Cyber Attack Storyline Generation Using Hidden Markov Models

by

Muhammed Raed AbuOdeh

(Under the Direction of Prashant Doshi)

Abstract

System calls performed during host-based cyber attacks are often recorded in audit logs. As log files grow in both size and complexity, the objective of detecting attacks, let alone specific phases of attacks, becomes more difficult. Recently published literature focuses on attack detection rather than classification. Using an end-to-end AI system such as Cyberian gives an added ability of identifying phases of a host-based cyber attack from a system call log by analyzing the extracted attack sequence and its respective provenance graph. It is still difficult, however, to successfully classify the attack in its current form. In this research we employ an inference step in Cyberian, a hidden Markov model, to take a sequence of system calls and infer a high-level sequence of abstracted actions, which we refer to as a *storyline*. The storyline helps explain the attack in a more human-readable format. We show that the HMM step of Cyberian significantly improves attack classification.

Index words: Hidden Markov Model, Baum-Welch, Viterbi, Anomalous behavior detection, Attack intent analysis, Intrusion detection systems

Cyber Attack Storyline Generation Using Hidden Markov Models

by

Muhammed Raed AbuOdeh

B.Sc., University of Sharjah, 2015

A Thesis Submitted to the Graduate Faculty of The
University of Georgia in Partial Fulfillment of the
Requirements for the Degree

Master of Science

Athens, Georgia

2020

Cyber Attack Storyline Generation Using Hidden Markov Models

by

Muhammed Raed AbuOdeh

Major Professor: Prashant Doshi

Committee: Kyu Hyung Lee
Le Guan

Electronic Version Approved:

Ron Walcott
Interim Dean of the Graduate School
The University of Georgia
August 2020

Dedication

To Mom and Dad

Acknowledgments

I wish to show my gratitude towards Dr. Prashant Doshi, for his guidance, mentorship, and motivation throughout my research. I would also like to thank my family and friends for their constant support over the past two years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Host-based intrusion detection systems (HIDs) are used in order to ensure that systems are protected from attackers. Systems such as [3, 26] bring anomalous activity to the attention of the defender. When an anomaly is detected on a system, forensic experts would study log records of the system to understand what happened, and attempt to model the behavior of the attacker. They are able to accomplish this by collecting logs at the application level (using web server logs) or at the system level (using system call logs).

Unfortunately, these log files present issues and limitations when used for analysis. The size of these log files is very large and exhaustive, as each generated log file contains millions of system call entries; a typical computer produces more than 10K low-level logged events each minute, with each entry containing information pertaining to that system call. Therefore, manually sifting through gigabytes of log files in order to identify attack phases is costly.

Another problem forensic experts come across is determining the granularity relevant to the attack at hand. While higher-level logging may produce more understandable events across a system, details that may prove to be of great importance are lost and, due to lack of information, coarse data is produced. This may introduce heterogeneity because of the diverse events emitted by different programs in the system. On the other hand, using tools

that provide detailed information may harm rather than help analysis; extracting meaningful abstractions from detailed and homogeneous data would be difficult. Also, if located, an attack could possibly be difficult to interpret due to the fact that it may contain individual steps that are similar in other different attack types.

To this end, we designed Cyberian, a system capable of automatically parsing system logs and extracting attack phases from within those logs. Cyberian is split into three steps: 1) extracting a desired sequence of system calls containing a cyber attack from a log file, 2) processing the sequence to further enable analysis, and 3) classifying the attack phase that the sequence resembles.

The Cyberian pipeline, illustrated in figure 1.1, starts with system call logs, collected through tools such as Linux Audit Log and Sysdig [24]. The collected logs are then passed on to GrAALF [23], a tool that parses log files, enabling forensic analysts to load, store, process, query, and display system events logs. Through GrAALF, we are able to generate provenance graphs from system logs, thus giving us a graphical view of the lineage of objects and their respective causalities. Using a query language similar to SQL, GrAALF is capable of querying logs in real time, isolating sequences of system calls that likely represent an attack path, and then generating an output in JSON format. An additional feature provided is the support for path queries and backtracking to an arbitrary depth from an identified resource, something that is not possible using SQL.

If a sequence is flagged as anomalous after analyzing and extracting it, the sequence is then converted into a JSON file for further analysis. Unfortunately, attempting to classify it proves to be difficult, as there is still noise in the data. When analyzing the current sequence, we see that it represents actions (i.e. starting a shell script) performed by an attacker over time along with background actions emitted by the system itself. In other words, we are given a system that emits observations tainted with noise. We can then proceed to infer the most likely sequence of higher-level actions which have emitted these observed actions.

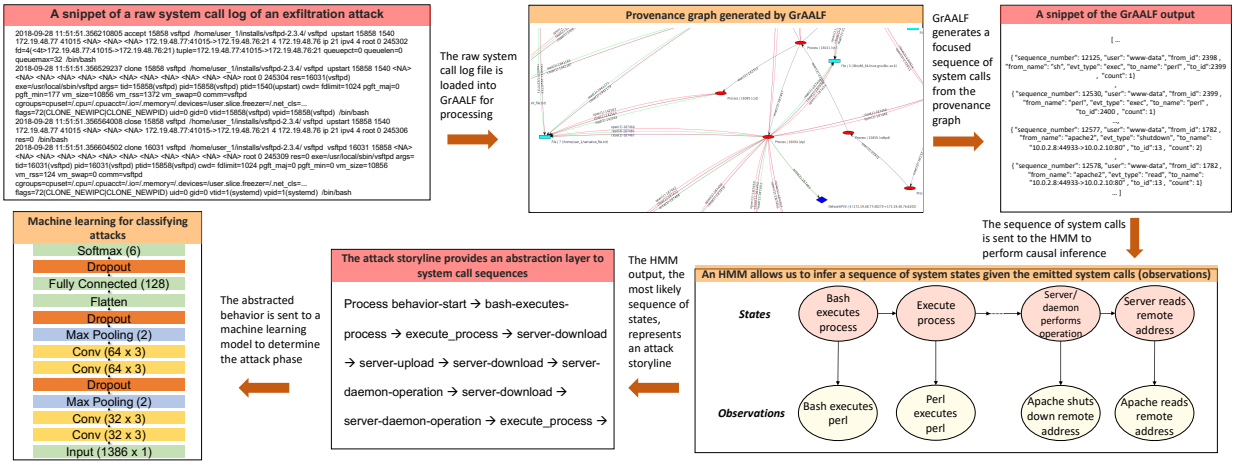

Figure 1.1: The Cyberian pipeline. Starting from the top left, a log file containing a possible attack is loaded into a graphical tool, generating a provenance graph. The result is then passed on to an HMM-based inference step to provide an understandable 'storyline', which facilitates identification of the attack phase using a CNN classifier.

This probabilistic action recognition problem can be modelled using a hidden Markov model (HMM) [19], where the attack trace represents a sequence of observations emitted by a latent, higher-level sequence of states, with the desired result generated by the HMM being the most-likely explanation (MLE) for the observed sequence (the attack trace). And so the aforementioned sequence produced by GrAALF is fed into the HMM, as shown in Fig. 1.1. The HMM generates an abstract and more readable representation of the given sequence. This representation is defined as the most-likely explanation of that sequence, which we will refer to as an *attack storyline*.

Given the high-level sequence generated by the HMM, Cyberian can now identify the attack phase. Each sequence generated by the HMM is associated with an attack phase, and so we can treat this as a multi-class classification problem where each label is a phase of an attack. Therefore, the attack storylines generated by the HMM are sent to the next step of Cyberian's pipeline, the classifier. Each attack storyline fed into the classifier is assigned

a specific attack phase based on the storyline. The chosen classifiers evaluated in this step are support vector machines, convolutional neural networks, and long short-term memory networks.

The audit log files were obtained from several attacks launched on a system, performed by an independent red team. The red team used Metasploit [21], a well-known penetration testing tool, in order to simulate a variety of attacks on a server. This yielded a total of 114 attack phases, which included asset discovery, privilege escalation, and data exfiltration, among others. We will use these attacks to evaluate 1) the HMM in correctly inferring the high-level actions of each attack, and 2) the classifiers in correctly identifying the attack phases.

Experimental results show that using an HMM resulted in a significantly higher accuracy when identifying attack phases. Consequently, we prove that through only system call logs, Cyberian can identify host-based attack phases with high accuracy.

As a final note, Cyberian is not an intrusion detection system (IDS); rather, it functions as a complement to them. An IDS monitors networks and systems for potential anomalies, and reports the behavior to a human analyst. The analyst must then study the questionable activity manually and act accordingly. The role of Cyberian is to take the place of the human analysis step by automating it.

When a log is flagged as anomalous, it is fed into Cyberian's pipeline, where the sequence of system calls is extracted, processed, and classified as one of the phases.

1.1 Related Work

In the field of security, particularly in attack analysis, probabilistic models such as HMMs have been used. Their application is found in intrusion detection research such as [16]. An HMM is used in Garcia et al. [5] along with k-means for *detecting* malicious activity. In

Wang et al. [27], an HMM is used to determine whether or not a sequence exceeds a certain threshold, thus classifying it as an attack, while Radhakrishna et al. [20] identify intrusions using temporal pattern mining.

The limitation of intrusion detection, however, is that it does not describe the nature of the attack; it only determines whether or not the system has been compromised. Analysis of attack characteristics or attack phase identification are not key features in intrusion detection. Looking at a specific application of intrusion detection, anomaly-based intrusion detection, we find that systems such as DeepLog [4] are used. Deeplog tackles this problem using deep learning to detect anomalous behavior through the analysis of low-level log data.

Like intrusion detection, anomaly detection is also limited. Anomaly detection only directs the specialist to the location of suspicious activity, while analysis and understanding of data, as well as incident response, is left to the specialist. Unfortunately, due to the limited time and considerably large amount of data, human analysts can only do so much, indicating the need for AI systems such as Cyberian.

Cyber attack classification is an area that has been explored before. Lippmann et al. [15] used network log data to analyze the performance of different models in detecting intrusions. The log data was extensive, containing both benign and malicious activity, and included labels corresponding to simple goal-based attack classes. The results highlighted the limitations facing rule-based detection systems. Another paper, Bolzoni et al. [2], used machine learning in anomaly detection to classify attack types based on n-gram analysis of attack payloads. Works such as these do not dissect attacks into individual phases but rather focus on classifying the attack as a whole. Also, there is no emphasis placed on understanding the similarities between different attacks, as many attack sequences share subsequences.

Additionally, a system named HOLMES [17] detects anomalies as phases of advanced and persistent threats (APTs). The system needs a sizeable amount of benign log data to train on in order to reduce false positives in the testing data. HOLMES uses a rule-based

system rather than a machine learning system to classify attack phases, and so the system does not learn from labeled phases to identify different patterns in the data.

1.2 Contributions

The main goal of Cyberian is to classify attack phases rather than only detecting them. In order to do so successfully, we are in need of a model such as the HMM. Therefore, this research is centered around the use of an HMM while still describing it within the context of Cyberian to demonstrate the HMM's significance. The contributions of this research are as follows:

1. Cleaning GrAALF sequences, thus removing redundant information, making the system call sequences understandable

2. Creating human-readable sequences known as attack storylines by annotating system calls in the sequences with states that best depict the behavior of the system over time. We will then be able to infer the MLE of a given sequence of system calls.

3. Compare the results of the classification of system call sequences after the use of the HMM with the results when the HMM is removed from the Cyberian pipeline.

The results we obtained from this research showed that the use of an HMM significantly improved attack phase classification.

1.3 Document structure

The remainder of this thesis is structured as follows. Chapter 2 discusses relevant background information in cyber forensics, as well as the HMM with two well-known algorithms: the Baum-Welch algorithm, and the Viterbi algorithm. In chapter 3, the Cyberian pipeline

is explained in more detail. The experiments conducted with their results are discussed in chapter 4. We conclude the thesis with chapter 5, including additional discussion on potential future work.

Chapter 2

Background

This chapter explains the field in which this research has been conducted, as well as the model specifically used in attack storyline generation, the hidden Markov model. In the first section of the chapter, cyber security, we explore topics and techniques pertaining to forensic analysis and the use of provenance graphs in security. Next, the hidden Markov model is described, as well as two famous algorithms used in training and inference, the Baum-Welch algorithm and the Viterbi algorithm, respectively.

2.1 Cyber security

Digital forensic analysis and provenance graphs

Digital forensic analysis, usually used in a postmortem evaluation, is the exploration of a digital system to extract information that can help highlight the cause of a security incident. Accurate forensic analysis requires data collection over long periods of time, generating considerably large amounts of data. Techniques must then be developed in order to handle this issue. In [8], for example, the authors were able to reduce the amount of data produced by forensic analysis without negatively impacting accuracy by using efficient reduction algo-

rithms.

Another approach taken to tackle large log files is provenance graphs. Provenance is used to explain the origin of something, or is a form of structured metadata that records the activities involved in data production [18]. In [13], provenance is used in forensic analysis to better analyze large log files, while [6] claims that 'provenance is the ideal data to use for such a task [system execution] and that provenance graph-based analysis is the ultimate means towards achieving complete security coverage'.

In host-based systems, provenance records system calls, explaining (for each system call) what called them and what action was performed on them, thus creating a chain or sequence of system calls. The more the graph of system calls grows, the more we gain a detailed understanding of the changes that happened in the system over time.

2.2 The hidden Markov model

The hidden Markov model (HMM) is a dynamic Bayesian network (DBN), a temporal model used to describe the time-varying nature of processes where part of the system is observed, while the other part is not observable yet is inferred through the observed part of the system. It is used in applications such as communication, speech recognition, and bioinformatics. The HMM is composed of the following:

1. A set of N states $S = \{S_1, S_2, ..., S_N\}$
2. A set of M observations $\{v_1, v_2, ..., v_M\}$
3. state transition probability matrix $A = \{a_{ij}\}$, where $a_{ij} = P[q_{t+1} = S_j | q_t = S_i]$, $1 \leq i, j \leq N$
4. emission probability matrix $B = \{b_j(k)\}$, where $b_j(k_t) = P[v_k \, at \, t | q_t = S_j]$, $1 \leq j \leq N$ and $1 \leq k \leq M$

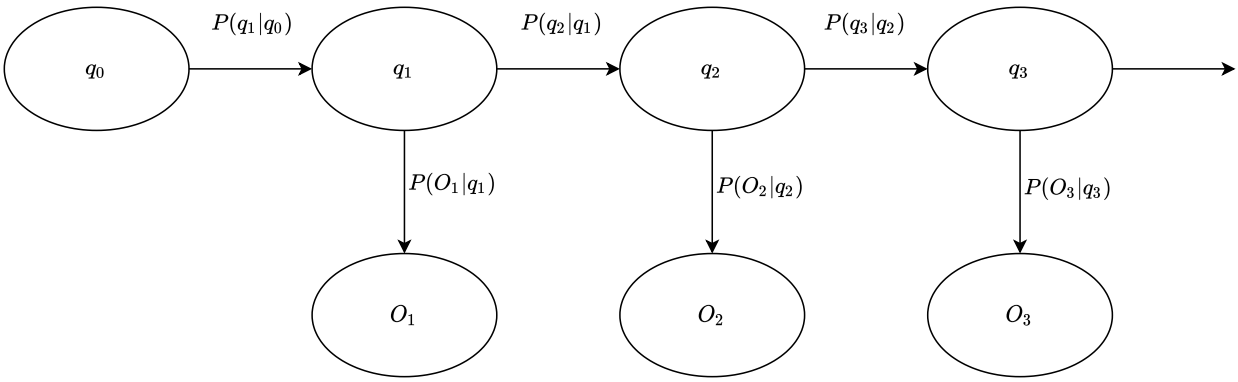

Figure 2.1: Structure of an HMM. The chain of q's represents the sequence of states that are inferred by the emitted observations, represented by the O's at each time step.

5. initial probability distribution $\pi = \{\pi_i\}$, where $\pi_i = P[q_1 = S_i]$, $1 \leq i \leq N$

to yield a model $\lambda = (A, B, \pi)$. The structure of the model is illustrated in figure 2.1. One of the qualities possessed by the HMM is the Markov property: a state at a time t depends only on the previous state at time t-1.

There are three types of problems which the HMM tackles: the decoding problem, the training problem, and the evaluation problem.

The decoding problem

Given a model λ and a sequence of observations, what is the most-likely sequence of states that emitted this sequence?

This problem helps us obtain the most-likely explanation (MLE) of a given sequence of observations. The algorithm used is the Viterbi algorithm (see section 2.4). Through this we are able to obtain the most-likely sequence of states that emitted the observation sequence, as well as the probability of that sequence of states.

The training problem

Given a model and a dataset (of sequences of states with their respective sequences of observations), what is the model that best fits this data?

For unsupervised learning, Baum-Welch is most commonly used, while Viterbi training is used for supervised learning. The algorithm used to train our model is Baum-Welch. Its details will be explained later (see section 2.3).

The evaluation problem

Given a model λ and a sequence of observations, what is the probability that the model emitted this sequence?

Using techniques such as the forward algorithm or the backward algorithm, we are able to get the probability that a model emitted a given sequence of observations.

For this project, only the decoding and training problems are considered.

2.3 The Baum-Welch algorithm

The Baum-Welch algorithm [1] is a form of the expectation maximization (EM) algorithm used to estimate the missing data and update the model. Baum-Welch uses the forward and backward algorithms in order to give the best-fitted model. The results after using the Baum-Welch algorithm are better than those with the EM algorithm.

The steps of the Baum-Welch algorithm are as follows:

1. Initialize parameters
2. Run the forward algorithm
3. Run the backward algorithm

4. calculate the new log-likelihood $P(x|\theta)$

5. Repeat steps 2-4 until convergence

Given a sequence of observations $O = O_1, O_2, ..., O_3$, π can either be initialized randomly or based on prior knowledge by using techniques such as the count-based approach. The forward and backward algorithms are both used to update π, a_{ij}, and $b_i(v_k)$.

The forward step

The forward step is used to update the forward variable $\alpha_t(i)$, defined as

$$\alpha_t(i) = P(O_1, O_2, ..., O_t, q_t = S_i|\lambda) \tag{2.1}$$

$\alpha_t(i)$ can be solved inductively as follows:

$$\text{Initialization:} \quad \alpha_1(i) = \pi_i b_i(O_1), \quad \text{where} \quad 1 \leq i \leq N \tag{2.2}$$

$$\text{Induction:} \quad \alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}), \quad \text{where} \quad 1 \leq t \leq T-1, \text{and} \quad 1 \leq j \leq N \tag{2.3}$$

$$\text{Termination:} \quad P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{2.4}$$

Where $P(O|\lambda)$ is the sum of the $\alpha_T(i)$'s.

The backward step

The backward step is used to calculate the backward variable $\beta_t(i)$, defined as

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, ..., O_T|q_t = S_i, \lambda) \tag{2.5}$$

$\beta_t(i)$ can be solved inductively as follows:

$$\text{Initialization:} \quad \beta_i(T) = 1, \quad \text{where} \quad 1 \leq i \leq N \tag{2.6}$$

$$\text{Induction:} \quad \beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j), \quad \text{where t = T-1, T-2, ..., 1, and} \quad 1 \leq i \leq N \tag{2.7}$$

Calculating the new log-likelihood

$\alpha_t(i)$ and $\beta_t(i)$ are then used to solve for $\gamma_i(t)$, which is the probability of being in state S_i at time t given the observation sequence O and the model λ, and $\xi_t(i,j)$, which is the probability of being in state S_i at time t, and state S_j and time t+1, given the observation sequence O and the model λ. Given the forward and backward variables, we can write $\gamma_i(t)$ and $\xi_{ij}(t)$ in the form

$$\gamma_i(t) = P(q_t = S_i | O, \lambda) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{2.8}$$

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j | O, \lambda) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)} = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)} \tag{2.9}$$

With both $\gamma_i(t)$ and $\xi_t(i,j)$, we can now update the values π_i, a_{ij}, and $b_j(k)$. Summing over $\gamma_i(t)$ gives the expected number of transitions from S_i, while summing over $\xi_t(i,j)$ gives the expected number of transitions from S_i to S_j. The final equations for re-estimation are

$$\pi_i^* = \gamma_1(i) \tag{2.10}$$

$$a_{ij}^* = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{2.11}$$

$$b_j^*(k) = \frac{\sum_{t=1 \text{ s.t. } O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{2.12}$$

Pseudocounts

The size of the dataset used when training the model may be problematic if not handled correctly; a small dataset may lead to over-fitting. To address this issue, pseudocounts are added in order to avoid 0 probabilities in our matrices. Larger pseudocounts are used when there is a strong prior belief, while small pseudocounts are used when the only goal is to avoid 0 probabilities.

2.4 Most-likely Explanation and the Viterbi algorithm

The most-likely explanation (MLE) is the state sequence with the highest probability that could have emitted a given observation sequence. That is, the path sequence that maximizes $P(Q, O|\lambda)$. This can be calculated using the Viterbi algorithm [25]. We will then proceed to solve

$$\delta_t(i) = \max_{q_1,q_2,\cdots q_{t-1}} P(q_1, q_2, ..., q_t = i, O_1, O_2, ..., O_t|\lambda) \tag{2.13}$$

using induction. The initialization step is

$$\delta_1(i) = \pi b_i(O_1) \tag{2.14}$$

$$\psi_1(i) = 0 \tag{2.15}$$

where δ is the starting state i, observing O_1, and ψ is the previous state. The induction step is

$$\delta_t(j) = \max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}] b_j(O_t) \tag{2.16}$$

$$\psi_t(j) = \arg\max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}] \tag{2.17}$$

$$2 \le t \le T, \quad 1 \le j \le N$$

Finally, the termination step is

$$P^* = \max_{1 \le i \le N} [\delta_T(i)] \tag{2.18}$$

$$q_T^* = \arg\max_{1 \le i \le N} [\delta_T(i)], \quad q_t^* = \psi_{t+1}(q*_{t+1}) \tag{2.19}$$

Chapter 3

Methodology

This chapter gives a detailed explanation of the different parts of the Cyberian pipeline. First, we describe how GrAALF is used to generate provenance graphs. Next, the pre-processing of data and the use of an HMM for inference are discussed. Finally, we explain how attack phases are identified using different classifiers.

3.1 Provenance of System Calls using GrAALF

GrAALF is a tool that gives the ability to visualize system logs in the form of provenance graphs. It also allows users to traverse logs through functionality such as querying. With GrAALF, we are able to extract attack traces found within log files, generating both a provenance graph and a JSON file containing the sequence of system calls representing the attack. The generated sequence represents an attack phase, such as asset discovery, data exfiltration, or privilege escalation.

GrAALF can be used to generate provenance graphs both manually and automatically. One can generate graphs manually by entering different queries until a sequence with its respective graph is reached. In order to do so automatically, a set of query templates must

be defined. Query templates are similar to predefined rules in a traditional rule-based system.

To obtain these queries, the user must first specify what system they want monitored, after which queries are derived from templates and are deployed by GrAALF. Here we will provide three different examples as to how this operation works.

Monitoring a file

If we want to monitor a changes made to a sensitive file, the following query would be used by GrAALF:

back select write from file where name is X

where 'X' represents the file name of interest that the host user provides.

Monitoring IP addresses

If we want to monitor whether or not IP addresses outside the local network have be accessed, we would use

back select * from soc where not name has 172.16.

where a range of local IP is '172.16.*.*'.

Monitoring a process

Process-based monitoring is possible with query templates in GrAALF. Suppose we want to monitor 'nc' and 'scp' processes, as they are both used in backdoors and exfiltration. Two queries would be used: the first query would be:

back select * from * where name is nc or name is scp

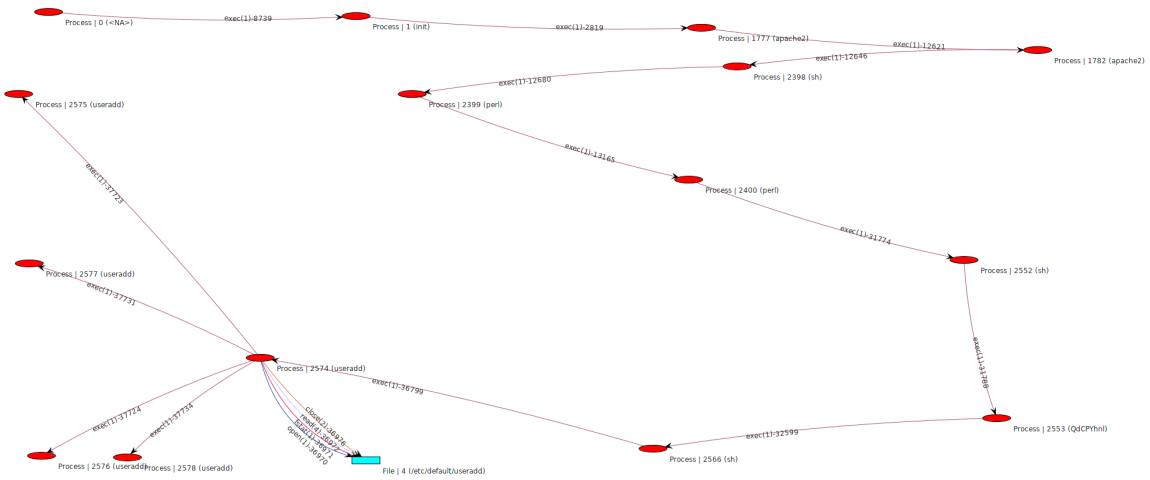

Figure 3.1: A provenance graph of a persistence attack

which allows us to monitor detailed behavior of the processes. The second query is

forward select * form * where name is nc or name is scp

which allows us to identify their remote hosts.

Figures 3.1 and 3.2 are examples of provenance graphs generated by GrAALF.

3.2 Attack Storylines using HMM

Upon examining system call logs generated by GrAALF, we found it difficult to extract useful information due to their extensiveness. For example, some of the larger files representing attack phases contain 8,722 and 10,154 records occupying up to 2.2MB, which makes manually parsing the sequences both tedious and time-consuming. Figure 3.3 is a snippet of a JSON file containing a few system calls of an attack phase. Additionally, each system call in an attack phase contains information that may potentially be redundant when classifying

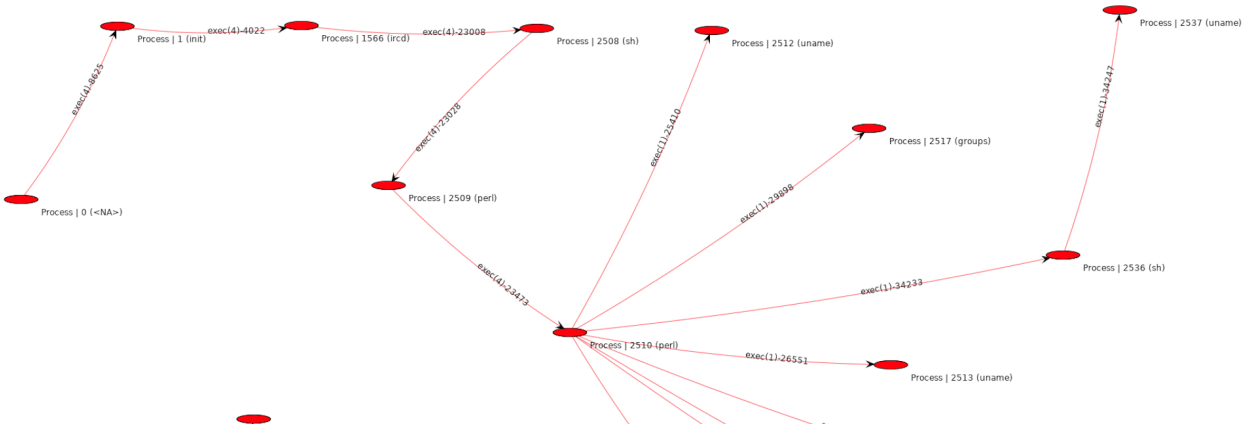

Figure 3.2: Part of a provenance graph of a discovery attack

phases.

In order to address this issue, we require a mathematical model that further abstracts these sequences, thus reducing the amount of detail present while retaining key features. The use of probabilistic graphical models (PGMs) such as hidden Markov models [19] and conditional random fields [12] can aid in abstracting system calls in the sequences, inferring actions for each system call. Each sequence is then tagged with a label based on its type (i.e. persistence, discovery, etc..). While HMMs and linear-chain CRFs [12] share many similarities, Cyberian utilizes HMMs as these are simpler to learn from data and do not require feature functions.

Cyberian's abstraction step is split into three parts: data cleaning, creating and training the HMM, and finally, inferring the most-likely explanation. The most-likely explanation captures the most important features present in the system calls, therefore giving a simplified and human-readable sequence of abstract actions.

```
{ "sequence_number": 13675, "user": "root", "from_id": 2397 , "from_name": "sshd",
        "evt_type": "exec", "to_name": "sshd" , "to_id":2416 ,  "count": 1}

,

{ "sequence_number": 14558, "user": "boba_fett", "from_id": 2416 , "from_name": "sshd",
         "evt_type": "exec", "to_name": "-bash" , "to_id":2420 ,  "count": 1}

,

{ "sequence_number": 83057, "user": "boba_fett", "from_id": 2420 , "from_name": "-bash",
        "evt_type": "exec", "to_name": "X4IrBtlf" , "to_id":2681 ,  "count": 1}

,

{ "sequence_number": 83799, "user": "boba_fett", "from_id": 2681 , "from_name": "X4IrBtlf",
            "evt_type": "exec", "to_name": "sh" , "to_id":2694 ,  "count": 1}

,

    { "sequence_number": 88190, "user": "root", "from_id": 2694 , "from_name": "sh",
        "evt_type": "exec", "to_name": "useradd" , "to_id":2702 ,  "count": 1}
```


Figure 3.3: Code snippet of an attack generated by GrAALF

Data Cleaning

On receiving system call sequences from GrAALF, the individual system calls, such as those shown in the first column in Table 3.3, are first cleaned in order to be handled by the HMM, as they do not aid in understanding the actual action. In particular, Linux audit logging includes details mentioned in table 3.1 for each system call. This information can be split into meta-information, parent process information, child process information, and parent-child event type. Some of the fields such as meta-information and IDs of processes do not directly explain the action that was performed while *from_name*, *evt_type*, and *to_name* contain valuable information for the HMM. As such, the latter fields were kept to be used with the HMM (see the second column of Table 3.3 for illustration).

Additionally, we use straightforward rules to reduce our observation space. For example,

Sequence number	Order in which the process was called in the sequence
User	User who called the process
From ID	ID of the parent process
From name	Name of the parent process
Event type	Action performed by parent process on the child process
To name	ID of the child process
To ID	Name of the child process
Count	Number of occurrences of this particular system call

Table 3.1: Details of each key in a system call

shell processes such as *sh*, *bash*, or *zsh*, are merged into *bash* as it is the most popular shell. Sockets were changed to 'some_socket', and files were changed to 'some_file'. Since our focus is on sequences and not individual processes, these changes do not harm the performance of our algorithms. In some sequences, we observe processes that create temporary child processes with randomly generated process names; we rename them to *temp_process*. Table 3.3 shows a real audit log and processed logs.

Finally, we filter out meaningless and redundant log events. For example, observations containing erroneous information (i.e., $\langle$NA$\rangle$) due to audit system fails are removed, and "NULL" and "pipe" substrings are excluded as well.

To ensure readability of data, observations are written in the triple format *from_name* $\rightarrow$ *evt_type* $\rightarrow$ *to_name*, where the *from_name* in the triple is the parent process, the *evt_type* is what the parent process performs, and *to_name* is the child process on which the event is performed. For example, *bash* $\rightarrow$ *exec* $\rightarrow$ *nc* means that bash executes the process netcat. This is the third and final step in converting system call sequences into observation sequences. The third column in Table 3.3 shows the observation for each system call, respectively.

Inferring Attack Storylines

We utilize a standard HMM for the inference whose structure is shown in Fig. 2.1. Observations to the HMM are the system calls in an attack phase sequence, cleaned and in triple

format as described in the previous section.

Upon analyzing different attack sequences, we noticed that they contained similar or almost exact subsequences. For example, we found that a sequence of system calls involving *sh* executing a temporary process which, in turn, executes another shell was present in attack phases such as reconnaissance and persistence. This subsequence represents a series of actions that occur when an attacker launches a shell that launches a process, which then starts another shell. Therefore, a number of abstracted actions can be used in place of these system calls to explain attacks at a higher level without loss of information. These abstracted actions form the states of the HMM, and a categorized listing of these states is given in Table 3.2.

After receiving the sequence of system calls from GrAALF, cleaning it, and passing it on to the HMM, the sequence of inferred states generated by the HMM is what we define as an *attack storyline*.

Before inferring attack storylines using the HMM, we must first train the model with data. In order to do so, we must learn the transition and emission probability tables of the HMM from the sequences of system calls annotated with abstracted actions mentioned previously, using a learning algorithm such as Baum-Welch [1].

To train the model, we first associate a state to each system call in the sequence; the state assigned is said to have emitted the system call. For example, the state Upload data emits the system call *nmbd* $\rightarrow$ *sendto* $\rightarrow$ *some_socket* and state Operate port emits the system call triple *proftpd* $\rightarrow$ *close* $\rightarrow$ *some_socket*. Table 3.3 gives additional and detailed examples of this mapping.

A trained HMM, when given an observation sequence, will infer the most likely explanation, known as the attack storyline, as shown in 1.1. The attack storylines are then passed to the next step in Cyberian's pipeline for identification.

Start states	Bash_execute_process Execute_process (by non bash process) Init_server_daemon
System operation states	System_operation Generic_process_read_write Generic_process_operation Netapp_operation Bash_operation Server_daemon_operation File_operation
Information states	System_information Library_state
Network states	Process_upload_download Netapp_upload_download Daemon_upload_download Server_download Server_upload

Table 3.2: A categorized listing of the states of the HMM.

3.3 Identifying attack phases

An attack storyline inferred by the HMM explains the steps taken in part of a particular attack, and can then be labeled by a specific attack phase. Since storylines are of a time series nature and can be associated to an attack phase, we may then proceed to solve this identification problem using multi-class classification. The following section will present the different machine learning classification models that will be used to identify attack phases using the previously generated storylines as an input.

Classification Models

Similar to the sequences of system calls, the storylines within a certain phase contain common repeated subsequences. Since 1D convolution is a common method in time-series data analysis, our primary model is a 1-dimensional convolutional neural network (CNN). It has also been used in works related to text classification [14, 28].

In addition to the CNN, a long short-term memory [7] network and a simple linear support vector machine (SVM) were also explored. LSTM-based networks are often used when dealing with data that is sequential and containing long-term temporal dependencies, while SVMs have been used for text classification previously [9].

3.4 Summary of methodology

In summary, the complete Cyberian pipeline is described as follows:

1. The tool Sysdig is used on the victim server to log attacks
2. Using monitoring targets (i.e. sensitive files), GrAALF's templated queries were instantiated and executed on the Sysdig log to generate a provenance graph along with the sequence of system calls of the attack
3. The sequence of system calls undergoes a pre-processing step as described in Section 3.2, and is then analyzed using the HMM
4. The performance of the HMM is then evaluated based on storyline generation accuracy, as we shall see in section 4.2, and the performance of the classifier in correctly labeling the attack phases in section 4.3.

System calls generated by GrAALF	from_name, evt_type, and to_name extracted from system calls	Processed system calls, in the form *from_name* → *evt_type* → *to_name*	Abstracted actions (HMM states)
{ "sequence_number": 12125, "user": "www-data", "from_id": 2398 , "from_name": "sh", "evt_type": "exec", "to_name": "perl" , "to_id":2399 , "count": 1}	*from_name*:sh *evt_type*: exec *to_name*: perl	bash → exec → perl	Bash_execute_process
{ "sequence_number": 12530, "user": "www-data", "from_id": 2399 , "from_name": "perl", "evt_type": "exec", "to_name": "perl" , "to_id":2400 , "count": 1}	*from_name*: perl *evt_type*: exec *to_name*: perl	perl → exec → perl	Execute_process
{ "sequence_number": 12577, "user": "www-data", "from_id": 1782 , "from_name": "apache2", "evt_type": "shutdown", "to_name": "10.0.2.8:44933 →10.0.2.10:80" , "to_id":13 , "count": 2}	*from_name*: apache2 *evt_type*: exec *to_name*: "10.0.2.8:44933 → 10.0.2.10:80"	apache2 → close → remote_address	Server-daemon-operation
{ "sequence_number": 12578, "user": "www-data", "from_id": 1782 , "from_name": "apache2", "evt_type": "read", "to_name": "10.0.2.8:44933→10.0.2.10:80" , "to_id":13 , "count": 1}	*from_name*: vUDpWjz9 *evt_type*: read *to_name*: 10.0.2.8:44933→ 10.0.2.10:80	apache2 → read → remote_address	Server-download

Table 3.3: Raw system calls extracted from log files and processed. The first column shows the system calls recorded by auditing software and output by GrAALF. The second column gives the information extracted from the system calls, and the third shows the format of the sequences as passed on to the HMM. The HMM models the sequences as being emitted from the corresponding states shown in the fourth column.

Chapter 4

Experiments

In this chapter, we first discuss the process with which we obtained the data set, the attack phases we chose for this research project, and the system call sequence generation using GrAALF. We then show experimental results obtained in the HMM inference step, as well as the results in the the attack phase classification step.

4.1 Data collection and provenance generation using GrAALF

In order to simulate a real-world setting, an independent red team composed of cyber security researchers proceeded to launch several attacks on a linux server using tools such as Metasploit [21]. The gathered number of attacks was 139 phases either categorized into *six* well-known attack phases or the attack is considered a benign sequence. The attack phase types are:

1. **system reconnaissance**: attacker gathers information about the system, including operating system version and user account information;

2. **persistence**: attacker implements measures to maintain access even after a system restart;
3. **privilege escalation**: attacker attempts to gain root or higher access on the victim machine;
4. **asset discovery**: attacker searches for assets such as sensitive files on a system;
5. **data exfiltration**: targeted attack, attacker transfers data out of the host;
6. **network discovery**: attacker explores different connections made to/from the victim machine.

The distribution of these types in the 139 attack phases is shown in Table 4.1. As discussed in 3.1, these log files were passed on to GrAALF in order to obtain sequences of system calls that can be analyzed in later stages of the Cyberian pipeline.

Attack phase	**Count**
system reconnaissance	39
persistence	12
privilege escalation	14
asset discovery	16
data exfiltration	14
network discovery	19
others/benign	25

Table 4.1: The distribution of various phases in our attack data set.

4.2 HMM Inference of Attack Phase

Upon receiving the 139 sequences from GrAALF, they are then cleaned, giving 1176 unique observations. The observations were assigned to one of the HMM's 17 states, as defined in Table 3.2. The package used to implement the HMM was Pomegranate [22], coded in Python. The HMM's performance was then evaluated using 5-fold cross validation on system call sequences, with their respective state sequences. The HMM is trained using four folds, starting with a count-based initialization [11]. The training involves learning the transition

and emission probabilities using the Baum-Welch algorithm. The inference of the storyline is evaluated using the fifth fold.

The following examples depict sequences of observations with their corresponding attack storylines. We add annotations to the state sequence elements in order to give more meaning, such that when an analyst were to go over a sequence they could understand what the attacker was doing on the system. We chose to add the child process as an annotation when bash executes a process other than temp_process, the parent process when the state is *system-operation*, and bash and su whenever they're executed.

Discovery phase

init → exec → ircd,	init-server-daemon,
init → exec → ircd,	init-server-daemon,
daemon → exec → bash,	server-daemon-operation,
bash → exec → perl,	bash-execute-process *perl*,
perl → exec → perl,	execute-process,
perl → exec → groups,	system-information,
perl → exec → whoami,	system-information,
perl → exec → bash,	execute-process *bash*,
bash → exec → temp_process,	bash-execute-process,
temp_process → exec → bash,	execute-process *bash*,
bash → exec → whoami,	bash-execute-process *whoami*,
bash → exec → groups,	bash-execute-process *groups*,
bash → exec→ getent	bash-execute-process *getent*

Persistence phase

init → exec → apache2,	init-server-daemon,
apache2 → exec → apache2,	init-server-daemon,
apache2 → exec → bash,	server-daemon-operation,
bash → exec → bash,	bash-execute-process *bash*,
bash → exec → bash,	bash-execute-process *bash*,
bash → exec → temp_process,	bash-execute-process,
temp_process → exec → bash,	execute-process *bash*,
bash → exec → useradd,	bash-execute-process *useradd*,
useradd → open → /etc/default/useradd,	system-operation *by useradd*,
useradd → fstat → /etc/default/useradd,	system-operation *by useradd*,
useradd → read → /etc/default/useradd,	system-operation *by useradd*,
useradd → close → /etc/default/useradd,	system-operation *by useradd*,
useradd → exec → useradd,	system-operation *by useradd*,
useradd → exec → useradd,	system-operation *by useradd*,
useradd → exec → useradd,	system-operation *by useradd*,
useradd → exec → useradd	system-operation *by useradd*

Remote network phase

init → exec → avahi-daemon,	init-server-daemon,
init → exec → nmbd,	init-server-daemon,
bash → exec → ruby2.3,	bash-execute-process *ruby2.3*,
daemon → sendto → remote_address_1,	daemon-upload-download,
daemon → sendmsg → remote_address_1,	daemon-upload-download,
daemon → sendto → remote_address_1,	daemon-upload-download,
daemon → sendmsg → remote_address_1,	daemon-upload-download,
ruby2.3 → accept → remote_address_2,	server-download,
ruby2.3 → fstat → remote_address_2,	server-daemon-operation,
ruby2.3 → exec → ruby2.3,	init-server-daemon,
ruby2.3 → read → remote_address_2,	server-download,
daemon → sendmsg → remote_address	daemon-upload-download

Privilege escalation phase

init → exec → ircd,	init-server-daemon,
daemon → exec → bash,	server-daemon-operation,
bash → exec → perl,	bash-execute-process *perl*,
perl → exec → perl,	execute-process,
perl → exec → bash,	execute-process *bash*,
bash → exec → temp_process,	bash-execute-process,
temp_process → exec → su	execute-process *su*

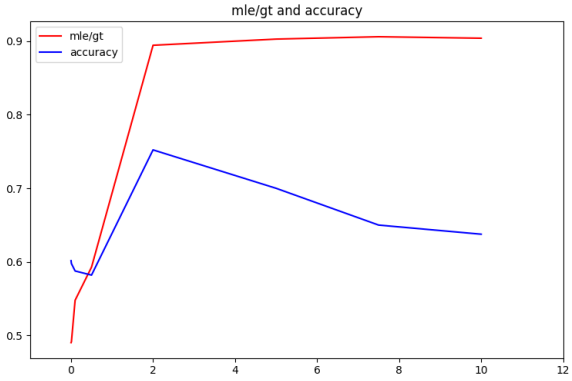

Figure 4.1: Model results based on pseudocounts

The first results reported are the log likelihoods of the system call sequences in the five test folds. When an attack sequence is passed to the Viterbi algorithm, the HMM generates the sequence of states that, with the highest probability, emitted the observations. That is, the state sequence is the one that most likely explains the attack sequence. The likelihood is the calculated probability that the generated state sequence emitted the observations. Table 4.2(a) shows the mean log likelihood for each fold. Note that the log likelihoods were first divided by the lengths of the sequences to normalize before calculating the means. Some sequences in the dataset were not predicted with high probability, thus leading to noticeable standard deviations in some folds. When the learned HMM encounters unseen system calls in sequences, a hard zero is given, preventing the HMM from predicting the sequence. To overcome this, pseudocounts were added during initialization, giving non-zero values, allowing the sequences to be predicted. To select the pseudocount suitable for our model, a range of numbers were tested, as illustrated in figure 4.1, and **2.0** was chosen.

Next, we evaluate the accuracy of the inferred storylines. For each attack storyline, the HMM generates a sequence of inferred states, the most likely explanation (MLE), along with the probability that the MLE emitted the sequence of observations. This probability is the

Fold	Mean LL
0	-5.647 ± 1.484
1	-5.552 ± 1.519
2	-5.191 ± 1.666
3	-5.891 ± 1.384
4	-5.454 ± 1.332

(*a*)

Fold	Mean LL ratio
0	0.945 ± 0.068
1	0.919 ± 0.096
2	0.937 ± 0.076
3	0.924 ± 0.086
4	0.964 ± 0.05

(*b*)

Table 4.2: (*a*) The mean and standard deviation of the log likelihood per test fold generated by the HMM. The mean across all folds is **-5.547**. (*b*) The log likelihood ratio per test fold. The mean ratio across all folds is **0.938**.

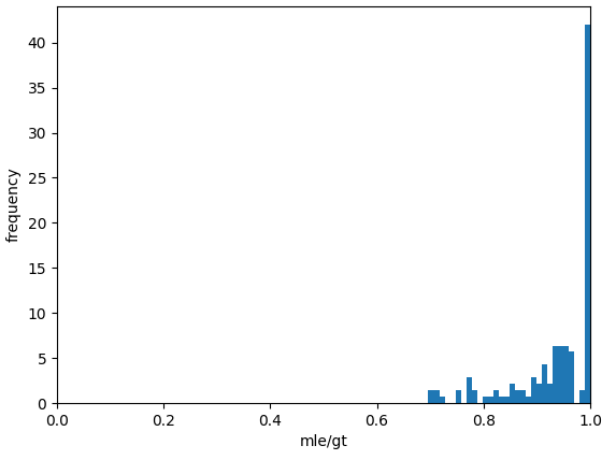

Figure 4.2: mle / gt score of all sequences.

log likelihood of the sequence. The second component required to get the accuracy of a given sequence is the *ground truth* sequence of states for the sequence. The probability computed by the HMM given the ground truth sequence of states and the corresponding sequence of

observations is the ground truth log likelihood. Using both the log likelihood and ground truth log likelihood, we calculate the likelihood ratio using

$$\frac{\text{LL of MLE}}{\text{LL of ground truth}}$$

In Table 4.2(b), we report the means and standard deviations of the likelihood ratio of the fold mean rather than individual sequences. It is worth noting that a ratio of 1.0 would mean that the actual and expected sequences are equal. Despite the fact that none of the folds gave a perfect ratio of 1.0, the folds are relatively close to 1.0 with minimal standard deviations, showing that the generated attack storylines were mostly correct. Figure 4.2 gives the results of individual sequences.

Model	Assetdisc.	Sys.recon.	Exfil	N/wdisc.	Persist	Priv.Escal.
HMM	0.959	0.977	0.927	**0.850**	1.0	0.987

Table 4.3: Mean log likelihood ratio of the HMM by attack phase. The lowest performance among the attack phases is highlighted.

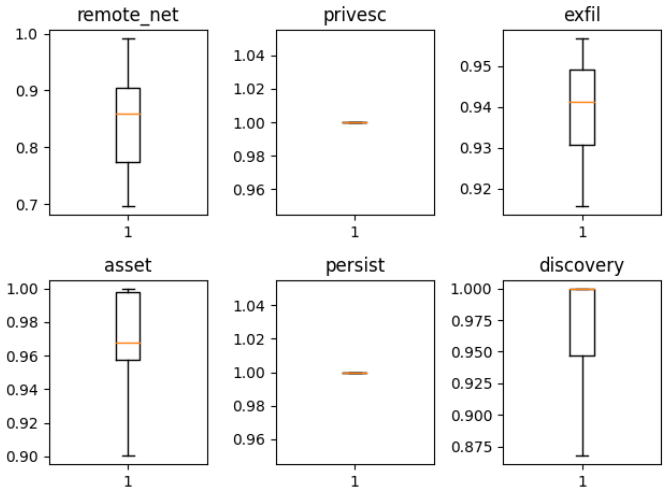

Figure 4.3: mle / gt score by phase.

The sequences are then decomposed by attack phase for further analysis. Table 4.3 contains the likelihood ratios by attack phase, and figure 4.3 contains the ratios in box plot

form. Persistence attacks gave a perfect score, with privilege escalation near 1.0. On the other hand, the mean asset discovery sequences gave a score of 0.85, due to the fact that some observations in several asset discovery sequences were consistently predicted incorrectly by the HMM.

4.3 Classifier Evaluation

To illustrate the significance of the HMM in identifying attack phases, each classifier is evaluated in two ways: one with data that passed through the Cyberian pipeline, and the other with data that bypasses the HMM step in the pipeline. In the first experiment, the ground truth sequences and storylines are both generated by the HMM for each attack phase. The second experiment only uses the sequences of system calls generated by GrAALF without being processed by the HMM. To be clear, storylines were not used in this experiment. To evaluate each model on this system call data set, a 5-fold cross validation was used. The first experiment trains each model with the ground truth sequences and tests them with the storylines.

Model	**Weighted-mean F1**	
	with HMM	**without HMM**
SVM	85.36 ± 7.80	64.79 ± 15.75
CNN	**92.09 ± 7.48**	14.51 ± 16.30
LSTM	82.90 ± 14.63	47.55 ± 26.03

Table 4.4: Weighted-mean F1-score and weighted standard deviation across all attack phases for each of the models with and without the use of storylines. Cross validation was performed in the experiment without storylines, so confusion matrices for each fold were combined to find final F1-scores.

Table 4.4 gives the weighted-mean F1-score for all of the models in both experiments using the non-benign instances. It is apparent that the use of storylines generated by the HMM significantly improves the accuracy of all classifiers, as opposed to only using raw system call sequences. The improvements are significant for the CNN specifically, and the

CNN also achieves the highest performance among the three; results show that it achieves a weighted-mean F1-score of just over **92%** and it is able to accurately identify the type of about 92% of the attack phases. Therefore, the use of HMMs to infer the MLE of observed system calls proves to be of great value in the classification of attack phases.

Chapter 5

Conclusion and Future Work

Attack storyline generation using the HMM has shown promising results in cybersecurity systems such as Cyberian. The Cyberian pipeline starts from raw system call analysis using GrAALF, then storyline generation through an HMM, finally ending with CNN-based machine learning. With the HMM, the classification accuracy was recorded at **92**%, demonstrating the significance of the HMM in Cyberian whose objective is to identify actionable phases of attacks from large system logs.

The resultant attack phases identified by Cyberian could allow cyber defenders to make informed decisions when responding to system attacks. This information provides insight into understanding the intent of attackers, which could, in turn, be used to deceive them in a form of novel cyberdeception.

Despite the fact that many of the log files being analyzed are of a sizeable nature, the number of log files is still considered to be small. Attacks, however, tend to be infrequent, showing that it is possible to achieve high accuracy given this limitation. A future direction would be to experiment with more sophisticated attacks, thus increasing the diversity of the dataset and observation space. Adding multi-host systems would help in this as well.

The HMM step can also be compared with other models such as CRFs (see 3.2) and

Deep Markov Models DMMs [10]. DMMs contain the structure of HMMs, while replacing the emission and transition matrices with multi-layer perceptrons (MLPs).

Finally, since we rely on system call names when generating attack storylines, there is a chance that we misclassify a sequence. If an attacker has privileges such that they can rename processes, our system may annotate the system call with the wrong state name. This will not severely affect the state sequence inferred, as only a small number of states will be affected; we focus on the overall sequence to describe the behavior of the attacker and not individual states.

Bibliography

[1] Leonard E Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, 41(1):164–171, 1970.

[2] Damiano Bolzoni, Sandro Etalle, and Pieter H Hartel. Panacea: Automating attack classification for anomaly-based network intrusion detection systems. In *International Workshop on Recent Advances in Intrusion Detection*, pages 1–20. Springer, 2009.

[3] Ronny Chevalier. *Detecting and Surviving Intrusions: Exploring New Host-Based Intrusion Detection, Recovery, and Response Approaches*. PhD thesis, CentraleSupélec, 2019.

[4] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1285–1298, New York, NY, USA, 2017. ACM.

[5] Karen A Garcia, Raul Monroy, Luis A Trejo, Carlos Mex-Perera, and Eduardo Aguirre. Analyzing log files for postmortem intrusion detection. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1690–1704, 2012.

[6] Xueyuan Han, Thomas Pasquier, and Margo Seltzer. Provenance-based intrusion de-

tection: opportunities and challenges. In *10th {USENIX} Workshop on the Theory and Practice of Provenance (TaPP 2018)*, 2018.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[8] Md Nahid Hossain, Junao Wang, Ofir Weisse, R Sekar, Daniel Genkin, Boyuan He, Scott D Stoller, Gan Fang, Frank Piessens, Evan Downing, et al. Dependence-preserving data compaction for scalable forensic analysis. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1723–1740, 2018.

[9] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998.

[10] Rahul G Krishnan, Uri Shalit, and David Sontag. Structured inference networks for nonlinear state space models. In *Thirty-first aaai conference on artificial intelligence*, 2017.

[11] Nicholas C Laan, Danielle F Pace, and Hagit Shatkay. Initial model selection for the baum-welch algorithm as applied to hmms of dna sequences. *Queen's University, Kingston, ON, Canada*, 2006.

[12] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning 2001*, 2001.

[13] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. High accuracy attack provenance via binary-based execution partition. In *NDSS*, 2013.

[14] Song-Mi Lee, Sang Min Yoon, and Heeryon Cho. Human activity recognition from accelerometer data using convolutional neural network. In *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 131–134. IEEE, 2017.

[15] Richard P Lippmann, David J Fried, Isaac Graf, Joshua W Haines, Kristopher R Kendall, David McClung, Dan Weber, Seth E Webster, Dan Wyschogrod, Robert K Cunningham, et al. Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation. In *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, volume 2, pages 12–26. IEEE, 2000.

[16] Ming Liu, Zhi Xue, Xianghua Xu, Changmin Zhong, and Jinjun Chen. Host-based intrusion detection system with system calls: Review and future trends. *ACM Computing Surveys (CSUR)*, 51(5):98, 2018.

[17] Sadegh M Milajerdi, Rigel Gjomemo, Birhanu Eshete, R Sekar, and VN Venkatakrishnan. Holmes: real-time apt detection through correlation of suspicious information flows. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1137–1152. IEEE, 2019.

[18] Paolo Missier, Jeremy Bryans, Carl Gamble, Vasa Curcin, and Roxana Danger. Provenance graph abstraction by node grouping. *School of Computing Science Technical Report Series*, 2013.

[19] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[20] Vangipuram Radhakrishna, Puligadda Veereswara Kumar, and Vinjamuri Janaki. A novel similar temporal system call pattern mining for efficient intrusion detection. *J. UCS*, 22(4):475–493, 2016.

[21] Rapid7. metasploit. https://metasploit.com/, 2020. [Online; accessed 20 Jan 2020].

[22] Jacob Schreiber. Pomegranate: fast and flexible probabilistic modeling in python. *The Journal of Machine Learning Research*, 18(1):5992–5997, 2017.

[23] Omid Setayeshfar, Christian Adkins, Matthew Jones, Kyu Hyung Lee, and Prashant Doshi. Graalf: Supporting graphical analysis of audit logs for forensics. *arXiv preprint arXiv:1909.00902*, 2019.

[24] Inc sysdig. Sysdig. https://sysdig.com/, 2019. [Online; accessed 25 May 2019].

[25] Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.

[26] David Wagner and Paolo Soto. Mimicry attacks on host-based intrusion detection systems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 255–264, 2002.

[27] Wei Wang, Xiao-Hong Guan, and Xiang-Liang Zhang. Modeling program behaviors by hidden markov models for intrusion detection. In *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826)*, volume 5, pages 2830–2835 vol.5, Aug 2004.

[28] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.