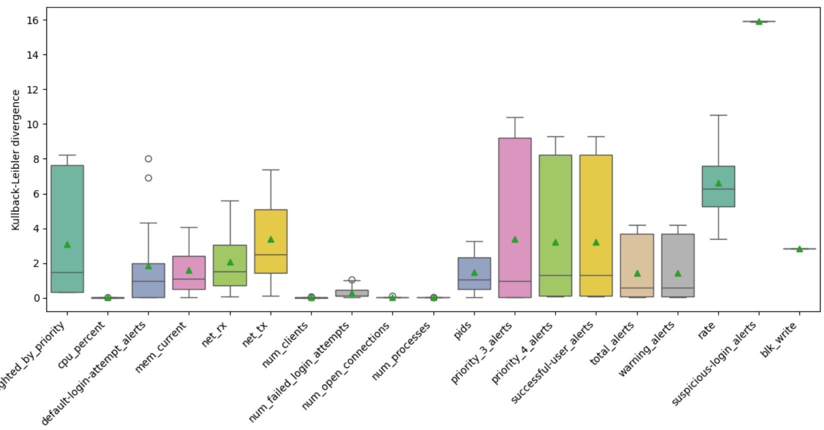

Figure 4.1: Boxplot of the Kullback-Leibler divergence values $D_{KL}(P||Q)$ for the metrics with more than 10 unique values. P is the probability distribution of a metric given the executed network command, and Q is the probability distribution of a metric given no executed network command.

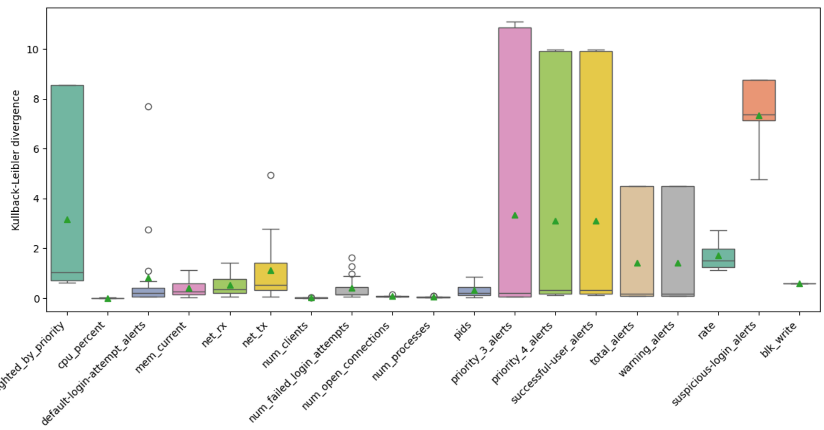

Figure 4.2: Boxplot of the Kullback-Leibler divergence values $D_{KL}(Q||P)$ for the metrics with more than 10 unique values. P is the probability distribution of a metric given the executed network command, and Q is the probability distribution of a metric given no executed network command. Note in Fig. 4.1 we compute $D_{KL}(P||Q)$.

We observe in Figs. 4.1–4.2 that the KLD between the distributions under the conditions of intrusion and no intrusion for the metrics `cpu_percent`, `num_clients`, `num_open_connections`, and `num_processes` is negligible. This indicates that these metrics provide little information for profiling network commands. Conversely, metrics with high KLD offer better potential for profiling network commands. Table 4.2 displays the median, the 90th percentile, and the 75th percentile for each metric and $D_{KL}(P||Q)$ and $D_{KL}(Q||P)$. Here P represents the probability distribution for an executed network command, and Q represents the probability distribution for no executed command. The KLD values for each executed network command in CSLE are aggregated, and the summary statistics are presented (median, 90th percentile, and 75th percentile).

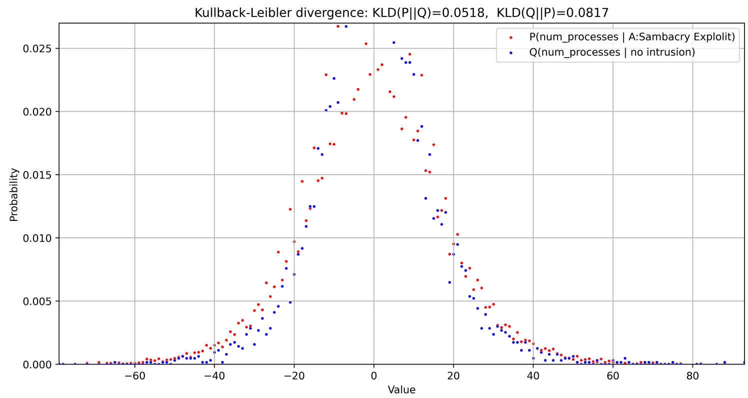

Figure 4.3: Probability distribution of collected data, given the executed network command *Sambacry Exploit*, and no intrusion for the metric `num_processes`.

Metric	Median	P90	P75
$D_{KL}(P\|\|Q)$ (all metric)	0.57325	7.12564	2.40907
$D_{KL}(Q\|\|P)$ (all metrics)	0.18455	4.50843	0.76055
$D_{KL}(P\|\|Q)$ (alerts_weighted_by_priority)	1.4665	8.10138	7.6498
$D_{KL}(Q\|\|P)$ (alerts_weighted_by_priority)	1.0403	8.56346	8.5554
$D_{KL}(P\|\|Q)$ (cpu_percent)	0.0163	0.02468	0.02
$D_{KL}(Q\|\|P)$ (cpu_percent)	0.0116	0.01626	0.015
$D_{KL}(P\|\|Q)$ (default-login-attempt_alerts)	0.9638	5.3642	1.9815
$D_{KL}(Q\|\|P)$ (default-login-attempt_alerts)	0.1973	1.7501	0.4112
$D_{KL}(P\|\|Q)$ (mem_current)	1.094	3.57804	2.403
$D_{KL}(Q\|\|P)$ (mem_current)	0.267	0.94012	0.5888
$D_{KL}(P\|\|Q)$ (net_rx)	1.5314	4.70584	3.0647
$D_{KL}(Q\|\|P)$ (net_rx)	0.3677	1.1806	0.7614
$D_{KL}(P\|\|Q)$ (net_tx)	2.508	7.05184	5.0856
$D_{KL}(Q\|\|P)$ (net_tx)	0.5377	2.69686	1.4292
$D_{KL}(P\|\|Q)$ (num_clients)	0.0185	0.03826	0.0225
$D_{KL}(Q\|\|P)$ (num_clients)	0.0167	0.03376	0.0212
$D_{KL}(P\|\|Q)$ (num_failed_login_attempts)	0.1161	0.91444	0.4787
$D_{KL}(Q\|\|P)$ (num_failed_login_attempts)	0.1487	1.10658	0.4403
$D_{KL}(P\|\|Q)$ (num_open_connections)	0.0349	0.05124	0.0428
$D_{KL}(Q\|\|P)$ (num_open_connections)	0.0727	0.0959	0.0785
$D_{KL}(P\|\|Q)$ (num_processes)	0.0276	0.04412	0.0313
$D_{KL}(Q\|\|P)$ (num_processes)	0.0384	0.07052	0.0473
$D_{KL}(P\|\|Q)$ (pids)	1.0486	3.2118	2.3179
$D_{KL}(Q\|\|P)$ (pids)	0.1979	0.7668	0.4363
$D_{KL}(P\|\|Q)$ (priority_3_alerts)	0.9635	10.25746	9.2065
$D_{KL}(Q\|\|P)$ (priority_3_alerts)	0.1968	11.02596	10.8614
$D_{KL}(P\|\|Q)$ (priority_4_alerts)	1.3253	9.15206	8.2429
$D_{KL}(Q\|\|P)$ (priority_4_alerts)	0.326	9.95748	9.9077
$D_{KL}(P\|\|Q)$ (successful-user_alerts)	1.3253	9.15206	8.2429
$D_{KL}(Q\|\|P)$ (successful-user_alerts)	0.326	9.95748	9.9077
$D_{KL}(P\|\|Q)$ (total_alerts)	0.5874	4.1155	3.7074
$D_{KL}(Q\|\|P)$ (total_alerts)	0.1887	4.49898	4.4922
$D_{KL}(P\|\|Q)$ (warning_alerts)	0.5874	4.1155	3.7074
$D_{KL}(Q\|\|P)$ (warning_alerts)	0.1887	4.49898	4.4922
$D_{KL}(P\|\|Q)$ (rate)	6.3404	9.9754	8.1125
$D_{KL}(Q\|\|P)$ (rate)	1.8725	2.8199	2.1121
$D_{KL}(P\|\|Q)$ (suspicious-login_alerts)	15.9015	15.9015	15.9015
$D_{KL}(Q\|\|P)$ (suspicious-login_alerts)	7.4204	8.6843	8.1022
$D_{KL}(P\|\|Q)$ (blk_write)	2.8404	2.8404	2.8404
$D_{KL}(Q\|\|P)$ (blk_write)	0.9904	0.9904	0.9904

Table 4.2: KLD Statistics

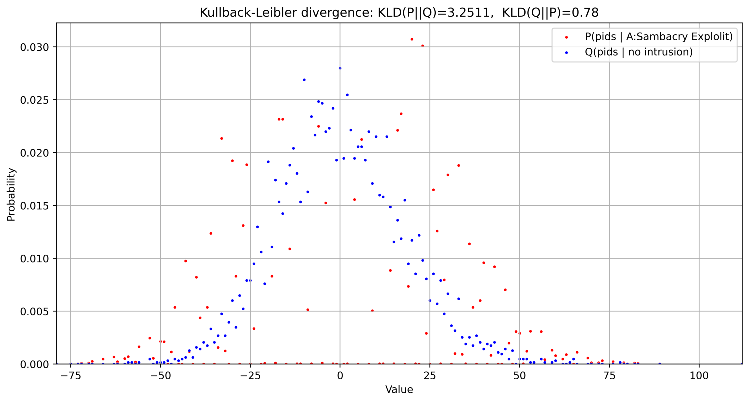

Figure 4.4: Probability distribution of collected data, given the executed network command *Sambacry Exploit*, and no intrusion for the metric `pids`.

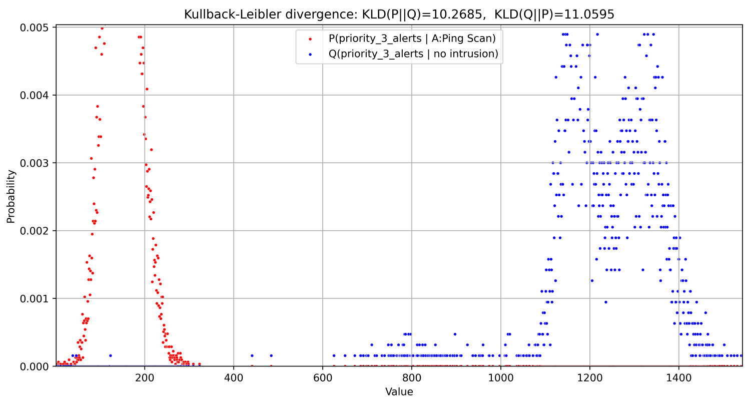

Figure 4.5: Probability distribution of collected data, given the executed network command *Ping scan*, and no intrusion for the metric `priority_3_alerts`.

Figures 4.3–4.5 display the probability distributions of collected data for a network command and no intrusion with a KLD value in the 90th percentile. Figure 4.3 shows the distributions for the metric `num_processes`, the KLD value is 0.0518 and 0.0817 for $D_{KL}(P||Q)$ and $D_{KL}(Q||P)$, respectively. We visually see that the distributions are similar, proven by the low KLD values.

A higher KLD value is observed for the metric `pids` using the same network command, seen in Fig. 4.4. Having KLD values of 3.2511 and 0.75 for $D_{KL}(P||Q)$ and $D_{KL}(Q||P)$. Noticeably, we can see how Fig. 4.4 is more scattered compared to Fig. 4.3, indicating that the scattered plot could contain useful data for profiling *Sambacry Exploit*.

Lastly, Fig. 4.5 shows two distributions that differ significantly. Specifically, the KLD values for $D_{KL}(P||Q)$ and $D_{KL}(Q||P)$ are 10.2685 and 11.0595, respectively. Again, we can visually note a significant difference in the two distributions.

Our investigation of the metrics suggests that some metrics contain useful information to enable accurate profiling based on system measurements. The metrics that appear to be most significant for profiling are `alerts_weighted_by_priority`, `priority_3_alerts`, `priority_4_alerts`, `successful-user_alerts`, `total_alerts`, and `warning_alerts`. We treat a large discrepancy between the median value and the 90th and 75th percentile to indicate that a metric contains valuable information for profiling.

4.3.2 Attack profiling through hidden Markov models

We formulate Prob. 3 as the problem of finding the most likely state sequence in a HMM [22], and we refer to our solution as the HMM profiler. Let the set of $\mathbb{A}$ (Def. 1) represent the hidden states, and let the set of infrastructure metrics $\mathcal{O}$ represent the observation space. The estimation problem can then be stated as

$$(a_1^\star, \dots, a_N^\star) = arg \max_{a_i \in \mathbb{A}} P(a_1, \dots, a_N \mid o_1, \dots, o_N) \qquad o_i \in \mathcal{O}, \tag{4.14}$$

where $P(a'|a)$ can be defined based on domain knowledge about possible sequences of attack commands. We define the state NO INTRUSION as the absence of an intrusion activity. This is included in the hidden states $\mathbb{A}$. The model adapts to this by adding it to the state-transition probabilities $P(a'|a)$, and by calculating the $P(o|a = \text{NO INTRUSION})$ and adding it to the discrete output probabilities $P(o|a)$ in our model.

We use an HMM for the probabilistic profiler because it allows profiling based on only system metrics, which is essential for real time attacker profiling. Another reason why the HMM is well-suited to the task of attacker profiling is that it can model probabilistic transitions, which allows to capture the uncertainty of an attacker's actions. Alternative methods to the HMM exist, for example deep learning. We choose HMM for its simplicity and good experimental results. The algorithm used is very well-established, providing reliability for our solution. Moreover, an HMM typically requires less computational resources than a deep learning model, making it more suitable for real time analysis.

We solve (4.14) using the Viterbi algorithm [22], which is given in Appendix A.7. The complexity of the Viterbi algorithm is $\mathcal{O}(NT^2)$, where N is the number of hidden states and T is the length of the observed sequence. Using this approach, we address the challenge posed by Prob. 3, where the attacker's actions are unknown and only system metrics are available.

4.3.2.1 Evaluation of HMM profiling

The HMM profiler is evaluated with generated sample sequences of attacker actions (states) and observations. Every generated sample state sequence $S_{sample} = \{a_1, \ldots, a_N\}$ begins in the initial state NO INTRUSION, and we define p as being the probability of remaining in the state NO INTRUSION. The transitions to an intrusion state are based on the domain knowledge from possible sequences of attack commands in CSLE. The expected value $\mathbb{E}$ of staying in the state NO INTRUSION is calculated using the geometric distribution with the parameter p in $(0, 1)$.

When we generate a sample sequence $S_{sample} = \{a_1, \ldots, a_N\}$, we also generate a belonging observation sequence $O_{sample} = \{o_1, \ldots, o_N\}$, based on the probabilities from our observation distribution $P(o|a)$. Given the sample observation sequence, O_{sample}, we want to find the most likely sequence $S^* = \{a_1^*, \ldots, a_N^*\}$ using our model λ. The intrusion start time is defined as the time when the HMM transitions from the initial state, NO INTRUSION, to any other state in the state space. We want to identify the start time of an intrusion by comparing the sampled state sequence with the sequence predicted by the model. The evaluation of the model is based on the fraction of correctly profiled actions and correctly identifying the start of an intrusion. Let l define the number of sample sequences. We can then define the following accuracy metrics [16].

acc_{action}: the fraction of correctly profiled single actions.

$$acc_{action} = \frac{1}{l} \sum_{i=1}^{l} \sum_{t=1}^{T} \mathbb{1}(a_t^i = a_t^{i\,*}).$$

acc_{start}: the fraction of correctly detected intrusion starts. Let a_{start} define the start of an intrusion.

$$acc_{start} = \frac{1}{l} \sum_{i=1}^{l} \mathbb{1}(a_{start} = a_{start}^*).$$

The testbed metrics used to define the observation in the HMM are the same metrics presented in 4.3.1. We use the Laplace smoothing technique [23] to make sure that the observation distribution $P(o \mid s)$ does not have any events with zero probability.

4.4 Implementation

We implement an attack profiler that solves Prob. 1–3 in Python. The code is available in appendices A.1 and A.2. Our implementation uses the library `MitreAttackData`* to implement the correspondences in (4.1-4.4). In CSLE, each network command is uniquely identified by an ID, such as TCP_SYN_STEALTH_SCAN_HOST. The IDs serve as references that are linked to the technique correspondences. To facilitate the mapping process, we use Common Attack Pattern Enumeration and Classification (CAPEC)™† to map network commands to techniques. The mappings of the network commands are available in Appendix A.3. The implementation of the attack graph is available in Appendix A.4, and the implementation of Algorithm 1 is available in Appendix A.5. The implementation of the HMM profiler, solving Prob. 3, is available in Appendix A.8–A.12. The HMM profiler is tested using sample sequences, based on the model, the implementation of the generation of sample sequences is available in appendices A.13. The implementations are also available in https://github.com/Limmen/csle/tree/1e247c7705cee0c38ea44595308c6ba9dd49cbd6/simulation-system/libs/csle-attack-profiler

*https://github.com/mitre-attack/mitreattack-python
†https://capec.mitre.org

Chapter 5

Results and discussion

In this chapter, we evaluate the attack profilers described in Chapter 4. We also discuss the implications of our results.

5.1 Results

We evaluate Algorithm 1 by pruning tactics from three attack sequences in CSLE. The sequences are shown in Tab. 5.1 with corresponding attack graphs in Figs. 5.1–5.3. The evaluation results are shown in Fig. 5.4.

sequence 1	sequence 2	sequence 3
1. TCP SYN SCAN	1. PING SCAN	1. PING SCAN
2. SSH DICTIONARY ATTACK	2. SAMBACRY EXPLOIT	2. SAMBACRY EXPLOIT
3. TELNET DICTIONARY ATTACK	3. NETWORK SERVICE LOGIN	3. NETWORK SERVICE LOGIN
4. FTP DICTIONARY ATTACK	4. INSTALL TOOLS	4. INSTALL TOOLS
5. NETWORK SERVICE LOGIN	5. PING SCAN	5. PING SCAN
6. INSTALL TOOLS	6. DVWA SQL INJECTION	6. SSH DICTIONARY ATTACK
7. SSH BACKDOOR	7. NETWORK SERVICE LOGIN	7. NETWORK SERVICE LOGIN
8. TCP SYN SCAN	8. INSTALL TOOLS	8. CVE 2010 0426
9. SHELLSHOCK EXPLOIT	9. PING SCAN	9. PING SCAN
10. NETWORK SERVICE LOGIN	10. CVE 2015 1427 EXPLOIT	10. DVWA SQL INJECTION
11. INSTALL TOOLS	11. NETWORK SERVICE LOGIN	11. NETWORK SERVICE LOGIN
12. SSH DICTIONARY ATTACK	12. INSTALL TOOLS	12. INSTALL TOOLS
13. NETWORK SERVICE LOGIN	13. PING SCAN	13. PING SCAN
14. CVE 2010 0426	14. SAMBACRY EXPLOIT	14. CVE 2015 1427
15. TCP SYN SCAN	15. NETWORK SERVICE LOGIN	15. NETWORK SERVICE LOGIN
	16. INSTALL TOOLS	16. INSTALL TOOLS
	17. PING SCAN	17. PING SCAN

Table 5.1: Attack sequences 1, 2, and 3.

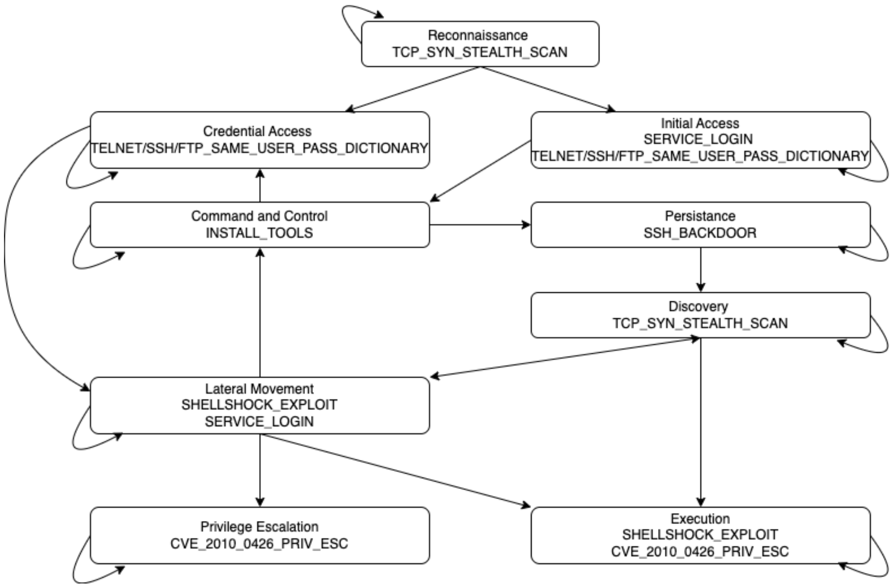

Figure 5.1: Attack graph for sequence 1 in Tab. 5.1. The tactic and the associated network command(s) are shown in the nodes.

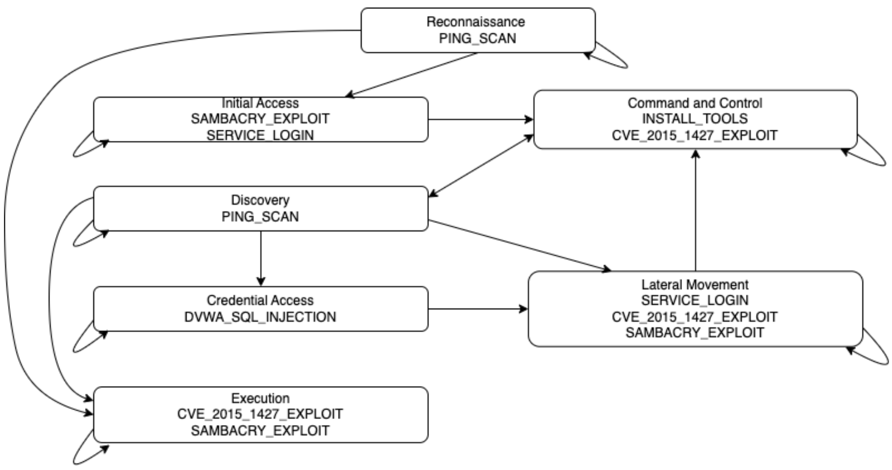

Figure 5.2: Attack graph for sequence 2 in Tab. 5.1. The tactic and the associated network command(s) are shown in the nodes.

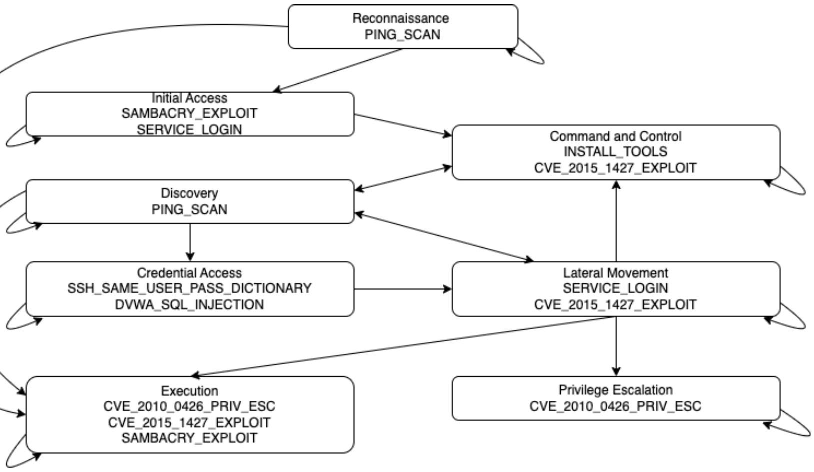

Figure 5.3: Attack graph for sequence 3 in Tab. 5.1. The tactic and the associated network command(s) are shown in the nodes.

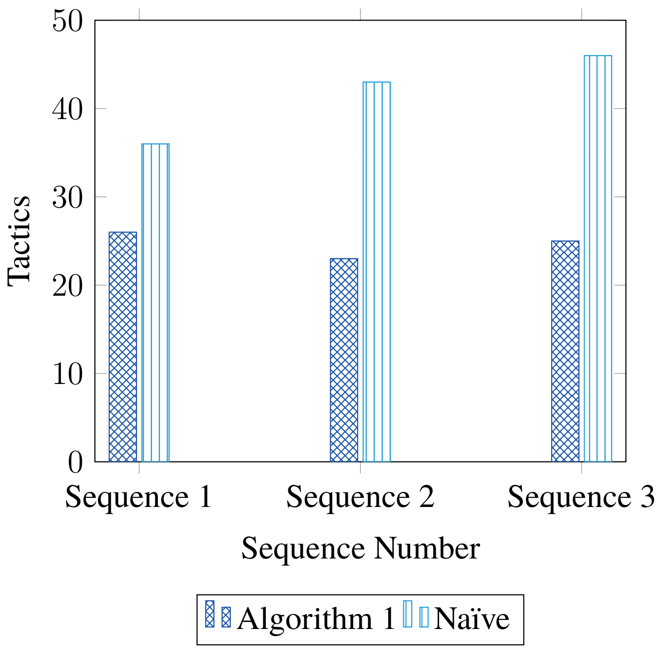

Figure 5.4: Comparison of the three sequences in Tab. 5.1 using Algorithm 1 and the naïve approach where each network command is profiled independently without an attack graph (Def. 2).

We observe in Fig. 5.4 that Algorithm 1 produces more precise profiling than the naïve profiler on all three sequences. On sequence 1 (Tab. 5.1) it reduces the total number of tactics from 36 to 26; on sequence 2 (Tab. 5.1) it reduces the number of tactics from 43 to 23; and on sequence 3 (Tab. 5.1) it reduces the number of tactics from 46 to 25.

We evaluate the HMM profiler, solving Prob. 3, using statistics from the testbed (see Section 3.1). The transition matrix is derived from sequences 1, 2, and 3 (Tab. 5.1) and is presented in Tab. 5.2. The observation matrices for the different metrics are too large to present — how these are calculated can be found in Appendix A.10. No metric data for the attacker action SSH BACKDOOR is available, therefore the observation distribution probabilities for this attacker action are calculated using the Laplace smoothing technique as mentioned in Section 4.3.2.1.

We use $p = 0.1$ in our experiments (recall that $1 - p$ is the probability of remaining in the state NO INTRUSION). The intrusion sequence I_{seq} length varies from 1 to 10. Thus, each episode starts at time t, and the intrusion starts at a random time drawn from the geometric distribution $Ge(p)$. The intrusion starts a sequence of length I_{seq}. From Tab. 5.1, we can see that the intrusion sequence starts in either the state PING SCAN or TCP SYN SCAN. We evaluate using 1000 generated sample sequences, including the sampled observation sequences for each I_{seq}. The results are shown in Figs. 5.5–5.6.

NO INTRUSION	$\frac{1}{10}$	$\frac{3}{10}$	0	0	0	0	0	0	0	0	$\frac{6}{10}$	0	0	0
TCP SYN SCAN	0	0	$\frac{1}{2}$	0	0	0	0	0	$\frac{1}{2}$	0	0	0	0	0
SSH DICTIONARY ATTACK	0	0	0	$\frac{1}{3}$	0	$\frac{2}{3}$	0	0	0	0	0	0	0	0
TELNET DICTIONARY ATTACK	0	0	0	0	1	0	0	0	0	0	0	0	0	0
FTP DICTIONARY ATTACK	0	0	0	0	0	1	0	0	0	0	0	0	0	0
NETWORK SERVICE LOGIN	0	0	0	0	0	0	$\frac{9}{11}$	0	0	$\frac{2}{11}$	0	0	0	0
INSTALL TOOLS	0	0	$\frac{1}{9}$	0	0	0	0	$\frac{1}{9}$	0	0	$\frac{7}{9}$	0	0	0
SSH BACKDOOR	0	1	0	0	0	0	0	0	0	0	0	0	0	0
SHELLSHOCK EXPLOIT	0	0	0	0	0	1	0	0	0	0	0	0	0	0
CVE 2010 0426	0	$\frac{1}{2}$	0	0	0	0	0	0	0	0	$\frac{1}{2}$	0	0	0
PING SCAN	0	0	$\frac{1}{8}$	0	0	0	0	0	0	0	0	$\frac{3}{8}$	$\frac{1}{4}$	$\frac{1}{4}$
SAMBACRY EXPLOIT	0	0	0	0	0	1	0	0	0	0	0	0	0	0
DVWA SQL INJECTION	0	0	0	0	0	1	0	0	0	0	0	0	0	0
CVE 2015 1427 EXPLOIT	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Table 5.2: Transition Matrix for Sequence 1, 2, and 3 (see Tab. 5.1)

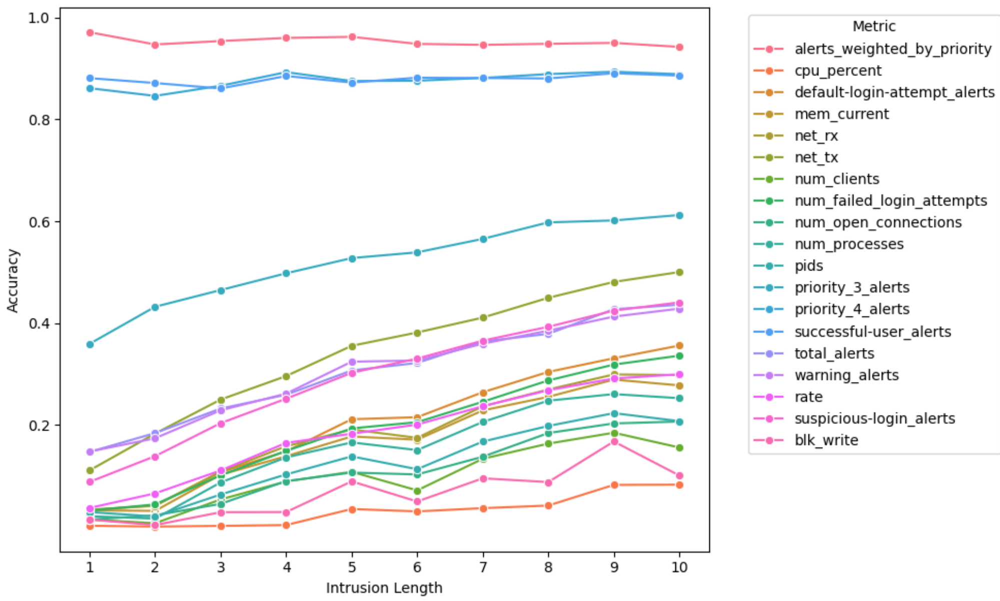

Figure 5.5: The fraction of correct profiled single actions (acc_{action}) for different intrusion lengths. Using Sequences 1, 2, and 3 (see Tab. 5.1)

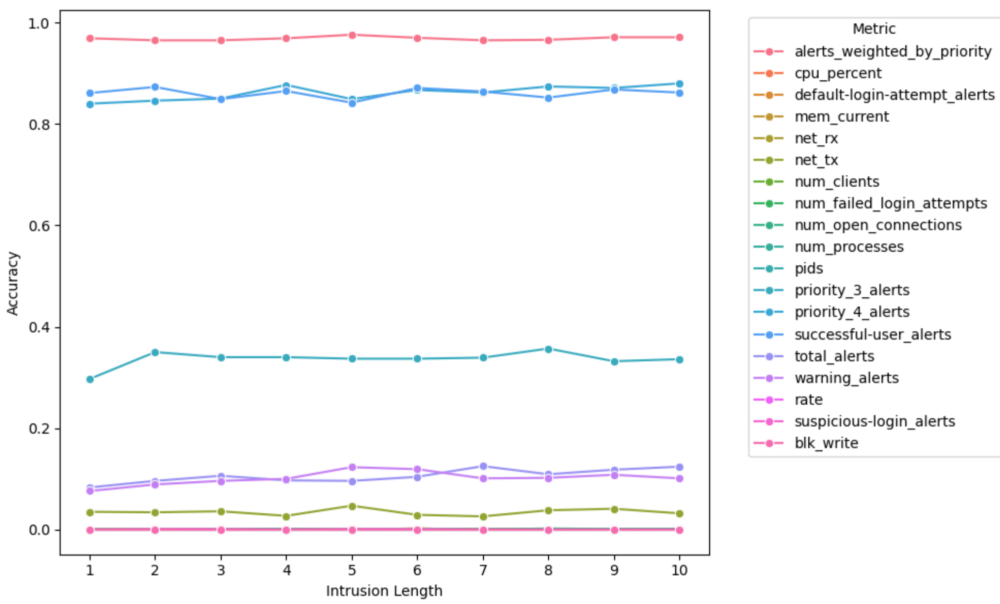

Figure 5.6: The fraction of correct detecting intrusion starts (acc_{start}) for different intrusion lengths. Using Sequences 1, 2, and 3 (see Tab. 5.1)

In Fig. 5.5, we observe a high accuracy for the three metrics ALERTS WEIGHTED BY PRIORITY, PRIORITY 4 ALERTS, and SUCCESSFUL USER ALERTS. Similarly, in Fig. 5.6 we observe a high accuracy of detecting the intrusion start time for the same three metrics. Furthermore, we can observe an increasing acc_{action} for most metrics when I_{seq} increases. This can be explained by some metrics' difficulties in identifying TCP SYN SCAN and PING SCAN from observations. The same phenomenon is also shown in Fig. 5.6, where we observe a constant low acc_{start} for some metrics and can see an increasing acc_{action} in Fig. 5.5 for those metrics. While some metrics are poor at profiling the actions initiating an intrusion sequence, they are more effective at identifying subsequent actions.

Similarly, we test the HMM profiler using sequences 2, and 3 (see Tab. 5.1). Table 5.3 shows the corresponding transition matrix. The same conditions as for the previous test are set. Note that an intrusion sequence only starts in the state PING SCAN. We have 1000 generated sample sequences, including the sampled observation sequences. The results are shown in Figs. 5.7–5.8.

NO INTRUSION	$\frac{1}{10}$	0	0	0	0	$\frac{9}{10}$	0	0	0
SSH DICTIONARY ATTACK	0	0	1	0	0	0	0	0	0
NETWORK SERVICE LOGIN	0	0	0	$\frac{4}{7}$	0	$\frac{3}{7}$	0	0	0
INSTALL TOOLS	0	0	0	0	0	1	0	0	0
CVE 2010 0426	0	0	0	0	0	1	0	0	0
PING SCAN	0	$\frac{1}{8}$	0	0	0	0	$\frac{3}{8}$	$\frac{2}{8}$	$\frac{2}{8}$
SAMBACRY EXPLOIT	0	0	1	0	0	0	0	0	0
DVWA SQL INJECTION	0	0	1	0	0	0	0	0	0
CVE 2015 1427	0	0	1	0	0	0	0	0	0

Table 5.3: Transition matrix for Sequences 2 and 3.

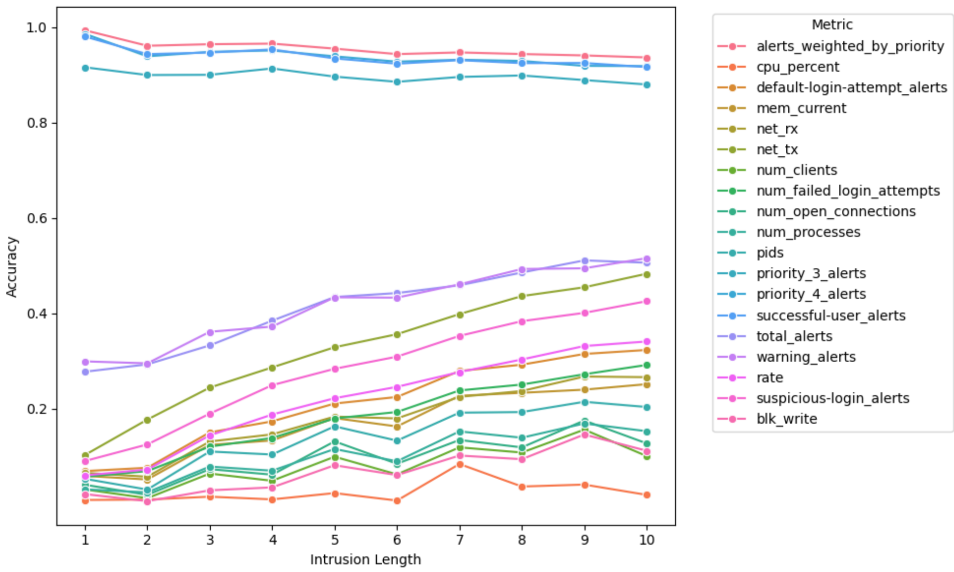

Figure 5.7: The fraction of correct profiled single actions (acc_{action}) for different intrusion lengths. Using Sequences 2 and 3 (see Tab. 5.1)

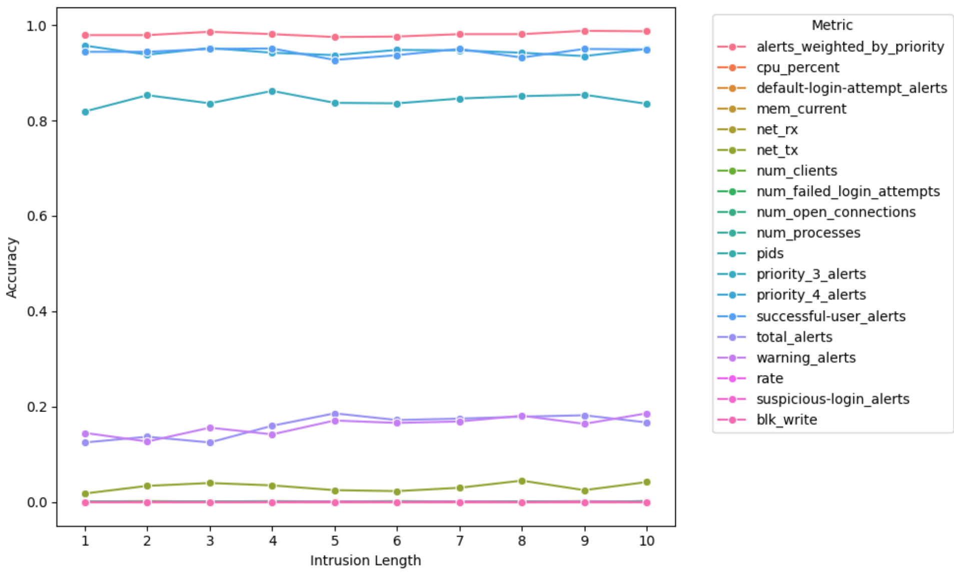

Figure 5.8: The fraction of correct detecting intrusion starts (acc_{start}) for different intrusion lengths. Using Sequences 2 and 3 (see Tab. 5.1)

In Figs. 5.7–5.8 we observe similar results to those shown in Figs. 5.5–5.6, with slightly better performance among all metrics generally. This is the expected result due to a simpler model with fewer hidden states. One can note that using the metric PRIORITY 3 ALERTS as observation in the HMM leads to significantly higher accuracy than the previous test. We conclude that the metric PRIORITY 3 ALERTS is better capable of detecting the start of an intrusion when PING SCAN is executed rather than TCP SYN SCAN, indicated in Fig. 5.8 with high acc_{start} for metric PRIORITY 3 ALERTS. Again, note that TCP SYN SCAN is not a state in this test setup (see Tab. 5.3).

Lastly, we evaluate the fraction of correctly profiled single actions (acc_{action}), omitting the state NO INTRUSION. Hence, this context makes the acc_{start} irrelevant. The transition matrix derived from Sequences 1, 2, and 3 (Tab. 5.1) is presented in Tab. 5.4. Again, no metric data for the attacker action SSH BACKDOOR is available. The observation distribution probabilities are calculated using the Laplace smoothing technique. The intrusion sequence I_{seq} varies from 1 to 10 and the initial states are TCP SYN SCAN, and PING SCAN with probabilities $\frac{1}{3}$ and $\frac{2}{3}$, respectively. We evaluate using 1000 generated sample sequences for each I_{seq}.

TCP SYN SCAN	0	$\frac{1}{2}$	0	0	0	0	0	$\frac{1}{2}$	0	0	0	0	0
SSH DICTIONARY ATTACK	0	0	$\frac{1}{3}$	0	$\frac{2}{3}$	0	0	0	0	0	0	0	0
TELNET DICTIONARY ATTACK	0	0	0	1	0	0	0	0	0	0	0	0	0
FTP DICTIONARY ATTACK	0	0	0	0	1	0	0	0	0	0	0	0	0
NETWORK SERVICE LOGIN	0	0	0	0	0	$\frac{9}{11}$	0	0	$\frac{2}{11}$	0	0	0	0
INSTALL TOOLS	0	$\frac{1}{9}$	0	0	0	0	$\frac{1}{9}$	0	0	$\frac{7}{9}$	0	0	0
SSH BACKDOOR	1	0	0	0	0	0	0	0	0	0	0	0	0
SHELLSHOCK EXPLOIT	0	0	0	0	1	0	0	0	0	0	0	0	0
CVE 2010 0426	$\frac{1}{2}$	0	0	0	0	0	0	0	0	$\frac{1}{2}$	0	0	0
PING SCAN	0	$\frac{1}{8}$	0	0	0	0	0	0	0	0	$\frac{3}{8}$	$\frac{1}{4}$	$\frac{1}{4}$
SAMBACRY EXPLOIT	0	0	0	0	1	0	0	0	0	0	0	0	0
DVWA SQL INJECTION	0	0	0	0	1	0	0	0	0	0	0	0	0
CVE 2015 1427 EXPLOIT	0	0	0	0	1	0	0	0	0	0	0	0	0

Table 5.4: Transition matrix for sequences 1, 2, and 3 (see Tab. 5.1), excluding the state NO INTRUSION.

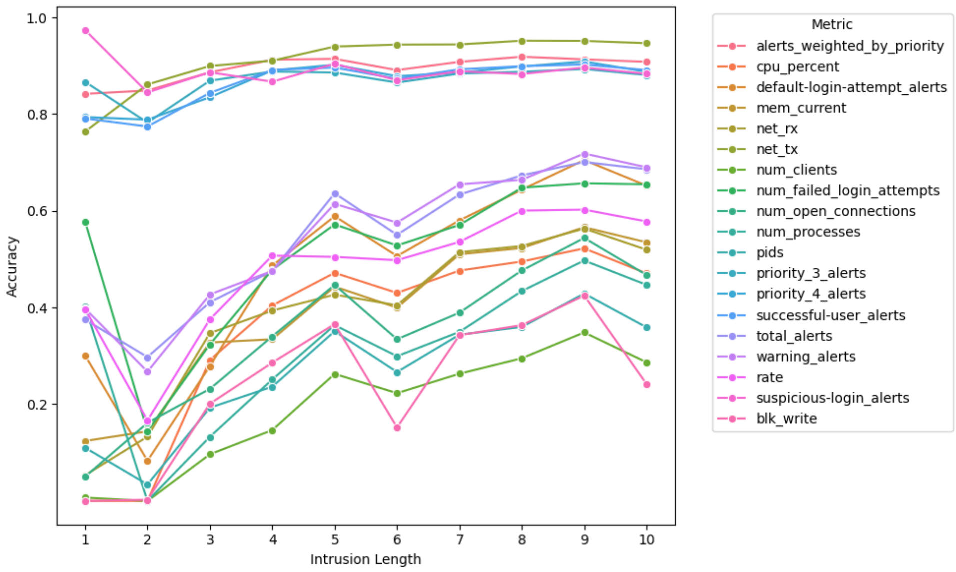

Figure 5.9: The fraction of correct detecting intrusion starts (acc_{start}) for different intrusion lengths. Using Sequences 2 and 3 (see Tab. 5.1). The state NO INTRUSION is not in the model.

Omitting the state NO INTRUSION in our model gives a better profiling accuracy among several metrics. In Fig. 5.9, we can observe six metrics that lead to high accuracy for all I_{seq}. Notably, SUSPICIOUS LOGIN ALERTS and NET TX have high accuracy compared to previous tests. We conclude that the observations of these metrics for NO INTRUSION are similar to PING SCAN and TCP SYN SCAN, therefore leading to poor accuracy of both acc_{action} and $acc_{intrusion}$.

In summary, the HMM profiler can be used for online profiling to detect the start of intrusion and to profile attacker actions. The results show high accuracy for acc_{action} and acc_{start} among some metrics, as shown in Figs. 5.5–5.8. For most metrics, acc_{action} increases for an increasing intrusion length. We believe this is because some metrics are better at profiling certain attacks than others. In this case, most metrics have difficulties detecting the start of intrusion, which starts with either the action TCP SYN SCAN or PING SCAN. If the intrusion is detected, observations in other metrics could be valuable to correctly profile an ongoing attack, which Fig. 5.9 highlights.

Notably, the metrics with high Kullback-Leibler divergence are those that perform well in the HMM profiler. This is expected since the high KLD means greater difference in the distributions. Conversely, the metrics with low KLD values perform poorly. This implies that when these metrics are used, it is difficult for the HMM to distinguish the difference between the state NO INTRUSION and the intrusion states.

Compared with the results presented by Wang and Stadler [16], we do not perform preprocessing of the infrastructure metrics before training the HMM profiler. By contrast, using clustering techniques, Wang and Stadler [16] pre-process the observation space by mapping the observations to six possible values. In comparison, the size of the observation space in our experiments varies between 16 (BLK WRITE) and 53117 (NET TX). Another difference between our experimental setup and the setup used in [16] is that we use $p = 0.1$ whereas [16] uses $p = 0.2$. Finally, another difference is that we evaluated the HMM profiler using sample sequences based on the model. In contrast, in [16], the data is divided uniformly at random, 70% for training and 30% for evaluation.

5.2 Discussion

The key findings from the evaluation are summarized as follows.

(i) The profiler solving Prob. 1 and Prob. 2 can achieve more accurate profiling of an attacker action (Def. 1) than the naïve profiler. The main enabler of the improved accuracy is the pruning of attack techniques based on the attack graph (Def. 2).

(ii) The analyzed metrics with high KLD correlate with the metrics showing high accuracy for profiling a single action and detecting the start of an intrusion.

(iii) The accuracy, acc_{action} is increasing for longer intrusion lengths for metrics with low acc_{start}.

We answer the research questions posed in §1.3 as follows.

- *How can we model different types of attacks in a general framework?*
  - We model an attacker action in CSLE using Def. 1) and we model an attack as a sequence of attacker actions.
- *How can we automatically profile attacks using the model?*
  - The attacker actions are profiled automatically using Alg. 1 and the offline profiler that solves Prob. 1 and Prob. 2.

- *How can we automatically profile attacks using only system measurements?*
  - The HMM profiler, solving Prob. 3, shows how system measurements can be used to profile attacker actions online.

Limitations While the results show that the attack graph (Def. 2) can allow for more accurate profiling of attacks, it is important to acknowledge that it relies on domain knowledge about the attacker. If such knowledge is not available, the attack graph provides little value. Another limitation of our framework for attacker profiling is that each attacker action in CSLE is manually labeled with the corresponding techniques in MITRE ATT&CK®before our experiments. This labeling is not trivial, requiring knowledge about attacker actions. We utilize the knowledge base CAPEC™ to aid us in this work. Finally, another limitation of our framework is that the HMM profiler requires a dataset of attack traces to train, which may not always be available in practice.

Chapter 6

Conclusions and Future work

In this thesis, we present a framework for automated profiling of cyber attacks based on MITRE ATT&CK. The framework includes two components: (1) a component for automated mapping of sequences of attacker actions to the corresponding tactics and techniques in MITRE ATT&CK; and (2) a component for probabilistic profiling of attacker actions based on testbed measurements.

The first component allows for offline (forensic) attacker profiling. It takes as input a sequence of attacker actions and outputs a corresponding sequence of attack techniques and tactics in MITRE ATT&CK. A key challenge when developing this profiler is that a single attacker action often maps to many techniques and tactics in MITRE ATT&CK, which limits the value of the profiling. To make the profiling more precise, we introduce a novel algorithm that leverages an attack graph to contextualize the attack sequence. This contextualization allows us to profile the attacker sequence more accurately and provides a natural way to encode domain expertise into the attack profiler.

The second component allows for online (real-time) attacker profiling. It takes as input a sequence of infrastructure metrics (e.g., log files and alerts) and outputs the most likely sequence of attack techniques and tactics. To find the most effective infrastructure metrics for profiling, we analyze many possible metrics and select the metrics that provide the most information for distinguishing between different attack stages, which we quantify using the Kullback-Leibler divergence. We then model the relation between attacker actions and values of the chosen metric using a hidden Markov model, which allows us to compute the most likely attacker action sequence using Viterbi's algorithm.

The experimental part of this thesis includes extensive profiling of emulated attacks in the Cyber Security Learning Environment (CSLE), which is a platform for emulating attacks and defenses in virtualized IT environments. From the results of running the first profiler (i.e., the offline profiler), we see that the attack graph leads to accurate profiling of sequences by allowing the pruning of the sets of attack techniques and tactics. When evaluating the second profiler (i.e., the HMM profiler), we find that the performance depends heavily on the choice of infrastructure metrics, where metrics based on alerts from an intrusion detection system tend to be the most useful for the types of attacks that we study.

In conclusion, this work demonstrates how we can automate the profiling of cyber attacks, thereby reducing the need for domain experts to conduct forensic analysis. From a sustainability perspective, the primary implication of this research is the potential for cost reduction and improved cybersecurity.

Future work The implementation of the offline profiler relies on domain knowledge to label the network commands in CSLE with the corresponding techniques in MITRE ATT&CK®. Investigating how to automatically map the commands to MITRE ATT&CK® based on Open Source Intelligence (OSINT) is a promising direction for future research. This would reduce the effort to label new network commands added to the platform.

The next step for the HMM profiler is to extend the profiler to consider more attack sequences. In this work, we focused on three possible attack sequences, which led to a relatively simple transition matrix. Extending the HMM profiler to consider more attack sequences would allow us to investigate the usefulness of the HMM profiler when the uncertainty about the attacker is high.

References

[1] K. Hammar. "The cybersecurity learning environment". Accessed 19-01-2024. (), [Online]. Available: https://limmen.dev/csle/.

[2] K. Hammar and R. Stadler, "Digital twins for security automation", in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, 2023, pp. 1–6. DOI: 10.1109/NOMS56928.2023.10154288.

[3] K. Hammar and R. Stadler, "Learning near-optimal intrusion responses against dynamic attackers", *IEEE Transactions on Network and Service Management*, vol. 21, no. 1, pp. 1158–1177, 2024. DOI: 10.1109/TNSM.2023.3293413.

[4] K. Hammar and R. Stadler, "Learning intrusion prevention policies through optimal stopping", in *2021 17th International Conference on Network and Service Management (CNSM)*, 2021, pp. 509–517. DOI: 10.23919/CNSM52442.2021.9615542.

[5] H. Shahriar, S. North, Y. Lee, and R. Hu, "Server-side code injection attack detection based on kullback-leibler distance", *International Journal of Internet Technology and Secured Transactions*, vol. 5, p. 240, Jan. 2014. DOI: 10.1504/IJITST.2014.065184.

[6] A. Applebaum, D. Miller, B. Strom, C. Korban, and R. Wolf, "Intelligent, automated red team emulation", in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ser. ACSAC '16, Los Angeles, California, USA: Association for Computing Machinery, 2016, pp. 363–373, ISBN: 9781450347716. DOI: 10.1145/2991079.2991111. [Online]. Available: https://doi.org/10.1145/2991079.2991111.

[7] J. Janisch, T. Pevný, and V. Lisý, "Nasimemu: network attack simulator & emulator for training agents generalizing to novel scenarios", in *Computer Security. ESORICS 2023 International Workshops*, S.

Katsikas *et al.*, Eds., Cham: Springer Nature Switzerland, 2024, pp. 589–608, ISBN: 978-3-031-54129-2.

[8] J. Schwartz and H. Kurniawatti, *Nasim: network attack simulator*, https://networkattacksimulator.readthedocs.io/, 2019.

[9] M. Standen, M. Lucas, D. Bowman, T. J. Richer, J. Kim, and D. Marriott, *Cyborg: a gym for the development of autonomous cyber agents*, 2021. arXiv: 2108.09118 [cs.CR].

[10] T. Nguyen, Z. Chen, K. Hasegawa, K. Fukushima, and R. Beuran, “Pengym: pentesting training framework for reinforcement learning agents”, in *Proceedings of the 10th International Conference on Information Systems Security and Privacy (ICISSP 2024)*, SCITEPRESS – Science and Technology Publications, Lda., 2024, pp. 498–509, ISBN: 978-989-758-683-5. DOI: 10.5220/0012367300003648.

[11] A. Chowdhary, D. Huang, J. S. Mahendran, D. Romo, Y. Deng, and A. Sabur, “Autonomous security analysis and penetration testing”, in *2020 16th International Conference on Mobility, Sensing and Networking (MSN)*, 2020, pp. 508–515. DOI: 10.1109/MSN50589.2020.00086.

[12] Y. Yang and X. Liu, *Behaviour-diverse automatic penetration testing: a curiosity-driven multi-objective deep reinforcement learning approach*, 2022. arXiv: 2202.10630 [cs.LG].

[13] L. Li, R. Fayad, and A. Taylor, “Cygil: A cyber gym for training autonomous agents over emulated network systems”, *CoRR*, vol. abs/2109.03331, 2021. arXiv: 2109.03331. [Online]. Available: https://arxiv.org/abs/2109.03331.

[14] M. Rodríguez, G. Betarte, and D. Galegari, “Discovering attacker profiles using process mining and the mitre attck taxonomy”, in *Proceedings of the 12th Latin-American Symposium on Dependable and Secure Computing*, ser. LADC ’23, , La Paz, Bolivia, Association for Computing Machinery, 2023, pp. 146–155, ISBN: 9798400708442. DOI: 10.1145/3615366.3615372. [Online]. Available: https://doi-org.focus.lib.kth.se/10.1145/3615366.3615372.

[15] Y. Wu, C. Huang, X. Zhang, and H. Zhou, “Grouptracer: automatic attacker ttp profile extraction and group cluster in internet of things”, *Security and Communication Networks*, vol. 2020, Article ID 8842539, 2020. DOI: 10.1155/2020/8842539.

[16] X. Wang and R. Stadler, *It intrusion detection using statistical learning and testbed measurements*, 2024. arXiv: 2402.13081 [cs.LG].

[17] P. Holgado, V. A. Villagrá, and L. Vázquez, "Real-time multistep attack prediction based on hidden markov models", *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 1, pp. 134–147, 2020. DOI: 10.1109/TDSC.2017.2751478.

[18] Y.-T. Huang, C. Y. Lin, Y.-R. Guo, K.-C. Lo, Y. S. Sun, and M. C. Chen, "Open source intelligence for malicious behavior discovery and interpretation", *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 2, pp. 776–789, 2022. DOI: 10.1109/TDSC.2021.3119008.

[19] S. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "Holmes: real-time apt detection through correlation of suspicious information flows", May 2019, pp. 1137–1152. DOI: 10.1109/SP.2019.00026.

[20] W. U. Hassan, A. Bates, and D. Marino, "Tactical provenance analysis for endpoint detection and response systems", in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1172–1189. DOI: 10.1109/SP40000.2020.00096.

[21] E. Miehling, M. Rasouli, and D. Teneketzis, "Optimal defense policies for partially observable spreading processes on bayesian attack graphs", in *Proceedings of the Second ACM Workshop on Moving Target Defense*, ser. MTD '15, Denver, Colorado, USA: Association for Computing Machinery, 2015, pp. 67–76, ISBN: 9781450338233. DOI: 10.1145/2808475.2808482. [Online]. Available: https://doi.org/10.1145/2808475.2808482.

[22] A. M. Fraser, *Hidden Markov Models and Dynamical Systems*. USA: Society for Industrial and Applied Mathematics, 2008, ISBN: 0898716659.

[23] M. Kikuchi, M. Yoshida, M. Okabe, and K. Umemura, "Confidence interval of probability estimator of laplace smoothing", in *2015 2nd International Conference on Advanced Informatics: Concepts, Theory and Applications (ICAICTA)*, 2015, pp. 1–6. DOI: 10.1109/ICAICTA.2015.7335387.

Appendix A

Implementation details

```python
class AttackProfiler():
    """
    Class represting the attack profile based on the MITRE ATT&CK framework for Enterprise.
    """

    def __init__(self, techniques_tactics: Dict[str,List[str]], mitigations: Dict[str, List[str]], data_sources: Dict[str, List[str]], subtechniques: Dict[str, List[str]], action_id: EmulationAttackerActionId):


        self.techniques_tactics = techniques_tactics
        self.mitigations = mitigations
        self.data_sources = data_sources
        self.subtechniques = subtechniques
        self.action_id = action_id
```

Listing A.1: Constructor for the attack profiler object

```python
def get_attack_profile(attacker_action: EmulationAttackerAction):
        """
        Returns the attack profile of the actions
        """
        mitre_attack_data = MitreAttackData("./src/csle_attack_profiler/enterprise-attack.json")

        attacker_id = attacker_action.id
        attack_mapping = EmulationAttackerMapping.get_attack_info(attacker_id)
```

```
    if attack_mapping == {None} or attack_mapping is None:
        return AttackProfiler({}, {}, {}, {}, None)

    attack_techniques_vals = [technique.value for technique in attack_mapping['techniques']]

    attacker_action_id = attacker_action.id
    techniques_tactics = {}
    mitigations = {}
    data_sources = {}
    sub_techniques = {}
    for technique_name in attack_techniques_vals:
        try:
            obj = mitre_attack_data.get_objects_by_name(technique_name, "attack-pattern")
        except:
            raise RuntimeError("Error in fetching the technique from the MitreAttackData")
        technique = obj[0]
        stix_id = technique.id

        tactics = [phase['phase_name'] for phase in technique.kill_chain_phases]
        techniques_tactics[technique_name] = tactics

        if hasattr(technique, 'x_mitre_data_sources'):
            data_sources[technique_name] = technique.x_mitre_data_sources
        try:
            mitigations_object = mitre_attack_data.get_mitigations_mitigating_technique(stix_id)
            mitigations_list = [mitig['object']['name'] for mitig in mitigations_object]
            mitigations[technique_name] = mitigations_list
        except:
            raise RuntimeError("Error in fetching the mitigations from the MitreAttackData")


    if 'subtechniques' in attack_mapping:
        sub_techniques_mapping = [sub_technique.value for sub_technique in attack_mapping['subtechniques']]
        for st in sub_techniques_mapping:
            try:
                sub_technique_obj = mitre_attack_data.
```

```
get_objects_by_name(st, "attack-pattern")
                    parent_technique_obj = mitre_attack_data.get_parent_technique_of_subtechnique(sub_technique_obj[0].id)
                    sub_techniques[parent_technique_obj[0]['object'].name] = st
                except:
                    raise RuntimeError("Error in fetching the sub-techniques from the MitreAttackData")


        return AttackProfiler(techniques_tactics, mitigations, data_sources, sub_techniques, attacker_action_id)
```

Listing A.2: Implementation of the attack profiler. Profiling a network command

```
class EmulationAttackerMapping():
    """
    Maps EmulationAttackerActionId's to tactics and techniques
    """
    @staticmethod
    def get_attack_info(id: EmulationAttackerActionId):
        """
        Maps id's to tactics and techniques
        """

        mapping = {
            EmulationAttackerActionId.TCP_SYN_STEALTH_SCAN_HOST: {
                "techniques": {Techniques.ACTIVE_SCANNING,
                               Techniques.GATHER_VICTIM_HOST_INFORMATION,
                               Techniques.NETWORK_SERVICE_DISCOVERY }
            },
            EmulationAttackerActionId.TCP_SYN_STEALTH_SCAN_ALL: {
                "techniques": {Techniques.ACTIVE_SCANNING,
                               Techniques.GATHER_VICTIM_HOST_INFORMATION,
                               Techniques.NETWORK_SERVICE_DISCOVERY }
            },
            EmulationAttackerActionId.PING_SCAN_HOST: {
                "techniques": {Techniques.ACTIVE_SCANNING,
```

```
                                        Techniques.
    GATHER_VICTIM_HOST_INFORMATION,
                                        Techniques.
    NETWORK_SERVICE_DISCOVERY }
                },
                EmulationAttackerActionId.PING_SCAN_ALL: {
                    "techniques": {Techniques.ACTIVE_SCANNING,
                                        Techniques.
    GATHER_VICTIM_HOST_INFORMATION,
                                        Techniques.
    NETWORK_SERVICE_DISCOVERY }
                },
                EmulationAttackerActionId.UDP_PORT_SCAN_HOST: {
                    "techniques": {Techniques.ACTIVE_SCANNING,
                                        Techniques.
    GATHER_VICTIM_HOST_INFORMATION,
                                        Techniques.
    NETWORK_SERVICE_DISCOVERY }
                },
                EmulationAttackerActionId.UDP_PORT_SCAN_ALL: {
                    "techniques": {Techniques.ACTIVE_SCANNING,
                                        Techniques.
    GATHER_VICTIM_HOST_INFORMATION,
                                        Techniques.
    NETWORK_SERVICE_DISCOVERY }
                },
                EmulationAttackerActionId.
    TCP_CON_NON_STEALTH_SCAN_HOST: {
                    "techniques": {Techniques.ACTIVE_SCANNING,
                                        Techniques.
    GATHER_VICTIM_HOST_INFORMATION,
                                        Techniques.
    NETWORK_SERVICE_DISCOVERY }
                },
                EmulationAttackerActionId.
    TCP_CON_NON_STEALTH_SCAN_ALL: {
                    "techniques": {Techniques.ACTIVE_SCANNING,
                                        Techniques.
    GATHER_VICTIM_HOST_INFORMATION,
                                        Techniques.
    NETWORK_SERVICE_DISCOVERY }
                },
                EmulationAttackerActionId.TCP_FIN_SCAN_HOST: {
                    "techniques": {Techniques.ACTIVE_SCANNING,
                                        Techniques.
    GATHER_VICTIM_HOST_INFORMATION,
                                        Techniques.
```

```
NETWORK_SERVICE_DISCOVERY }
            },
            EmulationAttackerActionId.TCP_FIN_SCAN_ALL: {
                "techniques": {Techniques.ACTIVE_SCANNING,
                               Techniques.
GATHER_VICTIM_HOST_INFORMATION,
                               Techniques.
NETWORK_SERVICE_DISCOVERY }
            },
            EmulationAttackerActionId.TCP_NULL_SCAN_HOST: {
                "techniques": {Techniques.ACTIVE_SCANNING,
                               Techniques.
GATHER_VICTIM_HOST_INFORMATION,
                               Techniques.
NETWORK_SERVICE_DISCOVERY }
            },
            EmulationAttackerActionId.TCP_NULL_SCAN_ALL: {
                "techniques": {Techniques.ACTIVE_SCANNING,
                               Techniques.
GATHER_VICTIM_HOST_INFORMATION,
                               Techniques.
NETWORK_SERVICE_DISCOVERY }
            },
            EmulationAttackerActionId.TCP_XMAS_TREE_SCAN_HOST
: {
                "techniques": {Techniques.ACTIVE_SCANNING,
                               Techniques.
GATHER_VICTIM_HOST_INFORMATION,
                               Techniques.
NETWORK_SERVICE_DISCOVERY }
            },
            EmulationAttackerActionId.TCP_XMAS_TREE_SCAN_ALL:
 {
                "techniques": {Techniques.ACTIVE_SCANNING,
                               Techniques.
GATHER_VICTIM_HOST_INFORMATION,
                               Techniques.
NETWORK_SERVICE_DISCOVERY }
            },
            EmulationAttackerActionId.OS_DETECTION_SCAN_HOST:
 {
                "techniques": {Techniques.ACTIVE_SCANNING,
                               Techniques.
GATHER_VICTIM_HOST_INFORMATION,
                               Techniques.
NETWORK_SERVICE_DISCOVERY }
            },
```

```
            EmulationAttackerActionId.OS_DETECTION_SCAN_ALL: {
                "techniques": {Techniques.ACTIVE_SCANNING,
                               Techniques.GATHER_VICTIM_HOST_INFORMATION,
                               Techniques.NETWORK_SERVICE_DISCOVERY }
            },


            EmulationAttackerActionId.VULSCAN_HOST: {
                "techniques": { Techniques.GATHER_VICTIM_HOST_INFORMATION,
                                Techniques.SOFTWARE_DISCOVERY },
                "subtechniques": {SubTechniques.SOFTWARE}
            },
            EmulationAttackerActionId.VULSCAN_ALL: {
                "techniques": {Techniques.GATHER_VICTIM_HOST_INFORMATION,
                               Techniques.SOFTWARE_DISCOVERY},
                "subtechniques": {SubTechniques.SOFTWARE}
            },
            EmulationAttackerActionId.NMAP_VULNERS_HOST: {
                "techniques": {Techniques.GATHER_VICTIM_HOST_INFORMATION,
                               Techniques.SOFTWARE_DISCOVERY},
                "subtechniques": {SubTechniques.SOFTWARE}
            },
            EmulationAttackerActionId.NMAP_VULNERS_ALL: {
                "techniques": {Techniques.GATHER_VICTIM_HOST_INFORMATION,
                               Techniques.SOFTWARE_DISCOVERY},
                "subtechniques": {SubTechniques.SOFTWARE}
            },


            EmulationAttackerActionId.TELNET_SAME_USER_PASS_DICTIONARY_HOST: {
                "techniques": {Techniques.BRUTE_FORCE,
                               Techniques.VALID_ACCOUNTS},
                "subtechniques": {SubTechniques.CREDENTIAL_STUFFING,
                                  SubTechniques.
```

```
DEFAULT_ACCOUNTS}
        },
        EmulationAttackerActionId.TELNET_SAME_USER_PASS_DICTIONARY_ALL: {
            "techniques": {Techniques.BRUTE_FORCE,
                           Techniques.VALID_ACCOUNTS},
            "subtechniques": {SubTechniques.CREDENTIAL_STUFFING,
                              SubTechniques.DEFAULT_ACCOUNTS}
        },
        EmulationAttackerActionId.SSH_SAME_USER_PASS_DICTIONARY_HOST: {
            "techniques": {Techniques.BRUTE_FORCE,
                           Techniques.VALID_ACCOUNTS},
            "subtechniques": {SubTechniques.CREDENTIAL_STUFFING,
                              SubTechniques.DEFAULT_ACCOUNTS}
        },
        EmulationAttackerActionId.SSH_SAME_USER_PASS_DICTIONARY_ALL: {
            "techniques": {Techniques.BRUTE_FORCE,
                           Techniques.VALID_ACCOUNTS},
            "subtechniques": {SubTechniques.CREDENTIAL_STUFFING,
                              SubTechniques.DEFAULT_ACCOUNTS}
        },
        EmulationAttackerActionId.FTP_SAME_USER_PASS_DICTIONARY_HOST: {
            "techniques": {Techniques.BRUTE_FORCE,
                           Techniques.VALID_ACCOUNTS},
            "subtechniques": {SubTechniques.CREDENTIAL_STUFFING,
                              SubTechniques.DEFAULT_ACCOUNTS}
        },
        EmulationAttackerActionId.FTP_SAME_USER_PASS_DICTIONARY_ALL: {
            "techniques": {Techniques.BRUTE_FORCE,
                           Techniques.VALID_ACCOUNTS},
            "subtechniques": {SubTechniques.CREDENTIAL_STUFFING,
                              SubTechniques.DEFAULT_ACCOUNTS}
        },
```

```
            EmulationAttackerActionId.
SMTP_SAME_USER_PASS_DICTIONARY_HOST: {
                "techniques": {Techniques.BRUTE_FORCE,
                               Techniques.VALID_ACCOUNTS},
                "subtechniques": {SubTechniques.
CREDENTIAL_STUFFING,
                                  SubTechniques.
DEFAULT_ACCOUNTS}
            },
            EmulationAttackerActionId.
SMTP_SAME_USER_PASS_DICTIONARY_ALL: {
                "techniques": {Techniques.BRUTE_FORCE,
                               Techniques.VALID_ACCOUNTS},
                "subtechniques": {SubTechniques.
CREDENTIAL_STUFFING,
                                  SubTechniques.
DEFAULT_ACCOUNTS}
            },
            #TODO: For database attacks, maybe not Initial
Access: Valid Accounts: Default Accounts?
            EmulationAttackerActionId.
CASSANDRA_SAME_USER_PASS_DICTIONARY_HOST: {
                "techniques": {Techniques.BRUTE_FORCE,
                               Techniques.VALID_ACCOUNTS},
                "subtechniques": {SubTechniques.
CREDENTIAL_STUFFING,
                                  SubTechniques.
DEFAULT_ACCOUNTS}
            },
            EmulationAttackerActionId.
CASSANDRA_SAME_USER_PASS_DICTIONARY_ALL: {
                "techniques": {Techniques.BRUTE_FORCE,
                               Techniques.VALID_ACCOUNTS},
                "subtechniques": {SubTechniques.
CREDENTIAL_STUFFING,
                                  SubTechniques.
DEFAULT_ACCOUNTS}
            },
            EmulationAttackerActionId.
IRC_SAME_USER_PASS_DICTIONARY_HOST: {
                "techniques": {Techniques.BRUTE_FORCE,
                               Techniques.VALID_ACCOUNTS},
                "subtechniques": {SubTechniques.
CREDENTIAL_STUFFING,
                                  SubTechniques.
DEFAULT_ACCOUNTS},
            },
```

```
            EmulationAttackerActionId.
IRC_SAME_USER_PASS_DICTIONARY_ALL: {
                "techniques": {Techniques.BRUTE_FORCE,
                               Techniques.VALID_ACCOUNTS},
                "subtechniques": {SubTechniques.
CREDENTIAL_STUFFING,
                                  SubTechniques.
DEFAULT_ACCOUNTS},
            },
            EmulationAttackerActionId.
MYSQL_SAME_USER_PASS_DICTIONARY_HOST: {
                "techniques": {Techniques.BRUTE_FORCE,
                               Techniques.VALID_ACCOUNTS},
                "subtechniques": {SubTechniques.
CREDENTIAL_STUFFING,
                                  SubTechniques.
DEFAULT_ACCOUNTS}
            },
            EmulationAttackerActionId.
MYSQL_SAME_USER_PASS_DICTIONARY_ALL: {
                "techniques": {Techniques.BRUTE_FORCE,
                               Techniques.VALID_ACCOUNTS},
                "subtechniques": {SubTechniques.
CREDENTIAL_STUFFING,
                                  SubTechniques.
DEFAULT_ACCOUNTS}
            },
            EmulationAttackerActionId.
POSTGRES_SAME_USER_PASS_DICTIONARY_HOST: {
                "techniques": {Techniques.BRUTE_FORCE,
                               Techniques.VALID_ACCOUNTS},
                "subtechniques": {SubTechniques.
CREDENTIAL_STUFFING,
                                  SubTechniques.
DEFAULT_ACCOUNTS}
            },
            EmulationAttackerActionId.
POSTGRES_SAME_USER_PASS_DICTIONARY_ALL: {
                "techniques": {Techniques.BRUTE_FORCE,
                               Techniques.VALID_ACCOUNTS},
                "subtechniques": {SubTechniques.
CREDENTIAL_STUFFING,
                                  SubTechniques.
DEFAULT_ACCOUNTS}
            },
            EmulationAttackerActionId.
MONGO_SAME_USER_PASS_DICTIONARY_HOST: {
```

```
                "techniques": {Techniques.BRUTE_FORCE,
                               Techniques.VALID_ACCOUNTS},
                "subtechniques": {SubTechniques.CREDENTIAL_STUFFING,
                                  SubTechniques.DEFAULT_ACCOUNTS}
            },
            EmulationAttackerActionId.MONGO_SAME_USER_PASS_DICTIONARY_ALL: {
                "techniques": {Techniques.BRUTE_FORCE,
                               Techniques.VALID_ACCOUNTS},
                "subtechniques": {SubTechniques.CREDENTIAL_STUFFING,
                                  SubTechniques.DEFAULT_ACCOUNTS}
            },

            EmulationAttackerActionId.NETWORK_SERVICE_LOGIN: {
                "techniques": {Techniques.VALID_ACCOUNTS,
                               Techniques.REMOTE_SERVICES,
                               Techniques.EXTERNAL_REMOTE_SERVICES}
            },
            EmulationAttackerActionId.FIND_FLAG: {
                "techniques": {Techniques.DATA_FROM_LOCAL_SYSTEM}
            },
            EmulationAttackerActionId.NIKTO_WEB_HOST_SCAN: {
                "techniques": {Techniques.ACTIVE_SCANNING,
                               Techniques.GATHER_VICTIM_HOST_INFORMATION}
            },
            EmulationAttackerActionId.MASSCAN_HOST_SCAN: {
                "techniques": {Techniques.ACTIVE_SCANNING,
                               Techniques.GATHER_VICTIM_HOST_INFORMATION,
                               Techniques.NETWORK_SERVICE_DISCOVERY}
            },
            EmulationAttackerActionId.MASSCAN_ALL_SCAN: {
                "techniques": {Techniques.ACTIVE_SCANNING,
                               Techniques.GATHER_VICTIM_HOST_INFORMATION,
                               Techniques.NETWORK_SERVICE_DISCOVERY}
            },
```

```
            EmulationAttackerActionId.FIREWALK_HOST: {
                "techniques": {Techniques.ACTIVE_SCANNING,
                               Techniques.
GATHER_VICTIM_NETWORK_INFORMATION}
            },
            EmulationAttackerActionId.FIREWALK_ALL: {
                "techniques": {Techniques.ACTIVE_SCANNING,
                               Techniques.
GATHER_VICTIM_NETWORK_INFORMATION}
            },
            EmulationAttackerActionId.HTTP_ENUM_HOST: {
                "techniques": {Techniques.ACTIVE_SCANNING,
                               Techniques.
GATHER_VICTIM_NETWORK_INFORMATION}
            },
            EmulationAttackerActionId.HTTP_ENUM_ALL: {
                "techniques": {Techniques.ACTIVE_SCANNING,
                               Techniques.
GATHER_VICTIM_NETWORK_INFORMATION}
            },
            EmulationAttackerActionId.HTTP_GREP_HOST: {
                "techniques": {Techniques.ACTIVE_SCANNING,
                               Techniques.
GATHER_VICTIM_IDENTITY_INFORMATION}
            },
            EmulationAttackerActionId.HTTP_GREP_ALL: {
                "techniques": {Techniques.ACTIVE_SCANNING,
                               Techniques.
GATHER_VICTIM_IDENTITY_INFORMATION}
            },
            EmulationAttackerActionId.FINGER_HOST: {
                "techniques": {Techniques.ACTIVE_SCANNING,
                               Techniques.
GATHER_VICTIM_HOST_INFORMATION}
            },
            EmulationAttackerActionId.FINGER_ALL: {
                "techniques": {Techniques.ACTIVE_SCANNING,
                               Techniques.
GATHER_VICTIM_HOST_INFORMATION}
            },
            EmulationAttackerActionId.INSTALL_TOOLS: {
                "techniques": {Techniques.
INGRESS_TOOL_TRANSFER}
            },
            EmulationAttackerActionId.SSH_BACKDOOR: {
                "techniques": {Techniques.
COMPROMISE_CLIENT_SOFTWARE_BINARY,
```

```
                                        Techniques.CREATE_ACCOUNT}
            },
            EmulationAttackerActionId.SAMBACRY_EXPLOIT: {
                "techniques": {Techniques.EXPLOIT_PUBLIC_FACING_APPLICATION,
                                        Techniques.REMOTE_SERVICES,
                                        Techniques.EXPLOITATION_OF_REMOTE_SERVICES,
                                        Techniques.NATIVE_API}
            },
            EmulationAttackerActionId.SHELLSHOCK_EXPLOIT: {
                "techniques": {Techniques.EXPLOIT_PUBLIC_FACING_APPLICATION,
                                        Techniques.EXPLOITATION_OF_REMOTE_SERVICES,
                                        Techniques.COMMAND_AND_SCRIPTING_INTERPRETER}
            },
            EmulationAttackerActionId.DVWA_SQL_INJECTION: {
                "techniques": {Techniques.EXPLOIT_PUBLIC_FACING_APPLICATION,
                                        Techniques.EXPLOITATION_FOR_CREDENTIAL_ACCESS,
                                        Techniques.CREDENTIALS_FROM_PASSWORD_STORES}
            },
            EmulationAttackerActionId.CVE_2015_3306_EXPLOIT: {
                "techniques": {Techniques.EXPLOIT_PUBLIC_FACING_APPLICATION,
                                        Techniques.VALID_ACCOUNTS,
                                        Techniques.FALLBACK_CHANNELS,
                                        Techniques.REMOTE_SERVICES},
            },
            EmulationAttackerActionId.CVE_2015_1427_EXPLOIT: {
                "techniques": {Techniques.EXPLOIT_PUBLIC_FACING_APPLICATION,
                                        Techniques.EXPLOITATION_OF_REMOTE_SERVICES,
                                        Techniques.COMMAND_AND_SCRIPTING_INTERPRETER,
                                        Techniques.FALLBACK_CHANNELS,}
            },
            EmulationAttackerActionId.CVE_2016_10033_EXPLOIT: {
                "techniques": {Techniques.
```

```
    EXPLOIT_PUBLIC_FACING_APPLICATION,
                                                Techniques.
    COMMAND_AND_SCRIPTING_INTERPRETER,
                                                Techniques.
    ABUSE_ELEVATION_CONTROL_MECHANISM,
                                                Techniques.VALID_ACCOUNTS,
                                                Techniques.FALLBACK_CHANNELS}
                   },
                   EmulationAttackerActionId.CVE_2010_0426_PRIV_ESC:
     {
                       "techniques": {Techniques.
    ABUSE_ELEVATION_CONTROL_MECHANISM,
                                                Techniques.
    COMMAND_AND_SCRIPTING_INTERPRETER,
                                                Techniques.
    EXPLOITATION_FOR_PRIVILEGE_ESCALATION},
                       "subtechniques": {SubTechniques.UNIX_SHELL}
                   },
                   EmulationAttackerActionId.CVE_2015_5602_PRIV_ESC:
     {
                       "techniques": {Techniques.
    ABUSE_ELEVATION_CONTROL_MECHANISM,
                                                Techniques.
    EXPLOITATION_FOR_PRIVILEGE_ESCALATION},
                       "subtechniques": {SubTechniques.
    SUDO_AND_SUDO_CACHING}
                   },
                   EmulationAttackerActionId.CONTINUE: {
                       None
                   },
                   EmulationAttackerActionId.STOP: {
                       None
                   },

               }

               return mapping.get(id, None)
```

Listing A.3: Technique mapping to network commands.

```
ChildNode = Tuple[Tactics, int]

class AttackGraph():
    """
    Class representing the attack graph
    """


```

```
    def __init__(self):
        """
        Class contructor
        The graph is represented as a list of tuples. Each tuple contains the node name, the children of the node and the node id.
        """
        self.graph = []

    def add_node(self, node_name: Tactics, children: List[ChildNode] = None, node_id: int = None):
        """
        Add a node to the graph
        """
        if node_id is None:
            node_id = len(self.graph) + 1
        if children is None:
            children = []
        self.graph.append((node_name, children, node_id))

    def add_edge(self, parent_node_name: Tactics, parent_node_id: int, child_node_name: Tactics, child_node_id: int):
        """
        Add an edge to the graph by defining the parent node and the children
        """
        for i, (node_name, children, node_id) in enumerate(self.graph):
            if node_name == parent_node_name and node_id == parent_node_id:

                if any(child[0] == child_node_name for child in children):
                    raise RuntimeError("Child node already exists in the parent node")
                else:
                    self.graph[i][1].append((child_node_name, child_node_id))

                break


    def get_node(self, node_name: Tactics, node_id: int):
        """
        Get the node from the graph
        """
```

```
        for node in self.graph:
            if node_name == node[0] and node[2] == node_id:
                return node

    def get_root_node(self):
        """
        Get the root node of the graph
        """
        return self.graph[0]


    def get_children(self, node_name: Tactics, node_id: int):
        """
        Get the children of the node
        """
        for node in self.graph:
            if node_name == node[0] and node[2] == node_id:
                return node[1]
```

Listing A.4: Implementation of the attack graph.

```
def get_attack_profile_sequence(attacker_actions: List[
    EmulationAttackerAction], attack_graph: AttackGraph):
        """
        Returns the attack profile of the actions in a
    sequence
        """

        attack_profiles = []
        for action in attacker_actions:
            attack_profiles.append(AttackProfiler.
    get_attack_profile(action))



        node = attack_graph.get_root_node()
        for profile in attack_profiles:

            techniques_tactics = profile.techniques_tactics
            techniques_to_keep = []
            children = attack_graph.get_children(node[0],
    node[2])
            possible_nodes = []
            for technique in techniques_tactics:
                if node[0].value in techniques_tactics[
    technique]:
                    techniques_to_keep.append(technique)
                    if node not in possible_nodes:
```

```
                    possible_nodes.append(node)

        for child in children:
            for technique in techniques_tactics:
                if child[0].value in techniques_tactics[technique]:

                    techniques_to_keep.append(technique)
                    if attack_graph.get_node(child[0], child[1]) not in possible_nodes:
                        possible_nodes.append(attack_graph.get_node(child[0], child[1]))

        if len(possible_nodes) == 1:
            node = possible_nodes[0]
        if not techniques_to_keep:
            continue
        techniques_to_remove = set(profile.techniques_tactics.keys()) - set(techniques_to_keep)
        for technique in techniques_to_remove:
            try:
                del profile.techniques_tactics[technique]
                del profile.mitigations[technique]
                del profile.data_sources[technique]
                del profile.subtechniques[technique]
            except:
                raise RuntimeError("Error in removing techniques from the attack profile")
```

Listing A.5: Implementation of the attack profiler. Pruning a sequence of network commands using the attack graph.

```
# KLD Backoff smoothing
P = X
Q = X_no_intrusion
CP = len(P)
CQ = len(Q)
SU = list(set(X + X_no_intrusion))
CU = len(SU)
epsilon = 0.0000001
SU_disjoint_P = len(list(set(SU) - set(P)))
SU_disjoint_Q = len(list(set(SU) - set(Q)))

pc = (sum(Y) + epsilon*(SU_disjoint_P) - 1) / CP
qc = (sum(Y_no_intrusion) + epsilon*(
```

```
SU_disjoint_Q) - 1) / CQ
        p_prime = []
        q_prime = []


        for val in SU:
            if val in P:
                p_prime.append((Y[X.index(val)] - pc))
            else:
                p_prime.append(epsilon)
            if val in X_no_intrusion:
                q_prime.append((Y_no_intrusion[X_no_intrusion.index(val)] - qc))
            else:
                q_prime.append(epsilon)

        p_prime_np = np.array(p_prime) / np.sum(p_prime)
        q_prime_np = np.array(q_prime) / np.sum(q_prime)

        KLD_PQ = np.around(np.sum(p_prime_np * np.log(p_prime_np / q_prime_np)), 4)
        KLD_QP = np.around(np.sum(q_prime_np * np.log(q_prime_np / p_prime_np)), 4)
```

Listing A.6: Implementation of the back-off smoothing algorithm to calculate the KLD values

```
    @staticmethod
    def viterbi(hidden_states: List[EmulationAttackerActionId], init_probs: List[float],
                trans_matrix: List[List[float]], emission_matrix: List[List[float]],
                obs: List[int], emissions_list: List[int]) -> List[float]:
        """
        Viterbi algorithm for Hidden Markov Models (HMM).

        :param hidden_states: The hidden states
        :param init_probs: The initial probabilities of the hidden states
        :param trans_matrix: The transition matrix
        :param emission_matrix: The emission matrix
        :param obs: The observation sequence
        :param emissions_list: The list of possible
```

```python
observations

    :return: The most likely sequence of hidden states
    """
    # Convert the emissions list to a numpy array, to use the where function
    emissions_list_typed: np.ndarray[int, Any] = np.array(emissions_list)

    # Check that the sum equals 1
    for i in range(len(emission_matrix)):
        if round(sum(emission_matrix[i]), 10) != 1:
            print(f'Sum of probabilities for state {hidden_states[i]} is not 1')
            print(f'Sum of probabilities: {sum(emission_matrix[i])}')

    # The number of hidden states
    S = len(hidden_states)
    # The number of observations
    T = len(obs)

    # The Viterbi matrix (prob) T x S matrix of zeroes
    prob = np.zeros((T, S))
    # The backpointer matrix (prev)
    prev = np.empty((T, S))
    # Initialization
    for i in range(S):
        # Fetch the index of the observation in the emission_matrix
        index, = np.where(emissions_list_typed == obs[0])
        if index[0].size > 0:
            prob[0][i] = init_probs[i] * emission_matrix[i][index[0]]
        else:
            print(f'Observation {obs[0]} not found in the emission matrix')
            sys.exit(1)

    # Recursion
    for t in range(1, T):
        index, = np.where(emissions_list_typed == obs[t])
        for i in range(S):
            max_prob = -1
            max_state = -1
            for j in range(S):
                new_prob = prob[t - 1][j] * trans_matrix[
```

```
j][i] * emission_matrix[i][index[0]]
                    if new_prob > max_prob:
                        max_prob = new_prob
                        max_state = j
                prob[t][i] = max_prob
                prev[t][i] = max_state

        path = np.zeros(T)
        path[T - 1] = np.argmax(prob[T - 1])
        for t in range(T - 2, -1, -1):
            path[t] = prev[t + 1][int(path[t + 1])]
        # Convert the path to a list
        typed_path: List[float] = path.tolist()

        return typed_path
```

Listing A.7: Implementation of the viterbi algorithm

```
class HMMProfiler:
    """
    The HMMProfiler class is used to profile a sequence of
    observations based on a Hidden Markov Model (HMM).
    """

    def __init__(self, statistics: List[EmulationStatistics],
     model_name: Union[str, None] = None) -> None:
        """
        Class constructor

        :param statistics: The list of EmulationStatistics
    objects
        :param model_name: The name of the model
        """
        self.statistics = statistics
        self.transition_matrix: List[List[float]] = []
        self.emission_matrix: List[List[float]] = []
        self.hidden_states: List[str] = []
        self.emission_matrix_observations: List[int] = []
        self.start_state_probs: List[float] = []
        self.model_name = None
```

Listing A.8: Constructor for the HMM profiler

```
    def create_model(self, transition_matrix: List[List[float
    ]],
                     hidden_states: List[str], metric: str,
                     save_model: bool = False, location: str
    = ".") -> None:
```

```python
        """
        Creates the HMM model based on the given transition matrix, states and metrics.
        If save = True, matrices are saved to given location

        :param transition_matrix: The transition matrix
        :param states: The list of states of the HMM (format: 'A:attack_name' or
        'no_intrusion' based on emulation statistics file)
        :param metrics: The list of metrics to profile
        :param save: Whether to save the matrices to a file
        :param location: The location to save the matrices, if save = True, e.g "./resources",
        default is current directory
        """
        emission_matrix, emission_matrix_observations = self.get_matrices_of_observation(self.statistics,
                                                  metric, hidden_states)
        self.emission_matrix = emission_matrix
        self.emission_matrix_observations = emission_matrix_observations
        self.transition_matrix = transition_matrix
        self.start_state_probs = self.calculate_initial_states(self.transition_matrix)
        self.hidden_states = hidden_states
        if save_model and location:
            np.save(f'{location}/transition_matrix.npy', transition_matrix)
            np.save(f'{location}/hidden_states.npy', hidden_states)
            np.save(f'{location}/start_state_probs.npy', self.start_state_probs)
            np.save(f'{location}/emission_matrix_{metric}.npy', emission_matrix)
            np.save(f'{location}/emission_matrix_observations_{metric}.npy', emission_matrix_observations)
```

Listing A.9: Creates an HMM model, based on statistics from the testbed

```python
def get_matrices_of_observation(self, statistics: List[EmulationStatistics],
                                metric: str, states: List[str]) -> Tuple[List[List[float]], List[int]]:
        """
        Creates the emission matrix for a given metric based on the statistics from the EmulationStatistics objects.
```

```

        :param statistics: The list of EmulationStatistics
    objects
        :param metric: The metric to get the emission matrix
    for

        :return: The emission matrix, the list of
    observations, the list of states
        """
        emission_matrix = []
        attack_observations = {}
        attack_observations_total_counts = {}
        all_keys = set()

        for stats in statistics:
            for condition, metric_distribution in stats.
    conditionals_counts.items():
                action = condition.split('_')
                if action[0] == 'no':
                    action[0] = 'no_intrusion'
                if action[0] not in attack_observations:
                    # We are not intrested in the
    observations from 'intrusion' or 'A:Continue'
                    if action[0] == 'intrusion' or action[0]
    == 'A:Continue':
                        continue
                    else:
                        # Add the observations of the attack
    to the dictionary
                        if metric in metric_distribution:
                            attack_observations[action[0]] =
    metric_distribution[metric]
                        # Sum the total counts of the
    observations
                            attack_observations_total_counts[
    action[0]] = sum(attack_observations[action[0]].values())
                # Aggregate the counts from the metric
    distribution
                else:
                    counts_observation = metric_distribution[
    metric]
                    for element in counts_observation:
                        if element in attack_observations[
    action[0]]:
                            # Aggregate the counts if the
    element is already in the dictionary
                            attack_observations[action[0]][
```

```
element] += counts_observation[element]
                    else:
                        attack_observations[action[0]][element] = counts_observation[element]
                # Sum the total counts of the observations
                attack_observations_total_counts[action[0]] += sum(attack_observations[action[0]].values())

            # Store all possible values for the observation
            if action[0] in attack_observations:
                all_keys.update(attack_observations[action[0]])

    # Normalize the counts
    for attack, _ in attack_observations.items():
        attack_observations_total_counts[attack] = sum(attack_observations[attack].values())
        for key in all_keys:
            int_key = int(key)
            if key in attack_observations[attack]:
                count = attack_observations[attack].pop(key, 0)
                attack_observations[attack][int_key] = count / attack_observations_total_counts[attack]
            else:
                attack_observations[attack][int_key] = 0
        # Sort the dictionary by key
        attack_observations[attack] = dict(sorted(attack_observations[attack].items()))

    # Take any attack as the reference to get the keys
    emission_matrix_observations = []
    emission_matrix_states = []
    # Create the emission matrix
    for state in states:
        if state in attack_observations:
            # Normalize the and then append
            emission_matrix.append(list(attack_observations[state].values()))
            # Get the keys of all observations
            emission_matrix_observations = list(attack_observations[state].keys())
            emission_matrix_states.append(state)
        else:
            # LaPlace smoothing for missing observations
```

```
                num_keys = len(all_keys)
                laplace_probability = 1 / (num_keys + 2)
                laplace_sum = laplace_probability * num_keys
                laplace_probability_adj = laplace_probability / laplace_sum
                emission_matrix.append([laplace_probability_adj] * num_keys)
                emission_matrix_states.append(state)

        # Check if the sum of the probabilities is 1
        for i in range(len(emission_matrix)):
            sum_prob = round(sum(emission_matrix[i]), 10)
            if sum_prob != 1:
                print(f'Sum of probabilities for state {emission_matrix_states[i]} is {sum_prob}')

        return (emission_matrix, emission_matrix_observations)
```

Listing A.10: Function for constructing emission matrix for an observation based on testbed statistics

```
    def profile_sequence(self, sequence: List[int]) -> List[str]:
        """
        Profiles a sequence of observations based on the HMM model.

        :param sequence: The sequence of observations


        :return: The most likely sequence of states
        """

        path = HMMProfiler.viterbi(self.hidden_states, self.start_state_probs,
                                   self.transition_matrix, self.emission_matrix,
                                   sequence, self.emission_matrix_observations)
        profiled_sequence = []
        for i in range(len(path)):
            profiled_sequence.append(self.hidden_states[int(path[i])])

        return profiled_sequence
```

Listing A.11: Profiles a sequence of observations

```python
def convert_states_to_profiles(self, states: List[str]) -> List[Union[AttackProfiler, str]]:
        """
        Converts a list of states to a list of AttackProfiles.

        :param states: The list of states to convert

        :return: The list of EmulationAttackerActionId
        """

        new_states: List[Union[AttackProfiler, str]] = []
        for state in states:
            if state == 'A:Continue':
                action = EmulationAttackerAction(id=EmulationAttackerActionId.CONTINUE, name="Continue", cmds=[],
                                                                  type=None, descr="CONTINUE", ips=[], index=0, action_outcome='')
                p = AttackProfiler.get_attack_profile(action)
                new_states.append(p)
            elif state == 'A:CVE-2015-1427 exploit':
                action = EmulationAttackerAction(
                    id=EmulationAttackerActionId.CVE_2015_1427_EXPLOIT, name="CVE-2015-1427 exploit", cmds=None,
                    type=EmulationAttackerActionType.EXPLOIT,
                    descr="Uses the CVE-2015-1427 vulnerability to "
                    "get remote code execution and then sets up a SSH backdoor"
                    "to upgrade the channel", index=None, ips=[],
                    action_outcome=EmulationAttackerActionOutcome.SHELL_ACCESS)
                p = AttackProfiler.get_attack_profile(action)
                new_states.append(p)
            elif state == 'A:DVWA SQL Injection Exploit':
                action = EmulationAttackerAction(
                    id=EmulationAttackerActionId.DVWA_SQL_INJECTION, name="DVWA SQL Injection Exploit",
                    cmds=None, type=EmulationAttackerActionType.EXPLOIT,
                    descr="Uses the DVWA SQL Injection exploit to extract secret passwords",
                    index=None, ips=[], action_outcome=EmulationAttackerActionOutcome.SHELL_ACCESS)
```

```
                p = AttackProfiler.get_attack_profile(action)
                new_states.append(p)
            elif state == 'A:Install tools':
                action = EmulationAttackerAction(
                    id=EmulationAttackerActionId.INSTALL_TOOLS, name="Install tools", cmds=None,
                    type=EmulationAttackerActionType.POST_EXPLOIT,
                    descr="If taken root on remote machine, installs pentest tools, e.g. nmap",
                    index=None, ips=[], action_outcome=EmulationAttackerActionOutcome.PIVOTING)
                p = AttackProfiler.get_attack_profile(action)
                new_states.append(p)
            elif state == 'A:Network service login':
                action = EmulationAttackerAction(
                    id=EmulationAttackerActionId.NETWORK_SERVICE_LOGIN, name="Network service login",
                    cmds=[], type=EmulationAttackerActionType.POST_EXPLOIT,
                    descr="Uses known credentials to login to network services on a server",
                    index=None, ips=None, action_outcome=EmulationAttackerActionOutcome.LOGIN)
                p = AttackProfiler.get_attack_profile(action)
                new_states.append(p)
            elif state == 'A:Ping Scan':
                action = EmulationAttackerAction(
                    id=EmulationAttackerActionId.PING_SCAN_HOST, name="Ping Scan",
                    cmds=None, type=EmulationAttackerActionType.RECON,
                    descr="A host discovery scan, it is quick because it only checks of hosts "
                    "are up with Ping, without scanning the ports.", ips=None, index=None,
                    action_outcome=EmulationAttackerActionOutcome.INFORMATION_GATHERING, backdoor=False)
                p = AttackProfiler.get_attack_profile(action)
                new_states.append(p)
            elif state == 'A:Sambacry Explolit':
                action = EmulationAttackerAction(
                    id=EmulationAttackerActionId.SAMBACRY_EXPLOIT, name="Sambacry Explolit", cmds=None,
                    type=EmulationAttackerActionType.EXPLOIT,
                    descr="Uses the sambacry shell to get
```

```
remote code execution and then"
                    "sets up a SSH backdoor to upgrade the
channel",
                    index=None, ips=[], action_outcome=
EmulationAttackerActionOutcome.SHELL_ACCESS)
                p = AttackProfiler.get_attack_profile(action)
                new_states.append(p)
            elif state == 'A:ShellShock Explolit':
                action = EmulationAttackerAction(
                    id=EmulationAttackerActionId.
SHELLSHOCK_EXPLOIT, name="ShellShock Explolit",
                    cmds=None, type=
EmulationAttackerActionType.EXPLOIT,
                    descr="Uses the Shellshock exploit and
curl to do remote code execution and create a backdoor",
                    index=None, ips=[], action_outcome=
EmulationAttackerActionOutcome.SHELL_ACCESS)
                p = AttackProfiler.get_attack_profile(action)
                new_states.append(p)
            elif state == 'A:SSH dictionary attack for
username=pw':
                action = EmulationAttackerAction(
                    id=EmulationAttackerActionId.
SSH_SAME_USER_PASS_DICTIONARY_HOST,
                    name="SSH dictionary attack for username=
pw", cmds=None,
                    type=EmulationAttackerActionType.EXPLOIT,
 index=None,
                    descr="A dictionary attack that tries
common passwords and usernames for SSH"
                    "where username=password", ips=None,
action_outcome=EmulationAttackerActionOutcome.SHELL_ACCESS
)
                p = AttackProfiler.get_attack_profile(action)
                new_states.append(p)
            elif state == 'A:FTP dictionary attack for
username=pw':
                action = EmulationAttackerAction(
                    id=EmulationAttackerActionId.
FTP_SAME_USER_PASS_DICTIONARY_HOST,
                    name="FTP dictionary attack for username=
pw", cmds=None, type=EmulationAttackerActionType.EXPLOIT,
                    index=None, descr="A dictionary attack
that tries common passwords and"
                    "usernames for FTP where username=
password", ips=None,
                    action_outcome=
```

```
EmulationAttackerActionOutcome.SHELL_ACCESS)
                p = AttackProfiler.get_attack_profile(action)
                new_states.append(p)
            elif state == 'A:Telnet dictionary attack for
username=pw':
                action = EmulationAttackerAction(
                    id=EmulationAttackerActionId.
TELNET_SAME_USER_PASS_DICTIONARY_HOST,
                    name="Telnet dictionary attack for
username=pw", cmds=None,
                    type=EmulationAttackerActionType.EXPLOIT,
 index=None,
                    descr="A dictionary attack that tries
common passwords and usernames for"
                    "Telnet where username=password", ips=
None,
                    action_outcome=
EmulationAttackerActionOutcome.SHELL_ACCESS)
                p = AttackProfiler.get_attack_profile(action)
                new_states.append(p)
            elif state == 'A:CVE-2010-0426 exploit':
                action = EmulationAttackerAction(
                    id=EmulationAttackerActionId.
CVE_2010_0426_PRIV_ESC,
                    name="CVE-2010-0426 exploit", cmds=None,
type=EmulationAttackerActionType.PRIVILEGE_ESCALATION,
                    descr="Uses the CVE-2010-0426
vulnerability to perform privilege escalation to get root
access",
                    index=None, ips=[], action_outcome=
EmulationAttackerActionOutcome.PRIVILEGE_ESCALATION_ROOT)
                p = AttackProfiler.get_attack_profile(action)
                new_states.append(p)
            elif state == 'A:TCP SYN (Stealth) Scan':
                action = EmulationAttackerAction(
                    id=EmulationAttackerActionId.
TCP_SYN_STEALTH_SCAN_HOST, name="TCP SYN (Stealth) Scan",
                    cmds=None, type=
EmulationAttackerActionType.RECON,
                    descr="A stealthy and fast TCP SYN scan
to detect open TCP ports on the subnet", ips=None,
                    index=None, action_outcome=
EmulationAttackerActionOutcome.INFORMATION_GATHERING,
backdoor=False)
                p = AttackProfiler.get_attack_profile(action)
                new_states.append(p)
            elif state == 'ssh backdoor':
```

```
                action = EmulationAttackerAction(
                    id=EmulationAttackerActionId.SSH_BACKDOOR, name="Install SSH backdoor",
                    cmds=None, type=EmulationAttackerActionType.POST_EXPLOIT,
                    descr="If taken root on remote machine, installs a ssh backdoor useful for"
                    "upgrading telnetor weaker channels", index=None, ips=[],
                    action_outcome=EmulationAttackerActionOutcome.PIVOTING, alt_cmds=None, backdoor=True)
                p = AttackProfiler.get_attack_profile(action)
                new_states.append(p)
            else:
                new_states.append(state)

        return new_states
```

Listing A.12: Convert states to attack profiles

```
def generate_sequence(self, intrusion_length: int, initial_state_index: int,
                          seed: Union[int, None] = None) -> Tuple[List[str], List[int]]:
        """
        Generates a sequence of states and corresponding observations based on the given emission matrix,
        and transition matrix. First, a sequence of observation from 'no intrusion' is generated
        based on the geometric distribution of the initial state. Then, a sequence observations and states are
        generated based on emission matrix and transition matrix. The length of this intrusion
        sequence is given by the intrusion_length parameter.

        :param intrusion_length: The length of the intrusion
        :param initial_state: The index of the initial state
        :param seed: The seed for the random number generator

        return: The sequence of states and observations
        """
        P_obs = self.emission_matrix
        P_states = self.transition_matrix
        states = self.hidden_states
        observations = self.emission_matrix_observations
        if seed:
            np.random.seed(seed)
```

```
    obs_len = len(observations)
    states_len = len(states)

    # Return the geometric distribution of the initial state
    dist = np.random.geometric(p=P_states[initial_state_index][0], size=1000)
    T_i = round(sum(dist) / len(dist))

    state_seq = [states[initial_state_index]] * T_i
    obs_seq = []
    for i in range(T_i):

        o_i = np.random.choice(obs_len, p=P_obs[initial_state_index])
        obs_seq.append(observations[o_i])

    recon_states_sum = np.sum(P_states[initial_state_index][1:])
    recon_states = P_states[initial_state_index][1:] / recon_states_sum

    intrusion_start_state = np.random.choice(states_len - 1, p=recon_states) + 1
    intrusion_start_observation = np.random.choice(obs_len, p=P_obs[intrusion_start_state])
    state_seq.append(states[intrusion_start_state])
    obs_seq.append(observations[intrusion_start_observation])

    s_i = intrusion_start_state
    if intrusion_length == 1:
        return state_seq, obs_seq
    for i in range(intrusion_length):
        # si ~ Ps(si | si-1)
        s_i = np.random.choice(states_len, p=P_states[s_i])
        # oi ~ Po(oi | si)
        o_i = np.random.choice(obs_len, p=P_obs[s_i])
        state_seq.append(states[s_i])
        obs_seq.append(observations[o_i])

    return state_seq, obs_seq
```

Listing A.13: Function to generate sample sequences based on HMM model

TRITA–EECS-EX- 2024:0000

€€€€ For DIVA €€€€

{
"Author1": { "Last name": "Pappila",
"First name": "Bength",
"Local User Id": "u1gqc4ls",
"E-mail": "brpa@kth.se",
"organisation": {"L1": "School of Electrical Engineering and Computer Science",
}
},
"Cycle": "2",
"Course code": "DA231X",
"Credits": "30.0",
"Degree1": {"Educational program": "Master's Programme, Computer Science, 120 credits"
,"programcode": "TCSCM"
,"Degree": "Master's Programme, Computer Science, 120 credits"
,"subjectArea": "Computer Science"
},
"Title": {
"Main title": "Automated Profiling of Cyber Attacks Based on MITRE ATT&CK",
"Language": "eng" },
"Alternative title": {
"Main title": "Automatiserad Profilering av Cyberattacker Baserat på MITRE ATT&CK",
"Language": "swe"
},
"Supervisor1": { "Last name": "Hammar",
"First name": "Kim",
"Local User Id": "u1jhfthv",
"E-mail": "kimham@kth.se",
"organisation": {"L1": "School of Electrical Engineering and Computer Science",
"L2": "Computer Science" }
},
"Examiner1": { "Last name": "Stadler",
"First name": "Prof. Rolf",
"Local User Id": "u158ez9a",
"E-mail": "stadler@kth.se",
"organisation": {"L1": "School of Electrical Engineering and Computer Science",
"L2": "Computer Science" }
},
"National Subject Categories": "10201, 10206",
"Other information": {"Year": "2024", "Number of pages": "xiii,75"},
"Copyrightleft": "copyright",
"Series": { "Title of series": "TRITA–EECS-EX" , "No. in series": "2024:0000" },
"Opponents": { "Name": "A. B. Normal & A. X. E. Normalè"},
"Presentation": { "Date": "2022-03-15 13:00"
,"Language":"eng"
,"Room": "via Zoom https://kth-se.zoom.us/j/ddddddddddd"
,"Address": "Isafjordsgatan 22 (Kistagången 16)"
,"City": "Stockholm" },
"Number of lang instances": "2",
"Abstract[eng]": €€€€
€€€€,
"Keywords[eng]": €€€€
Attack emulation, Attack profiling, Autonomous network security, Cyber security, Hidden Markov Model, Mitre Att&ck, The Cyber Security Learning Environment (CSLE) €€€€,
"Abstract[swe]": €€€€
€€€€,
"Keywords[swe]": €€€€
Attack emulering, Attack profilering, Autonom nätverkssäkerhet, Cybersäkerhet, Dold Markovmodell, Mitre Att&ck, The Cyber Security Learning Environment (CSLE) €€€€,
}

acronyms.tex

```
%%% Local Variables:
%%% mode: latex
%%% TeX-master: t
%%% End:
% The following command is used with glossaries-extra
\setabbreviationstyle[acronym]{long-short}
% The form of the entries in this file is \newacronym{label}{acronym}{phrase}
%                                       or \newacronym[options]{label}{acronym}{phrase
     }
% see "User Manual for glossaries.sty" for the  details about the options, one
     example is shown below
% note the specification of the long form plural in the line below
\newacronym[longplural={Debugging Information Entities}]{DIE}{DIE}{Debugging
     Information Entity}
%
% The following example also uses options
\newacronym[shortplural={OSes}, firstplural={operating systems (OSes)}]{OS}{OS}{
     operating system}

% note the use of a non-breaking dash in long text for the following acronym
\newacronym{IQL}{IQL}{Independent Q^^e2^^80^^91Learning}
\newacronym{TTP}{TTP}{Tactics, techniques and procedure}
\newacronym{CSLE}{CSLE}{The Cyber Security Learning Environment}
\newacronym{KTH}{KTH}{KTH Royal Institute of Technology}
\newacronym{VPN}{VPN}{Virtual Private Network}
\newacronym{CAPEC}{CAPEC}{Common Attack Pattern Enumeration and Classification}
\newacronym{HMM}{HMM}{Hidden Markov Model}
\newacronym{KLD}{KLD}{Kullback-Leibler divergence}
\newacronym{IDS}{IDS}{Intrusion Detection System}
\newacronym{CVE}{CVE}{Common Vulnerabilities and Exposures}

\newacronym{LAN}{LAN}{Local Area Network}
\newacronym{VM}{VM}{virtual machine}
% note the use of a non-breaking dash in the following acronym
\newacronym{WiFi}{Wi^^e2^^80^^91Fi}{Wireless Fidelity}

\newacronym{WLAN}{WLAN}{Wireless Local Area Network}
\newacronym{UN}{UN}{United Nations}
\newacronym{SDG}{SDG}{Sustainable Development Goal}
```