

Spring 2018

## Intrusion Detection in Containerized Environments

Shyam Sundar Durairaju  
*San Jose State University*

Follow this and additional works at: [https://scholarworks.sjsu.edu/etd\\_projects](https://scholarworks.sjsu.edu/etd_projects)



Part of the [Computer Sciences Commons](#)

---

### Recommended Citation

Durairaju, Shyam Sundar, "Intrusion Detection in Containerized Environments" (2018). *Master's Projects*. 637.

DOI: <https://doi.org/10.31979/etd.g8hf-tbtb>

[https://scholarworks.sjsu.edu/etd\\_projects/637](https://scholarworks.sjsu.edu/etd_projects/637)

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact [scholarworks@sjsu.edu](mailto:scholarworks@sjsu.edu).

# Intrusion Detection in Containerized Environments

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In partial fulfillment

Of the requirements for the Degree

Master of Science

by

Shyam Sundar Durairaju

May 2018



\

The Designated Project Committee Approves the Project Titled

Intrusion Detection n Containerized Environments

by

Shyam Sundar Durairaju

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2018

Dr. Robert Chun                      Department of Computer Science

Dr. Thomas Austin                      Department of Computer Science

Mr. Praveen Subramaniyam              NVIDIA Corporation

## ABSTRACT

In this paper, we present the results of using Hidden Markov Models for learning the behavior of Docker containers. This is for use in anomaly-detection based intrusion detection system. Containers provide isolation between the host system and the containerized environment by efficiently packaging applications along with their dependencies. This way, containers become a portable software environment for applications to run and scale. Unlike virtual machines, containers share the same kernel as the host operating system. This is leveraged to monitor the system calls of the container from the host system for anomaly detection. Thus, the monitoring system is not required to have any knowledge about the container nature, neither does the host system or the container being monitored need to be modified.

## ACKNOWLEDGEMENTS

I am grateful and take this opportunity to sincerely thank my thesis advisor, Dr. Robert Chun, for his constant support, invaluable guidance, and encouragement. His work ethic and constant endeavor to achieve perfection have been a great source of inspiration.

I wish to extend my sincere thanks to Dr. Thomas Austin and Mr. Praveen Subramaniyam for consenting to be on my defence committee and for providing invaluable suggestions to my project without which this project would not have been successful.

I also would like to thank my friends and family, for their support and encouragement throughout my graduation.

# TABLE OF CONTENTS

## Table of Contents

<b>ABSTRACT .....</b>	<b>1</b>
<b>ACKNOWLEDGEMENTS .....</b>	<b>2</b>
<b>TABLE OF CONTENTS.....</b>	<b>3</b>
<b>List of Figures .....</b>	<b>5</b>
<b>CHAPTER 1.....</b>	<b>6</b>
<b>Introduction .....</b>	<b>6</b>
<b>CHAPTER 2.....</b>	<b>8</b>
<b>Preliminaries .....</b>	<b>8</b>
2.1 Introduction .....	8
2.2 Intrusion detection.....	9
2.3 Virtual machines and containers .....	10
2.4 Machine learning techniques.....	12
2.5 Supervised learning.....	12
2.6 Unsupervised learning .....	13
2.7 Hidden Markov Models (HMM) .....	13
2.8 Notation .....	15
2.9 Analyzing UNIX commands using HMM.....	17
2.10 Schonlau dataset.....	17
2.11 Sysdig.....	19

<b>CHAPTER 3.....</b>	<b>20</b>
<b>System Architecture.....</b>	<b>20</b>
3.1    Basic architecture.....	20
3.2    Logging UNIX commands .....	21
3.3    Environment setup.....	23
3.4    Collecting the data .....	24
<b>CHAPTER 4.....</b>	<b>26</b>
<b>Project implementation .....</b>	<b>26</b>
4.1    Data processing.....	26
4.2    Training the model .....	28
4.3    Testing an observation sequence .....	30
<b>CHAPTER 5.....</b>	<b>33</b>
<b>Model Evaluation.....</b>	<b>33</b>
5.1    Accuracy measure .....	34
5.2    Receiver operating characteristic (ROC) .....	35
<b>CHAPTER 6.....</b>	<b>37</b>
<b>Experimental results .....</b>	<b>37</b>
6.1    The Schonlau dataset results .....	37
6.2    Experiments with system calls from database intrusion .....	40
<b>CHAPTER 7.....</b>	<b>48</b>
<b>Conclusion .....</b>	<b>48</b>
<b>LIST OF REFERENCES .....</b>	<b>49</b>



## List of Figures

Figure 1-Virtual machine .....	10
Figure 2 - Containers .....	11
Figure 3 - HMM states and observation sequence .....	14
Figure 4 - HMM.....	15
Figure 5 - Schonlau dataset.....	18
Figure 6 - System architecture .....	21
Figure 7 - System calls from Docker container running Ubuntu .....	23
Figure 8 - Forward algorithm.....	30
Figure 9 - Forward algorithm.....	31
Figure 10 - Confusion matrix.....	33
Figure 11 - ROC curve example .....	36
Figure 12 - HMM results for Schonlau dataset.....	38
Figure 13- ROC for Schonlau dataset.....	39
Figure 14- HMM scores for low-moderate usage.....	41
Figure 15 - ROC curve for low-moderate usage.....	42
Figure 16 - HMM results for moderate usage.....	43
Figure 17 - ROC for moderate usage .....	44
Figure 18 - HMM results for high usage .....	46
Figure 19 - ROC for high usage.....	47

# CHAPTER 1

## Introduction

Intrusion is the unauthorized access to a system's data or resources by an unauthorized user, the masquerader, or a software program. It is a threat faced by computer systems every day. Intrusion detection systems (IDS) and Intrusion Prevention Systems (IPS) are used to detect and prevent intrusions. IDS monitor networks and host systems to detect anomalies in system behavior, and abuse of system resources or policies. This includes collecting system usage information, and creating profiles based on normal and anomalous usage patterns [1]. Profiles can be created based on a variety of parameters like CPU usage, login time, application usage order, login location, login time, session times (start and end), commands issued, system calls etc. If an abuse or anomalous activity that does not match the normal usage profiles is detected, then the session can be reported as an intrusion.

The IDS are broadly classified into host intrusion detection systems (HIDS) and network intrusion detection systems (NIDS). HIDS monitors activity on a host system, whereas the NIDS analyzes network traffic on a host system or a network, like a network firewall. These systems detect anomalies using an anomaly-based detection technique, or a signature-based detection technique. Signature-based IDS look for specific patterns, such as known instruction organization in malicious attacks, or packet sequences in network traffic. The signature-based IDS are efficient in detecting known attacks, however these systems cannot detect new attacks since no patterns are available [2]. Anomaly-based IDS are mainly used to detect unknown attacks. This is because of the rapid development and changes in malware. The system creates a profile of normal system usage and then compares the new activity against this model. Though this system is effective in

detecting intrusions, the system is prone to false positives, and training this type of system is a challenging task in terms of resources and time required.

The Hidden Markov Model (HMM) [3] is a powerful tool that can extract significant statistical information from data. Even when the underlying assumption of a Markov process is questionable, HMM's are often applied with success. This project uses a logging tool to log the system calls that a container running a database makes to the host system. The data is collected as files of different sessions, epochs. The collected data is cleaned for processing using HMM. The HMM is trained using normal user behavior and can be used to detect anomalous behavior.

The dataset created contains all the UNIX system calls that a container makes to the host operating system. This is similar to the Schonlau dataset [4] which contains only of a sequence of UNIX commands collected from users.

This paper is organized as follows: Chapter 2 presents preliminaries. Chapter 3 shows the basic system architecture, data collection and processing techniques used in the system. Chapter 4, 5 give the details about the implementation of the project. Chapter 6 explains the results and comparison with the previous dataset. Finally, Chapter 7 presents the conclusion and future work.

## CHAPTER 2

### Preliminaries

This chapter discusses the previous work related to intrusion detection.

#### 2.1 Introduction

Computer systems face the threat of intrusion every day. As a protective measure, authentication systems and intrusion detection systems are used to detect and prevent intrusions. Authentication systems secure host systems [5] with passwords, physical authentication devices (security tokens), biometric systems (fingerprint and face detection), and in the recent days by using virtualization. IDS and IPS on the other hand detect and prevent intrusions.

Intrusion detection systems are classified based on the mode of application, being host monitoring systems called as host intrusion detection systems, and network monitoring systems called network intrusion detection systems. The IDS's detect any violation of normal usage patterns, and any anomalies are detected and reported.

These systems detect intrusions using either a signature-based detection technique, or an anomaly-based intrusion detection technique. Since the ways of intrusion consistently change over time, signature-based anomaly detection techniques are not very effective in detecting the anomaly. Anomaly-based detection techniques on the other hand are a better choice. A model/profile of normal system behavior is created, and using this model, any breaches in normal behavior can be detected [1]. The information used to create this model from a system usage perspective would include CPU usage, login time, application usage order, login location, login time, session times (start and end), commands issued, system calls, etc.

This paper focuses on creating a user profile based on UNIX commands issued by a Docker container to the host system's kernel during normal usage. The profile/model for this ideal usage is created and stored in the system. Each time the application in the container is run, a profile of current system usage is created and compared with the profile that exists in the system. If there is a malicious activity like an intrusion, then an alert is triggered.

## 2.2 Intrusion detection

The problem of intrusion was first documented [6] by James P. Anderson in 1980. The report introduces intrusion detection as a strategy by **analyzing log files of the affected systems**. The idea was improvised and, in a paper, Denning [7] introduces an intrusion-detection model capable of detecting break-ins, penetrations and other forms of computer abuse. The system works on the basis of James P. Anderson's hypothesis that audit trails provide insights into anomalies in computer systems. This was one of the first detection systems to be developed, and it provided a foundation to the future of intrusion-detection systems.

In the following years, there were many changes made to the initial IDS, which is described in a report by Symantec [8]. The improvements included analysis of more data, and introduction of systems that analyzed network data (NIDS) to detect anomalies. This included using techniques like deep packet inspection and pattern matching. It was not until the year 1990, when the first commercial intrusion detection system was introduced [9].

## 2.3 Virtual machines and containers

A virtual machine is the software replica of the underlying computer system [10]. The virtual machines provide the user with the same experience as the original system, but with clear isolation between the physical and virtual environments. Virtual machines use specialized software called **hypervisors**, which emulate the underlying hardware for the virtual systems. This ultimately provides a hardware level virtualization. Hardware or platform virtualization refers to the act of creating a virtual machine that is like a real machine with an operating system. The **host machine** is the actual machine where the virtualization takes place, whereas the **guest machine** is the virtual machine that gets created on the host system.

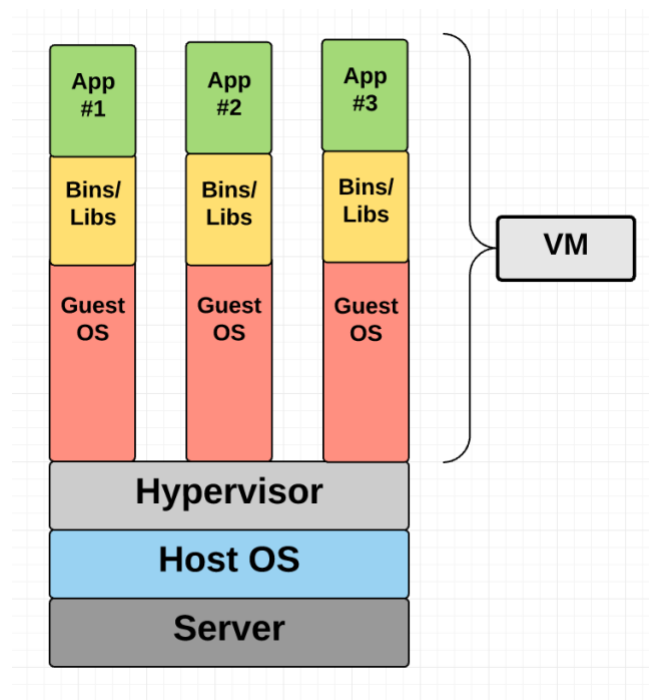


Figure 1-Virtual machine

Containers are a way of packaging and deploying applications along with their dependencies. They are lightweight, portable software environments for applications to run and

scale. Unlike virtual machines, containers provide **operating system level virtualization**. This is accomplished by abstracting the user space. Containers seem like virtual machines with private spaces for **execution, private network interface and IP address**. However, the big difference between containers and virtual machines is that **containers share the same kernel as the host operating system**.

The images (Fig. 1, Fig. 2) illustrates the virtual machines and containers, and how the virtual machines have their own guest OS kernel, whereas the container applications use the same kernel as the host operating system. The operating system's architecture is shared by the different containers in a host system. The only parts that are created from scratch are the bins and libs, making containers lightweight.

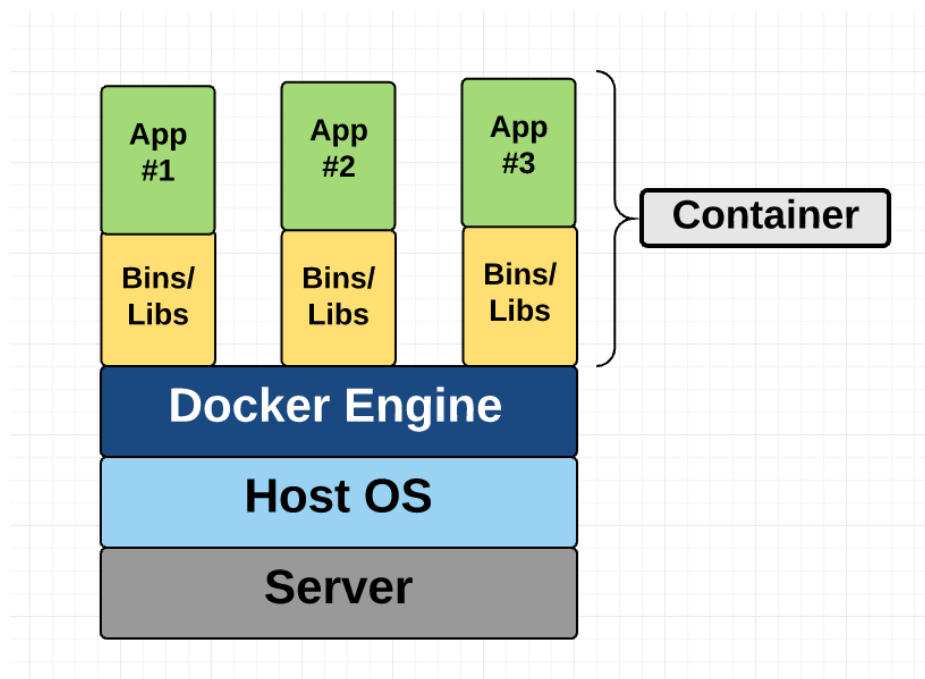


Figure 2 - Containers

## 2.4 Machine learning techniques

Machine learning techniques are widely used to progressively improve the performance of a certain task. Machine learning studies the development of algorithms that can be used to learn from and make predictions on data. Machine learning finds applications in a variety of areas like virtual personal assistants, predictions while commuting, self-driving cars, video surveillance, spam and malware filtering, search engine refining, online customer support, product recommendations, online fraud detection, analyzing the human genome and a lot more.

One example where machine learning is being used in everyday life is in traffic predictions. Mobile phone users use GPS for navigation services for commuting between places. While users get the directions to their desired destination, the user's location and velocity is being stored at a central server. This data is then used to build a map based on current traffic around the user's location. With data from many users in the same location and historical data, the maps application is effectively able to analyze patterns in traffic. With this data, and using appropriate machine learning algorithms, traffic congestions can be predicted.

Machine learning tasks are broadly classified into supervised learning and unsupervised learning.

## 2.5 Supervised learning

When the training of the machine learning task is performed by mapping every input with the corresponding target, it is called supervised learning. Once trained, the machine learning algorithm will be able to provide the target for any new input after sufficient training. The learning algorithm maps a function that maps to a target for a specified input data. If the target is expressed as classes, then the problem is a classification problem. For example, classifying a file under scrutiny as



benign or malicious. Alternatively, if the target is continuous, it is called a regression problem. For example, the problem of predicting housing prices at a location based on historical prices.

## 2.6 Unsupervised learning

On the contrary, if the machine learning algorithm is trained only with a set of input, it is called unsupervised learning. In this type of learning, the algorithm tries to differentiate the input based on characteristics in the data. The algorithm ultimately generates a structure or relationship between the different inputs. One of the important unsupervised learning algorithm is clustering. The clustering algorithms analyze the data and place the data in separate clusters. When a new input is provided, the algorithm maps the input to an appropriate cluster. Other than clustering, some of the unsupervised learning algorithms are Expectation-maximization algorithm, anomaly detection, singular value decomposition, etc. An example of application of the unsupervised clustering algorithm is the clustering of news articles by Google News. The service clusters news articles based on keywords and terms in the article and places the articles in appropriate clusters like world news, sports, technology, etc.

## 2.7 Hidden Markov Models (HMM)

Hidden Markov Model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (i.e., hidden) states [3]. The algorithm finds its application in areas such as reinforcement learning, speech recognition, handwriting recognition, gesture recognition and bio-informatics.

The HMM is a probabilistic state machine and the transitions between states are only dependent on the previous state. The term hidden refers to the Markov process that represents states and exists connected to a given observation sequence. Observation refers to the data that is known and can be observed. For example, the Markov process can be represented using the below image [11] where the states of a rainy or sunny day is observed based on observations like walk, shop and clean. The states are connected by state transition probabilities. And the states are in turn connected to the observations by observation probabilities.

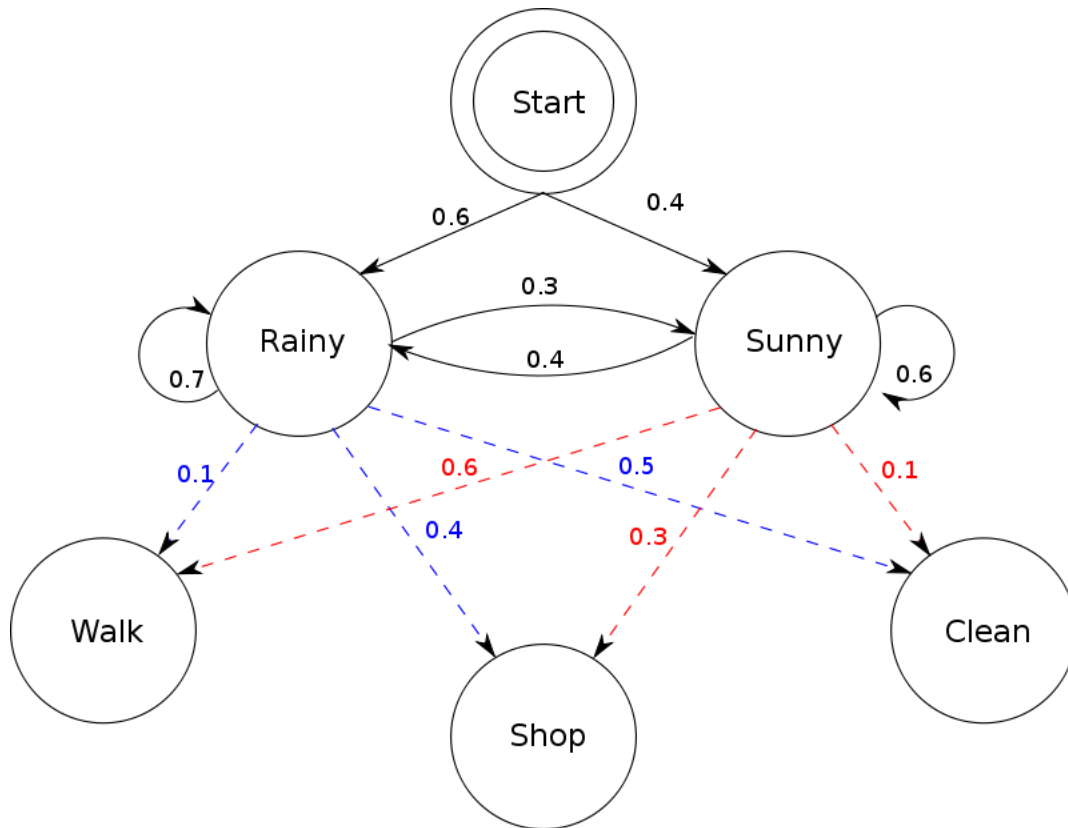


Figure 3 - HMM states and observation sequence

In machine learning sense, the observation sequence is the training data, and the hyperparameter is the states in the model. An interesting property of the HMMs is that the structure of the data can be deduced from the model itself. HMMs are based on discrete probability and a transition from one state to another depends only on the current state and the fixed probabilities. The below image [3] is a general illustration of the HMM,

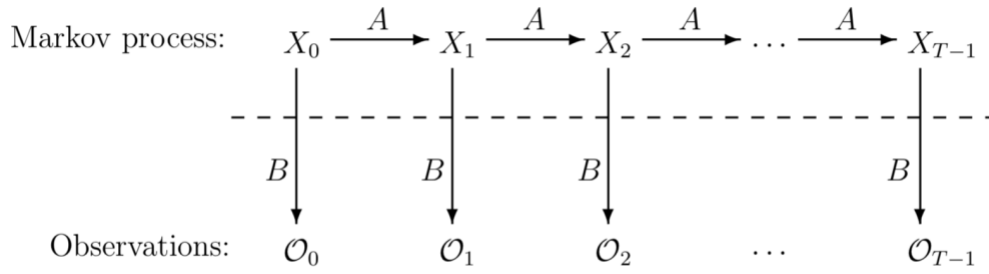


Figure 4 - HMM

## 2.8 Notation

The below notations are used in HMMs,

- $T$  = length of the observation sequence
- $N$  = number of states in the model
- $M$  = number of observation symbols
- $Q = \{q_0, q_1, \dots, q_{N-1}\}$  = distinct states of the Markov process
- $V = \{0, 1, \dots, M-1\}$  = set of possible observations
- $A$  = state transition probabilities
- $B$  = observation probability matrix
- $\pi$  = initial state distribution
- $\mathcal{O} = (\mathcal{O}_0, \mathcal{O}_1, \dots, \mathcal{O}_{T-1})$  = observation sequence.

FIG 4 – HMM NOTATIONS

The **state transitions** between states are connected by the **state transition probabilities (A)**. Similarly, the **transition between the observations** is represented as the **observation probability matrix (B)**.

With a HMM, there are 3 problems [3] that can be solved.

### 1) Problem 1

Given a HMM, what is the likelihood of an observation sequence to be happening. **This is the scoring step of a machine learning process**. With the given observation sequence, we compute the **score of the observation sequence** with respect to the model.

### 2) Problem 2

Given the HMM and an observation sequence (O), what is the optimal hidden state sequence. In other words, this is the process of **uncovering the hidden part of the HMM or decoding the hidden state**. **This is useful to know if the state of weather is “warm” or “cold”**.

### 3) Problem 3

Given an observation sequence (O), what is the model that **maximizes the probability of O**. In machine learning sense, **this is the training phase where the machine learning model tries to find a function that determines the target value**.

In the context of the current project, the UNIX commands that represent a user is the observation sequence. **The observation sequence is connected to states of the sequence being**

normal user behavior and an intrusion. The goal is to create a model of normal user behavior and to determine the probability of a new observation sequence of UNIX commands. A HMM is used here and trained with UNIX commands of normal usage. When a new sequence is scored, the model assigns a high score if the new sequence is similar to the trained sequence and a low score otherwise.

## 2.9 Analyzing UNIX commands using HMM

HMMs have been extensively used in sequence analysis [12]. In the current context, UNIX commands from the Schonlau dataset [4] is the observation sequence. By using the UNIX commands from the data set and leveraging the Problem 3 of the HMMs, a model can be trained to differentiate the states of the machine. This model can be refined so that it maximizes the probability of the observation sequence, the UNIX commands. Later, when a new sequence of UNIX commands is supplied to the model, the probability score can be computed to differentiate the state of the sequence. A high probability means normal usage, and a low probability means anomalous usage.

## 2.10 Schonlau dataset

The Schonlau dataset [4] is a publicly available dataset comprising of UNIX commands collected from system usage based on 50 users. The dataset is organized as 50 files, with each file representing data collected from an individual user. Each file consists of about 15,000 UNIX commands with the first 5000 commands being used for training. The next 10,000 commands are

grouped in blocks of 100 and are seeded with **masquerading** commands. The 10,000 commands are used for testing purpose.

The organization of the dataset is illustrated below [13],

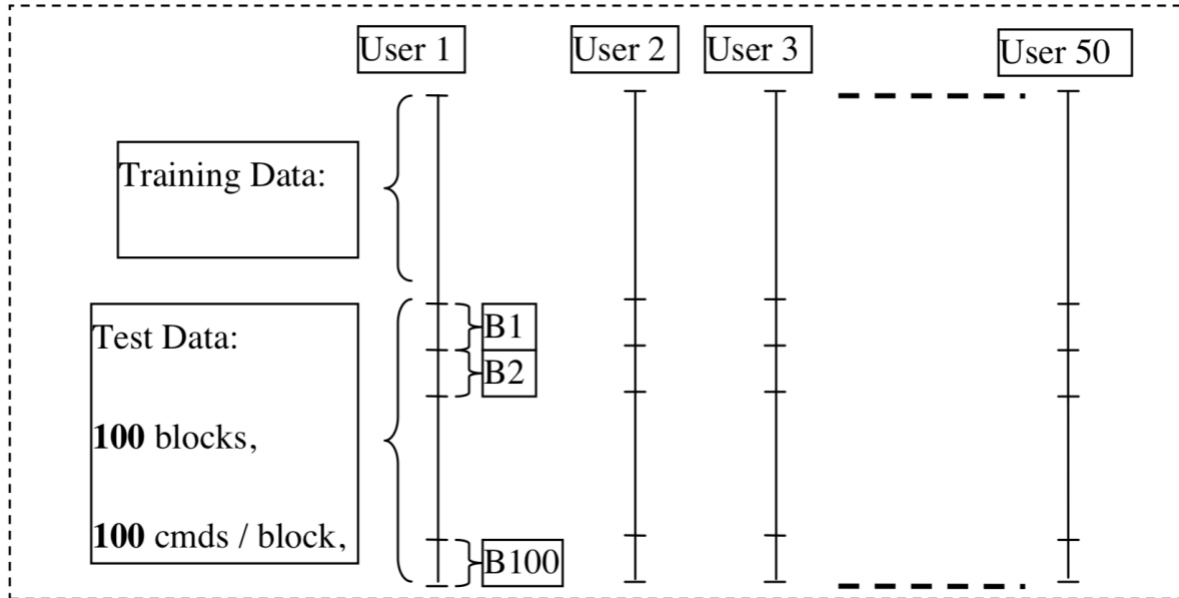


Figure 5 - Schonlau dataset

A limitation of the dataset is that it can only be used for initial training purpose for validating the claim that an intrusion can be detected **by using the UNIX system calls**. However, in a live system, **the operating system executes different systems calls for users**. The order of these system calls is not the same as well. In this case, if a training set is created based on a **certain list of system calls**, and if the operating system executes **a different set of benign system calls**, there are higher chances of it being tagged as **an intrusion**. This is the reason behind the **higher false positives** in the execution of an anomaly detection algorithm.

## 2.11 Sysdig

Sysdig [14] is a Linux tool that is used for system monitoring. The tool installs into the Linux system kernel and captures the system calls and other OS events. In addition to system insights, it offers native support for containers. This can be leveraged to monitor the system calls that a container makes to a host system. Using Sysdig's command line interface, useful information about the system calls made to the host kernel can be retrieved. The package is ideally used to inspect systems live in real-time, or to generate trace files of the system activity.

## CHAPTER 3

### System Architecture

#### 3.1 Basic architecture

To start with, the initial training is performed using the Schonlau dataset. The Schonlau dataset consists of data collected from about 50 users, with files containing both masquerading and normal user data. Using this data, the hypothesis of HMM being able to detect anomalies in UNIX commands can be tested. However, the same model cannot be applied to test the system calls that the containers make to the host operating system. This is because the Schonlau dataset not only contains the system calls collected from user-initiated applications, but also the system calls made by the host operating system.

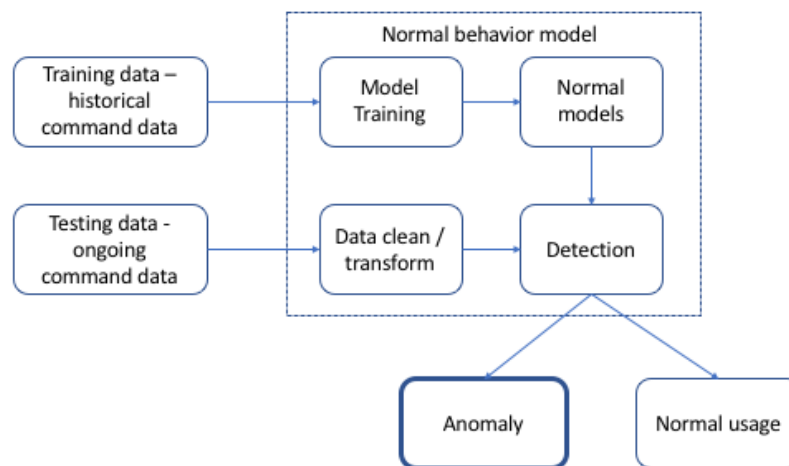
In order to fix this, a new dataset filtered based on the calls made to the host system by a particular Docker container needs to be created. In addition to this, appropriate datasets are collected to support the hypothesis of classifying system state, normal or intruded, based on system calls made during different application loads. This is accomplished by simulating different application loads in the container running the application being monitored. Once the dataset is created, the model is trained based on calls matching normal system usage. This model is then used to determine the state of the container by monitoring the system calls between the host kernel and the container.

The technique used in this paper is a sequence-based anomaly detection technique, which detects anomalies in a system at different application loads. The system calls made by a container to the host system is compared against the machine learning model to predict the state of the



**application.** If the model reports a high probability score, then the sequence is similar to the sequence used to train the model. Otherwise the sequence of system calls is reported as an anomaly.

The below image shows the architecture of the system.



*Figure 6 - System architecture*

### 3.2 Logging UNIX commands

The limitations of using the **Schonlau dataset** for testing is that it contains both **the system calls made by the user, as well as, the operating system**. Though the dataset makes available a large **amount of data in the form of system calls**, the actual **system calls issued by an application running in the containers** would be different. If we were to train a model only using system calls available in the Schonlau dataset, then the system is likely to **classify any other system calls like the ones**

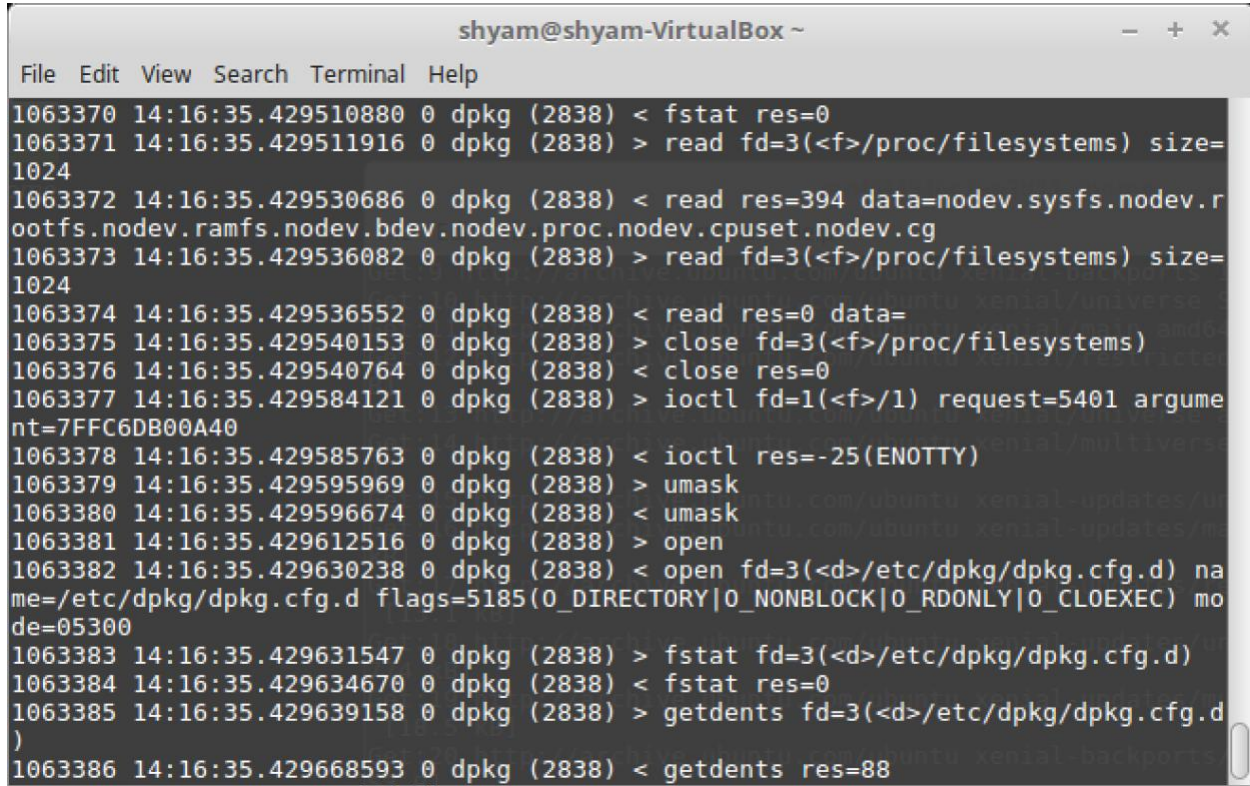
made by a container running an application as an intrusion. This leads to a high possibility of false positives to be reported by the model.

In order to overcome the issue of false positives being detected, the following steps are performed. The first step is logging UNIX commands issued by the container under scrutiny. This data forms the basis of creating a user profile based on normal usage. The collection of the data is done using Sysdig. The tool is tuned so that only calls from the container under scrutiny is logged.

The steps for collecting this data is discussed below,

1. Upon start of a container Sysdig starts collecting the system calls executed by the container that is being monitored.
2. The system calls made by the container to the host system are logged in a separate (file extension used - .scap) file in the host system.
3. Since we are monitoring only a particular container running in the host system, the system calls are filtered to log only the system calls of the chosen container.
4. The logging starts at the start of the container, collects through normal activity and terminates when the container terminates execution.
5. The generated file is processed to create a file with a full list of system calls retrieved from the container.
6. In order to simulate varying system conditions, the collection process is repeated with different application loads (considering a database running inside the container).

Below is a screenshot of the system calls made by a container running Ubuntu to the host operating system during an update session.



```

shyam@shyam-VirtualBox ~
File Edit View Search Terminal Help
1063370 14:16:35.429510880 0 dpkg (2838) < fstat res=0
1063371 14:16:35.429511916 0 dpkg (2838) > read fd=3(<f>/proc/filesystems) size=
1024
1063372 14:16:35.429530686 0 dpkg (2838) < read res=394 data=nodev.sysfs.nodev.r
ootfs.nodev.ramfs.nodev.bdev.nodev.proc.nodev.cpuset.nodev.cg
1063373 14:16:35.429536082 0 dpkg (2838) > read fd=3(<f>/proc/filesystems) size=
1024
1063374 14:16:35.429536552 0 dpkg (2838) < read res=0 data=
1063375 14:16:35.429540153 0 dpkg (2838) > close fd=3(<f>/proc/filesystems)
1063376 14:16:35.429540764 0 dpkg (2838) < close res=0
1063377 14:16:35.429584121 0 dpkg (2838) > ioctl fd=1(<f>/1) request=5401 argume
nt=7FFC6DB00A40
1063378 14:16:35.429585763 0 dpkg (2838) < ioctl res=-25(ENOTTY)
1063379 14:16:35.429595969 0 dpkg (2838) > umask
1063380 14:16:35.429596674 0 dpkg (2838) < umask
1063381 14:16:35.429612516 0 dpkg (2838) > open
1063382 14:16:35.429630238 0 dpkg (2838) < open fd=3(<d>/etc/dpkg/dpkg.cfg.d) na
me=/etc/dpkg/dpkg.cfg.d flags=5185(O_DIRECTORY|O_NONBLOCK|O_RDONLY|O_CLOEXEC) mo
de=05300
1063383 14:16:35.429631547 0 dpkg (2838) > fstat fd=3(<d>/etc/dpkg/dpkg.cfg.d)
1063384 14:16:35.429634670 0 dpkg (2838) < fstat res=0
1063385 14:16:35.429639158 0 dpkg (2838) > getdents fd=3(<d>/etc/dpkg/dpkg.cfg.d
)
1063386 14:16:35.429668593 0 dpkg (2838) < getdents res=88

```

Figure 7 - System calls from Docker container running Ubuntu

### 3.3 Environment setup

For the purpose of experiments, we use a Docker container running the official MySQL database. The container installed as a database is basically an installation of the database in a Linux operating system. The MySQL port is mapped to a custom container port by Docker. Initially, since there is no data set available, multiple experiments are performed to collect the data from the container at different stages of execution. The idea is to collect data based on ideal normal usage of the database and using the data to build a normal usage profile. This data can be used to find and predict the probability of a new observation sequence.

The testing data is generated using `mysqlslap` [15], a diagnostic program developed for emulating client load for a MySQL server. The program reports the timing for different stages at different server loads. The load is simulated as if there are multiple clients accessing the server.

The application runs in 3 stages [15]:

1. Single client stage – where a schema is created, and any stored programs or data is used for testing the database. This simulates requests from a single client to the database.
2. Multi-client stage – another stage of operation runs the load test by simulating multiple client requests.
3. The clean-up stage where the tables are dropped, if any, and the database is disconnected. This is performed using a single client.

The program is used for generating training and test data to be used to train the HMM. It can be customized to run a specified number of iterations to perform load testing. The program also offers customizations to choose the number of queries and iterations to perform on the database. The tool runs on the host operating system and interacts with the database running in the container. This helps to simulate the database server at different server loads: normal database usage, moderate usage and high usage.

### 3.4 Collecting the data

`mysqlslap` simulates different client loads on the database running in the container. During this step, the system call logging tool, `Sysdig`, is used to monitor and log all the system calls that

the container makes to the host operating system. The calls are filtered so that only calls from the container running the database are monitored and logged to a file on the host operating system.

The extracted file contains information like process id, time the call was executed, system call and target data. Since the HMM we consider is a sequence-based model, only the system calls are to be considered. The extracted file is then passed to a custom data cleaning program that filters the system calls to a separate file. The system calls are appended to a continuous string to be ready for processing by the HMM. In the current context, the tracking and cleaning process is manually performed to track a Docker container and extract the system call information.

## CHAPTER 4

### Project implementation

#### 4.1 Data processing

The first step of training the model is processing the data available. Two sets of data available: the Schonlau dataset and the data retrieved from running the database at different usage loads. Each data set is processed differently. The Schonlau dataset contains only system calls in each line of the file. However, the data collected from the Docker contains other information like process id, time, system call, target data and so on. The data cleaning step is performed in the following steps, explained separately for each data set.

Since we use a sequence-based anomaly detection strategy, the end goal is to create an appended string of system calls. This long sequence of system calls is fed to the HMM for training, and later for testing the probability of a new sequence.

A custom python script is used for processing the Schonlau dataset which contains about 50 files, and 15,000 system calls in each file. Each file in a dataset corresponds to system calls collected from a particular user. Each file of the dataset is processed separately to create batches of benign and anomalous files containing separate system calls. For the model, we use the benign calls to train the model, and the anomalous calls to test the model that was created.

1. The individual files are opened, and the file is read line by line to extract individual system calls.
2. A counter is maintained to extract the anomalous and benign calls in batches.

3. Each call is removed of any extra spaces and is converted to lower case. The cleaned system call is appended to a separate file that maintains benign and anomalous system calls.
4. A counter is maintained to append the first 5000 calls to a file of benign system calls. The next 10,000 calls are written in batches of 100 to separate files marked as files seeded with masquerading calls.
5. The process is repeated for the 50 users, which results in the creation of 101 files for each user in the dataset.

The processing for the data extracted from monitoring a Docker container varies to a certain extent. This is because of the additional information that the call logging tool extracts while monitoring the container. The dataset is first created by logging system calls made by the Docker container running the MySQL database. The processing of the file is performed using a custom Python script.

The below steps are performed for each file:

1. Each line of the file is read and validated for presence of system call information. The lines with command line data is excluded.
2. Each line contains additional information like process id, time, system call, target data from which the system call is extracted.
3. Once the system call is extracted, the call is removed of any extra spaces and converted to lower case.
4. Lines that do not contain a system call are skipped.

5. The performance can be improved by using a dictionary of know system calls and can be improved as new system calls are made by a container.

## 4.2 Training the model

For the project, there are 2 sets of data available to perform experiments. The first set is the data available in the Schonlau dataset [4]. This dataset can be used to validate the claim of using system calls to detect anomalies in a system. For this purpose, the HMM to be used in the project is a sequence-based anomaly detection model. This considers the system calls as a continuous sequence. The system calls from the Schonlau dataset are appended to form a long string of characters. The dataset provided is separated into appropriate sets of benign and anomalous datasets.

The Schonlau dataset contains of 15,000 calls collected per user. The 15,000 system calls are made of 5000 system calls and 10,000 anomalous calls grouped in sets of 100. The anomalous call set is seeded with intrusive system calls with a probability of 80% [4]. On reading the file for an individual user, custom scripts are used to clean and separate the individual system calls.

The second set of data is collected from running the mysqlslap application at different application loads. The dataset is the system calls that the container makes to the host operating system. The processing of the data from the system call logger to extract only the system calls is discussed in the previous section. Since we use a sequence-based anomaly detection model, the system calls are appended into a long string to be fed to train the model.



Training the HMM is performed in the following order. The motive is to solve Problem 3 of the HMM, where a model is to maximize the probability of the observation sequence used for training.

1. When the training is started we initiate the transition probability matrix, observation probability matrix, initial probability matrix to random values [3]. The model is re-estimated for maximizing the observation probability of the given observation sequence.
2. The training is started considering 2 hidden states. The hidden states signify the states of the system, being normal behavior and an intruded state.
3. During the training, the observation sequence, being the long string of system calls is fed to the HMM for estimating the state transition probabilities, observation transition probabilities.
4. At the start of training the alpha pass, beta pass is performed, and it is used to compute the gamma and di-gamma values [3]. The model is re-estimated using the gamma and di-gamma values.
5. The iterations are repeated until either a minimum iteration count (1000 in the current experiments) is reached, or when the change in observation sequence probability is very low. The changes in probability starts with higher differences and drops to a lower value once a local maxima is reached. So, a check that stops iterations if the probability change is lower than 0.1 is used.

The training is repeated for datasets generated from different user loads on the MySQL database. This creates different profiles based on the loads experienced by the application. These different user profiles are used for evaluating the new observation sequence during the testing phase of the application.

### 4.3 Testing an observation sequence

Once the normal user profiles are created, a new observation is supplied to test the model. Testing the model is done by using the **forward algorithm** [3]. In the context of a HMM, the forward algorithm provides the probability of a certain sequence with the model. The probability is a measure of how close a sequence is to the sequence that was used to train the model.

The below image [16] illustrates the forward algorithm,

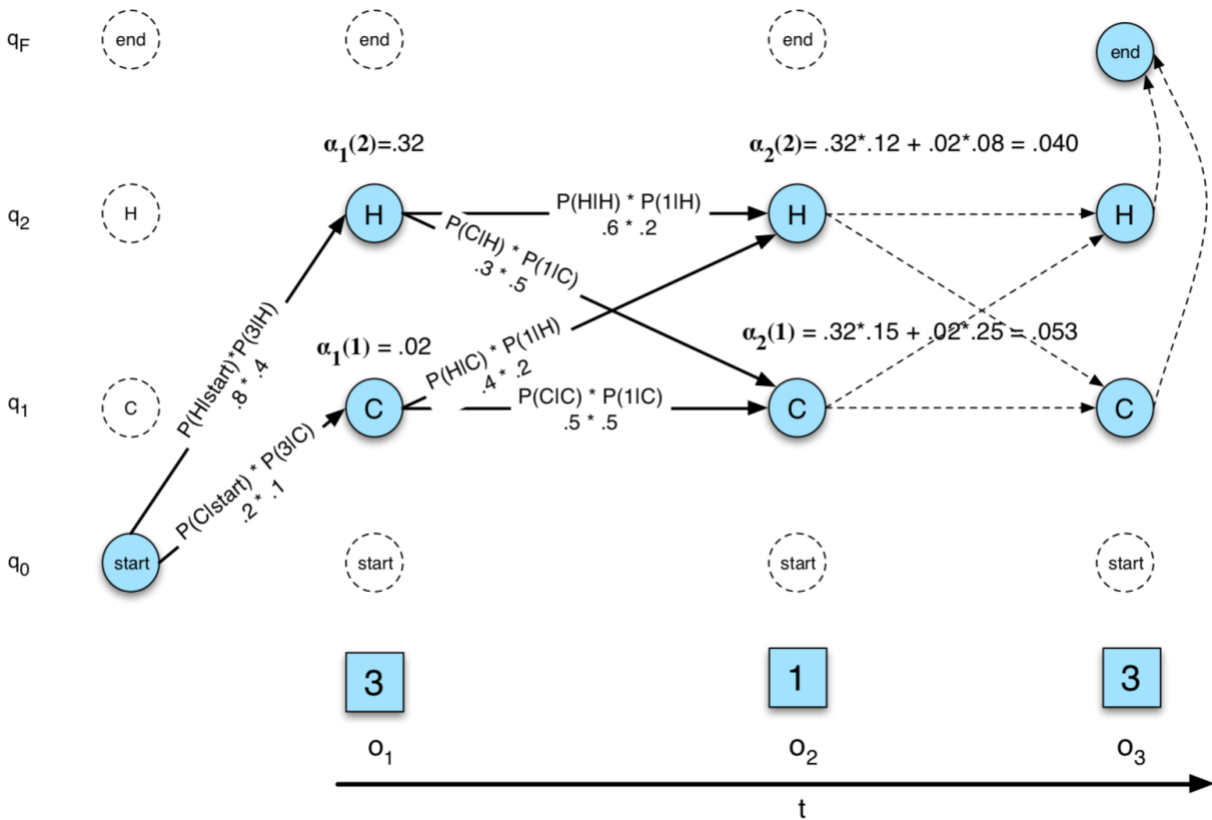


Figure 8 - Forward algorithm

The pseudocode for the forward algorithm is below [16]:

```

function FORWARD(observations of len  $T$ , state-graph of len  $N$ ) returns forward-prob

  create a probability matrix  $forward[N+2, T]$ 
  for each state  $s$  from 1 to  $N$  do                                ; initialization step
     $forward[s, 1] \leftarrow a_{0,s} * b_s(o_1)$ 
  for each time step  $t$  from 2 to  $T$  do                                ; recursion step
    for each state  $s$  from 1 to  $N$  do
       $forward[s, t] \leftarrow \sum_{s'=1}^N forward[s', t-1] * a_{s',s} * b_s(o_t)$ 
   $forward[q_F, T] \leftarrow \sum_{s=1}^N forward[s, T] * a_{s,q_F}$                 ; termination step
  return  $forward[q_F, T]$ 

```

Figure 9 - Forward algorithm

Testing is done by first recording a new set of system calls from the container being monitored. In order to simulate an intrusion attack on the database, we use a penetration testing tool. The function of the tool is to perform an automatic injection attack on a target MySQL database. In this case, the target database is the database running in the container. For this purpose, we use sqlmap [17] an automatic penetration testing tool. Once the attack is performed, system calls from the container are logged. The data in the logged file is extracted and processed the same way we process benign system calls.

The processed system calls made by the Docker container that was collected during the penetration test forms our new observation sequence. This is the new observation sequence that will be used for testing with the model. As discussed in the earlier sections of the paper, if the observation sequence corresponds with the sequence that was used to train the model, then a high

probability score is returned. Otherwise, if the new observation sequence does not correspond to the model, a low probability score is obtained.

Initially during the training phase, multiple user profiles were created based on varying usage patterns., i.e. database loads. This step is performed to minimize the occurrence of false positives. The new observation sequence is tested with all the user profiles that were created for a probability score. Based on the probability score, the sequence is classified as a normal usage pattern or an intrusion.

## CHAPTER 5

### Model Evaluation

The performance of the classifier is determined using an accuracy measure of the outcomes. In general, there are four possible outcomes that a classifier produces [18].

1. True positive (TP) – when a classifier correctly scores a sample sequence of system calls as an intrusion.
2. True negatives (TN) – the observation sequence is not an intrusion, and the classifier classifies it correctly as a benign sequence.
3. False positive (FP) – the scored sequence of system calls is not an intrusion, but the classifier incorrectly classifies the sequence as an intrusion.
4. False negative (FN) – the scored sequence of system calls is an intrusion, but the classifier incorrectly classifies the sequence as an intrusion.

The four cases are illustrated in the below image [18],

		Classifier Prediction	
		Positive	Negative
Actual Value	Positive	True Positive	False Negative
	Negative	False Positive	True Negative

*Figure 10 - Confusion matrix*

The true labels indicate correct outcomes, positive and negative. Whereas, the false outcomes, positive and negative, indicate an incorrect outcome. When an experiment is performed, the number for each outcome is populated in the table. This yields a confusion matrix. From values in the confusion matrix we can compute the accuracy of the model. The accuracy is a measure of how correct the model is performing given the data supplied. In addition to accuracy, the true positive rate(TPR) and the true negative rate(TNR) can also be computed.

## 5.1 Accuracy measure

Machine learning algorithms function with the goal to learn patterns from data. Once the algorithm learns a pattern, it attempts to generalize the experience on unseen data and generates a target classification or a continuous range of values. Once a model has been trained and generated, it is important to measure how accurately the model is making predictions. This is called the accuracy measure of a model.

In the current case, we consider two states of the system. This operates similar to a binary classification algorithm, that classifies data into one of two categories. To achieve this classification, the new observation sequence is scored with the to calculate the probability of the sequence. Based on the score, a certain classification threshold, determined using the ROC curve, is used to determine the classification. Any probability value below and above the cutoff threshold are classified differently. In the current scenario, since we consider profiles based on different application loads, a unique threshold value is assigned for each profile created.

Based on the classifications that a model makes, the confusion matrix created. From the confusion matrix the accuracy of the model is computed. The accuracy of a model is calculated using the below mathematical formula. The higher the accuracy measure, the better is the performance of the model.

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$$

## 5.2 Receiver operating characteristic (ROC)

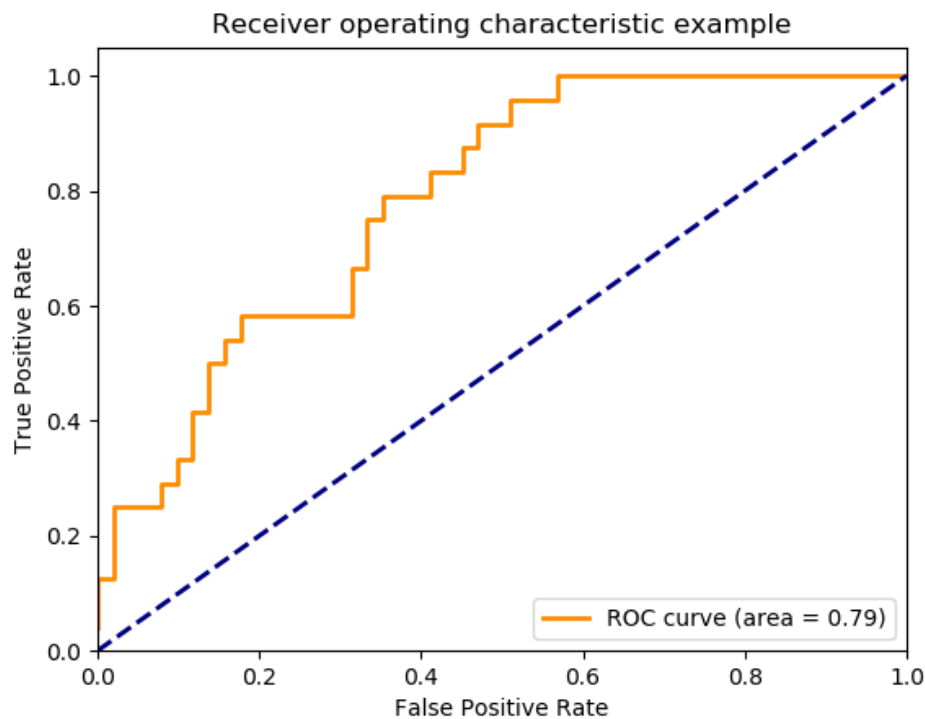
The performance of a machine learning algorithm is determined using the ROC curve [18]. It illustrates the diagnostic ability of a model. Using the ROC curve, optimal models are chosen, and the suboptimal ones are discarded. The curve is plotted using the TPR and FPR computed using the confusion matrix. The TPR values and FPR are computed for different classifications made by the classifier. This is plotted as points in the ROC curve, which is made of FPR values in the x-axis, and TPR values in the y-axis.

The TPR and FPR values are computed as below,

$$TPR = \frac{TP}{TP+FN}$$

$$FPR = \frac{FP}{FP+TN}$$

A sample ROC curve is show below [19],



*Figure 11 - ROC curve example*

ROC curves show the performance of a machine learning algorithm. The closer the curve is to the left-hand border of the graph, the more accurate the test results are. As the curve slopes downward to a 45 degree angle, the less accurate the test results are. From the curve in the ROC diagram we can compute the AUC, which is a measure of classification accuracy. The AUC is a measure of the discrimination ability of the classifier, which signifies the ability of a classifier to correctly classify a new test dataset supplied to the model.



## CHAPTER 6

### Experimental results

The training is performed applying the technique described in the training section. Once trained the different user profiles that signify normal usage are stored in the host system. These user profiles are used to compute the probability of a new observation sequence. The testing step computes the probability score of a new observation sequence, using the forward algorithm [3]. Since there were multiple user profiles created considering different user loads on the database, the observation sequence is to be tested with the different models that was generated.

The experiments were performed in 2 stages, and the results are discussed below.

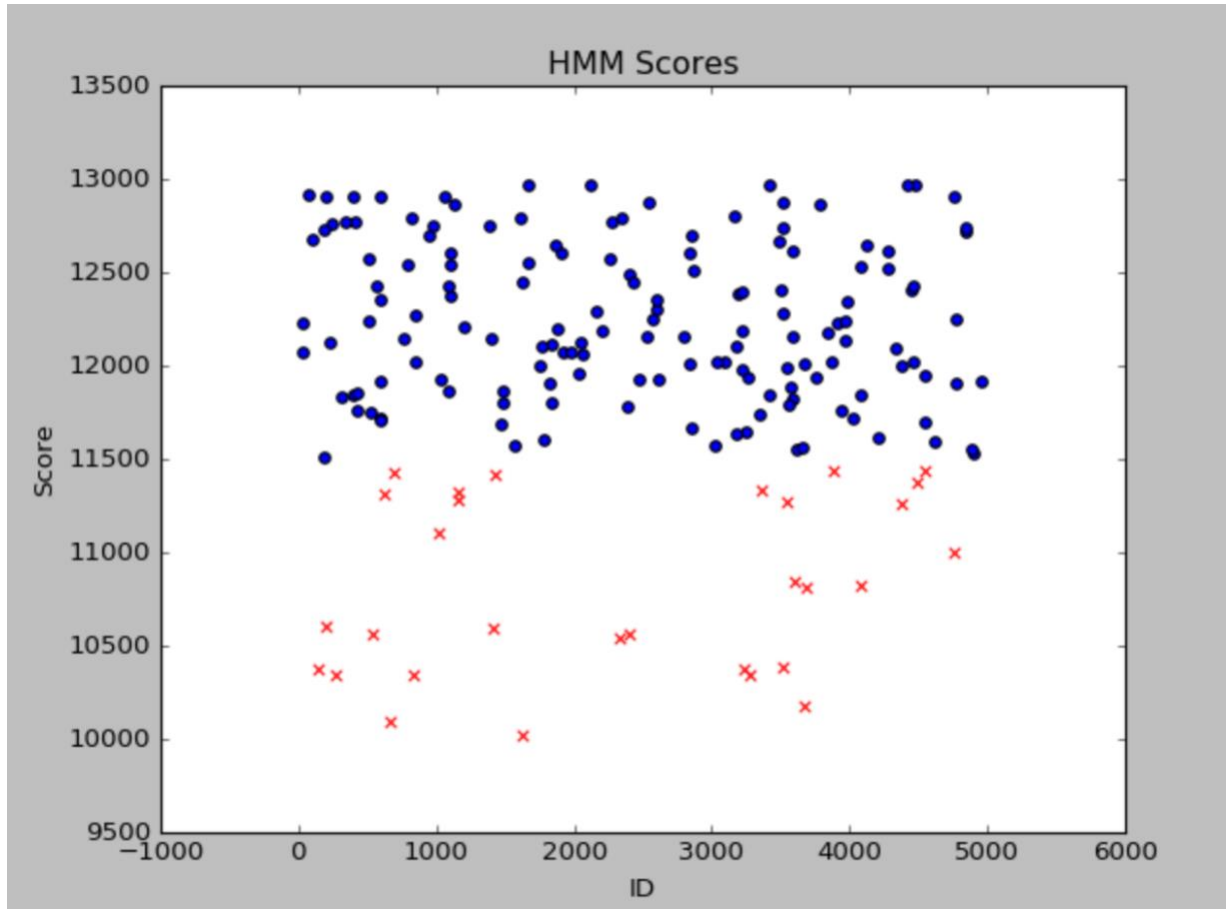
1. Using the Schonlau dataset.
2. Using the dataset generated from the database in the Docker container.

### 6.1 The Schonlau dataset results

In order to prove the claim, the initial experiments are performed using the Schonlau dataset. The model is first trained with the benign calls in the dataset. In this case, the model should return a high probability score for system calls that are benign in nature. For anomalous system calls the model returns a low probability score.

Multiple HMM experiments are performed by considering the data from different user files similar. The model is trained with benign user data and tested using the datasets seeded with anomalous system calls. The output score from the model is used to classify the dataset as benign or anomalous.

The below image shows the scores and datasets that were classified using the HMM,



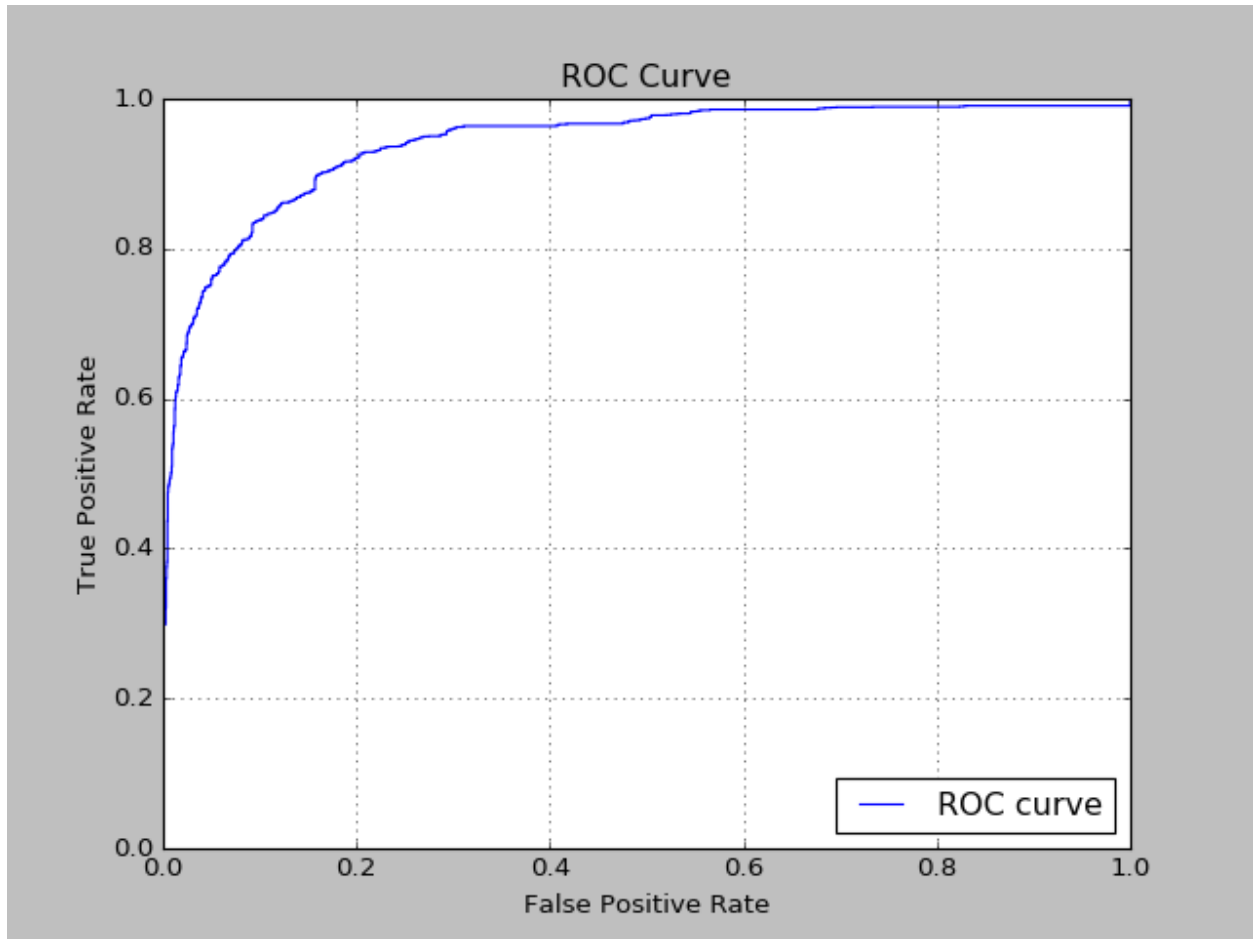
*Figure 12 - HMM results for Schonlau dataset*

The graph shows an illustration of the points that were classified as anomalous (“x”) and normal states (“o”). The circles represent normal user behavior, while the crosses “x” represent anomalous activity.

In the graph the **Y axis** denotes the **log likelihood** of the system calls predicted by the model, while the **X axis** denotes the **index of the dataset** that was considered for generating the score. From the graph, we can observe a separation between the benign and anomalous system calls. However, we can also note the high number of false positives – points that have been **incorrectly classified as intrusive**. A possible reason for the occurrence of false positives is the

limited availability of training data used to train the HMM. HMM's have historically performed bad with low training data as studied in [20].

The ROC curve for the dataset is shown below,



*Figure 13- ROC for Schonlau dataset*

Based on the classifications obtained from the Schonlau dataset the ROC curve is shown in Fig 13. From the ROC curve we can observe the accuracy of predictions made by the model. The highest AUC obtained from the model is 0.92 and the model classifies most of the system call segments into its respective classes. Though the accuracy is comparatively lower than previous HMM experiments performed with the data [21]. We could observe that the model is able to

classify **most of the benign and intrusive calls** in the dataset. With the classifier classifying observation sequences based on the sequence ordering, the experiments are expanded to testing with observation sequences generated from the container.

## 6.2 Experiments with system calls from database intrusion

The testing is performed on the observation sequence of system calls collected from the container running the database. This is done by generating the log likelihood for an observation sequence, and classifying the sequence based on the score. Similar to the testing for the Schonlau, dataset we consider two states in the HMM. The states are assumed to be the states of the machine being, normal behavior or an intruded state of operation.

For training the model, sequences of system calls collected during normal usage at different application loads is used. This results in multiple models based on different usage patterns of the database. While testing the model, we use the new sequence of system calls for scoring against the normal usage models generated from the training data. Since the models are generated considering normal database usage, a normal sequence of system calls would return a higher log likelihood than system calls collected during an anomalous activity. This is an inherent nature of how HMMs work [3].

With this we begin experiments by simulating different application loads at normal and anomalous activity. The experiments are discussed as follows.

## Experiment -1

The first set of experiments is conducted considering low-moderate application usage conditions. The load is simulated using the load simulator, and system call sequences during different times is collected. The process is repeated for normal and when an anomalous event occurs, which is simulated with the penetration testing tool.

We use the sequences of system calls collected from monitoring the container for the scoring phase. The scoring is achieved using the forward algorithm, where each system call sequence is assigned a log likelihood score by the model. The process is repeated for all sets of system call sequences that were collected during the experiment.

The markers with “x” denote the anomalous call sequences, while the markers with “o” denote the benign system call sequences. The results of the experiment are shown below:

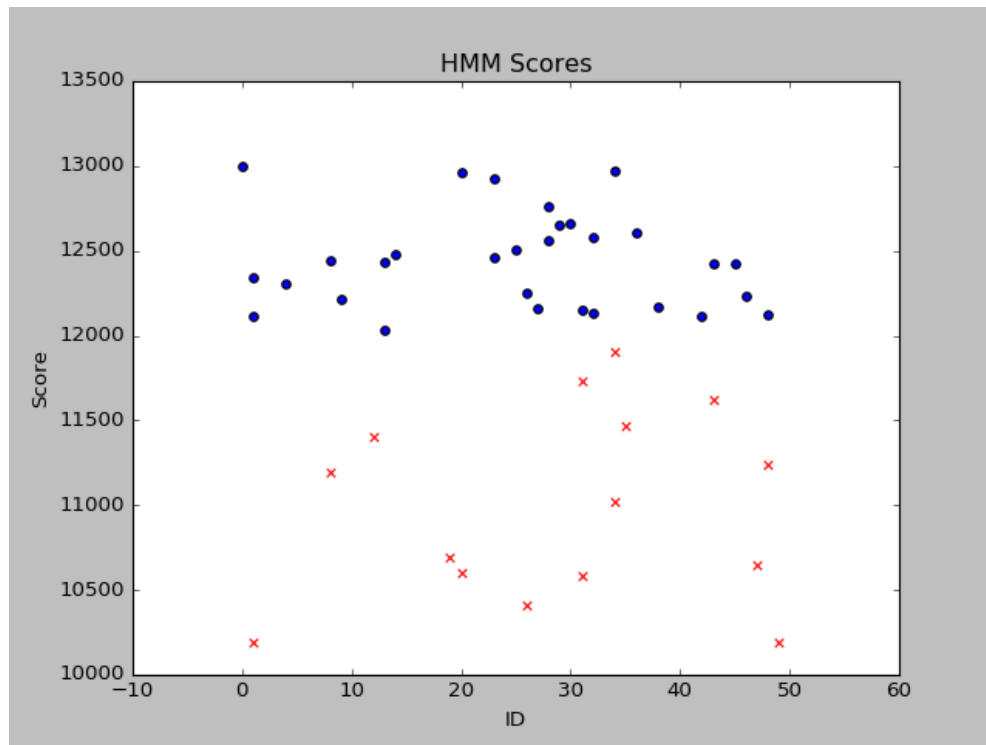


Figure 14- HMM scores for low-moderate usage

From the graph above, we can observe the different sequences of system calls classified as benign and anomalous. The X axis represents the sequence ID that was used for scoring, and the Y axis represents the HMM log likelihood score returned by the model. Since benign system calls were used to train the model, all benign sequences return a higher log likelihood than the anomalous calls. This can be observed from the difference in scores on the higher and lower side of the Y axis.

The ROC curve is plotted for the results and the highest AUC achieved is 0.836. The experiments are then repeated under moderate database usage conditions.

The ROC curve for the experiment:

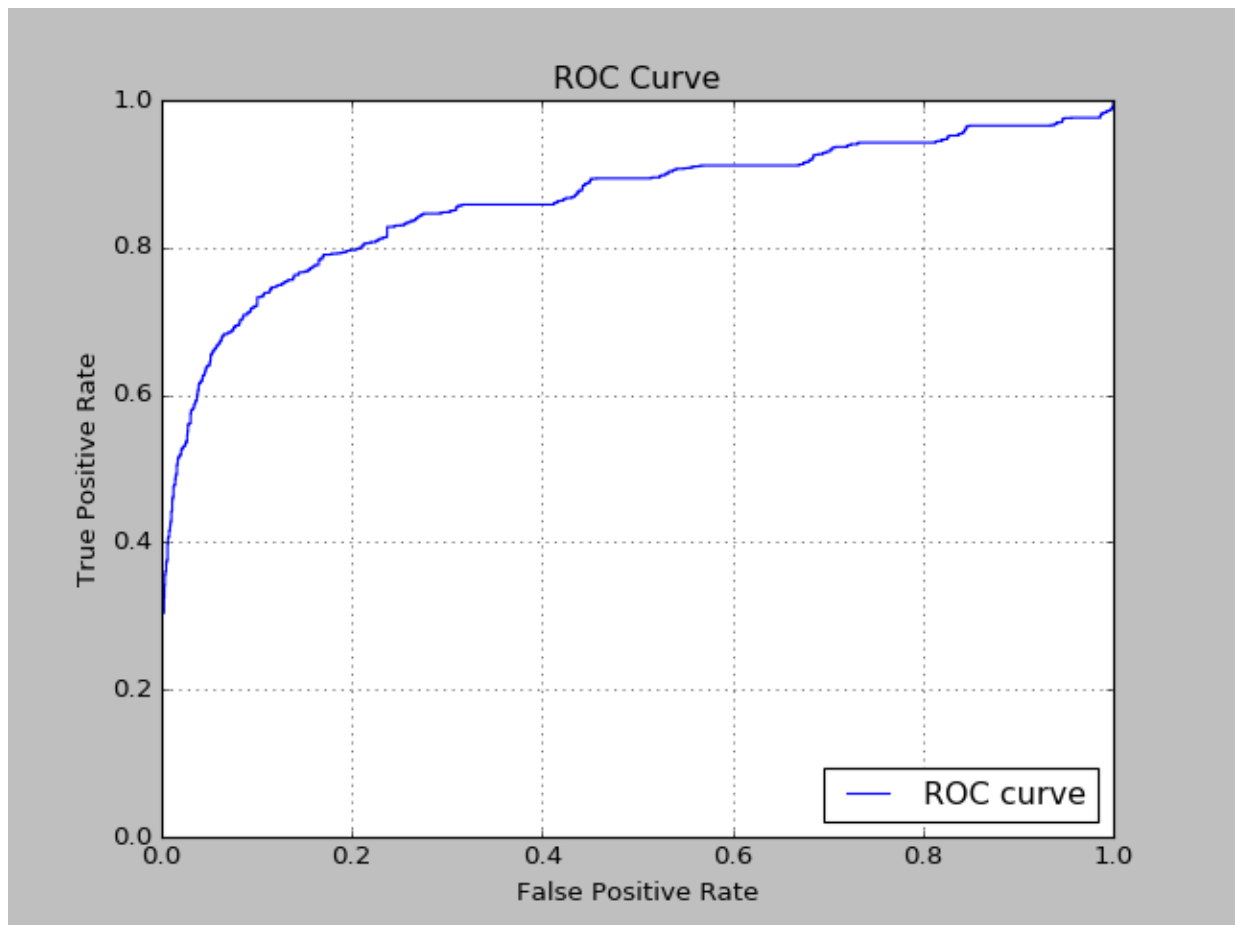


Figure 15 - ROC curve for low-moderate usage

From the graph in Figure 15, we can see that most of the anomalous sequences of system calls are detected by the model. However, a relatively low accuracy is achieved. The possible reason is the occurrence of system calls, though benign, not present in the training set.

## Experiment – 2

The second set of experiments is conducted with system calls that are collected during moderate system usage. This includes a combination of both benign and anomalous system call sequences.

The system call sequences are collected and used in the scoring phase of the application. During this phase the model is used to assign a log likelihood to each sequence of system call. The HMM results are shown below. The markers with “x” denote the anomalous call sequences, while the markers with “o” denote the benign system call sequences.

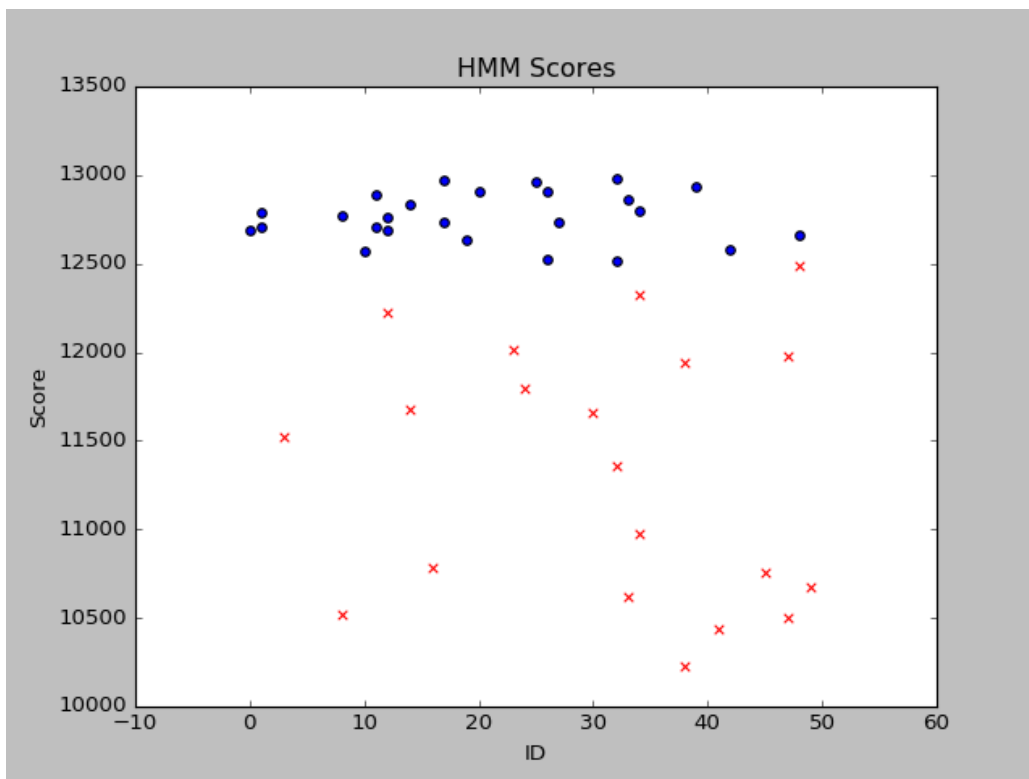
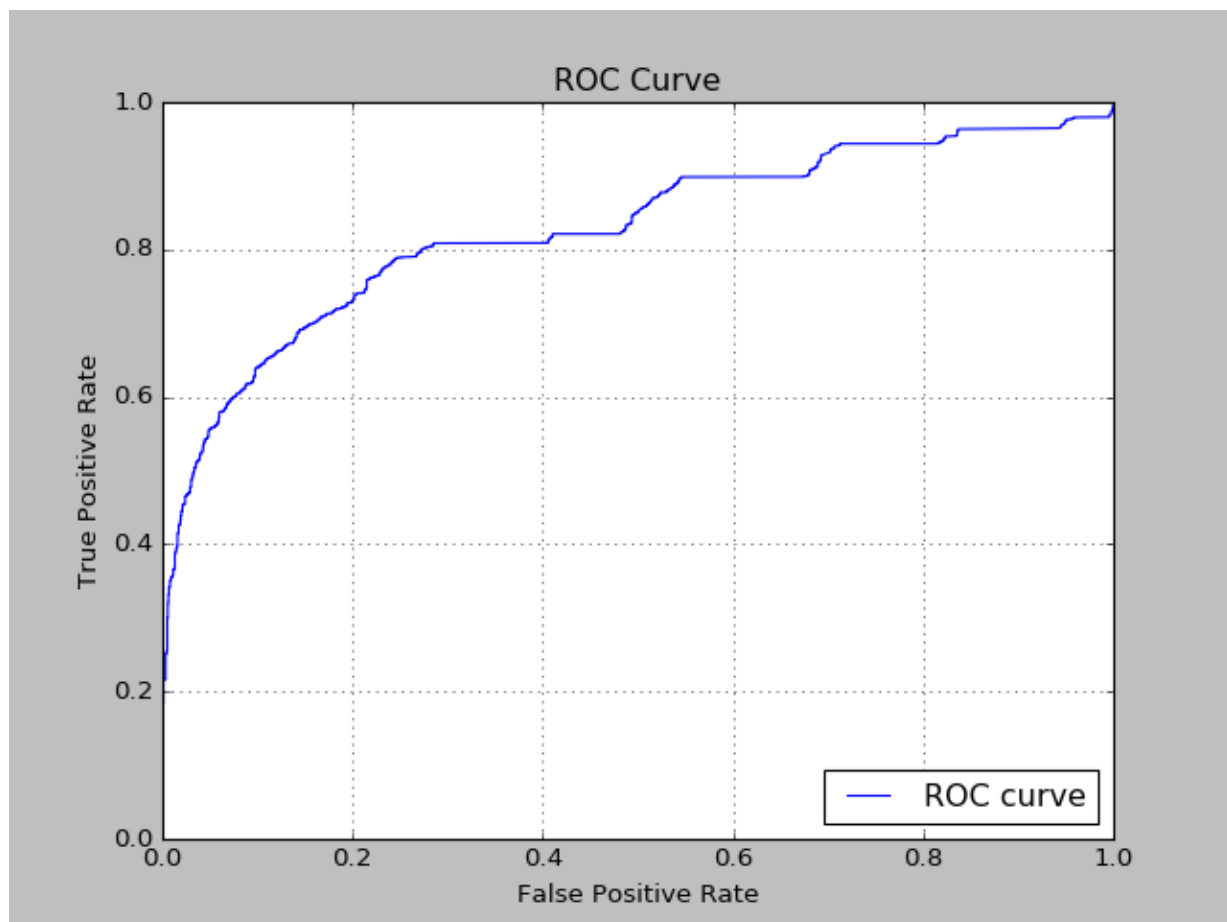


Figure 16 - HMM results for moderate usage

The model classifies most of the system call sequences into the respective categories. However, from the ROC curve we can notice an equal increase in the number of false positives, ie., benign sequences wrongly classified as anomalous. This is because the model assigns a low log likelihood for even the benign sequences of system calls. The reason for the lower performance could be the occurrence of system calls that were not encountered in the training phase, and higher similarity between the anomalous and normal system calls as the usage of the application increases.

The ROC curve is given below:



*Figure 17 - ROC for moderate usage*



As we could observe from the ROC curve, the experiment resulted in a lowered performance than the previous experiment with low system usage. The AUC achieved is (0.812), which is lower when compared to the previous experiments conducted. The lowered performance is due to the increasing similarity between the benign and anomalous system call sequences. This results in the model assigning a similar log likelihood to the benign system calls and anomalous sequences.

### Experiment – 3

The next set of experiments are performed considering high application usage. The system calls for the test set are collected by simulating a high application usage and by performing normal and anomalous usage of the application. The anomalous usage is simulated using the penetration testing tool.

The model training is done using system calls collected from the container during a high system usage event. We use benign system calls for training the model. By repeating the simulation multiple times, several benign system call sequences are collected. These sequences are used to create the different models to be used for scoring new sequences of system calls.

The HMM results for the experiment is below:

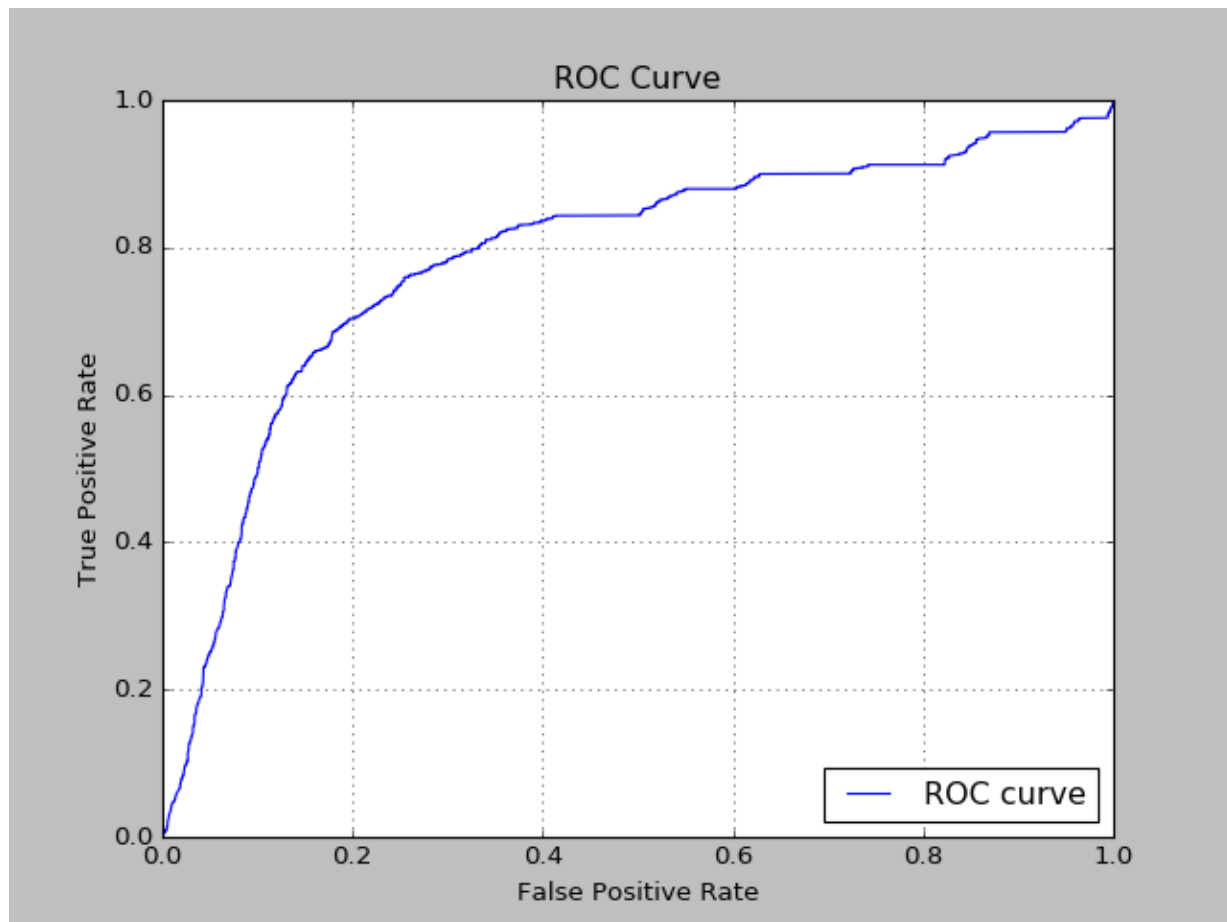


*Figure 18 - HMM results for high usage*

From the graph we can observe the increased occurrence of false positives, i.e., more number of system call sequences being tagged as anomalous activity. This is because of the increased occurrence of system calls not encountered during the training phase. In addition to this, another reason could be the similarity of statistics between the normal and anomalous system calls.

We determine an optimal threshold value to separate the benign and anomalous sequences. The markers with “x” denote the anomalous call sequences, while the markers with “o” denote the benign system call sequences.

The ROC for the experiment is given below,



*Figure 19 - ROC for high usage*

The highest AUC achieved for the experiments was 0.784. As observed in Figure – 19, there is a steep rise in the occurrence of false positives. This is because of the similarity in log likelihood of the anomalous and benign system call sequences. The reason being higher occurrence of system calls not observed during the training phase, and a higher similarity between sequences.

## CHAPTER 7

### Conclusion

In this paper we presented the results of using a sequence based HMM to detect anomalies in system call sequences from an application inside a container. The HMM is trained using system call sequences collected at different application loads. This information is used to classify system call sequences. Using the trained model, we were able to successfully classify anomalous and benign sequences of system calls achieving an overall high AUC of about 0.836 as observed in Figure 15.

The model created was able to classify benign system call sequences from the anomalous ones. However, as the system usage increases, we notice a steady increase in the occurrence of false positives. The reason could be dissimilarities between the benign system call sequences, or the occurrence of system calls not encountered during training. In previous experiments conducted by [22] and [23], a higher accuracy has been reported by considering frequency of system calls, and optimized start states.

We used sequences of system call sequences collected at different application loads. The data is collected, and the experiments are conducted in an offline mode. As future work, the application can be extended to include a workflow that reports anomalies in real time. In addition to this, other features of the data like frequency of system calls in a sequence – using a sliding window, or a bag of words approach [22] can be used to process the data for better performance. Both these approaches are frequency-based anomaly detection techniques which groups system call sequences based on order, or frequency of occurrence. However, a tradeoff would be reduced performance as the requirement for cleaning, and data preparation increases considerably.

## LIST OF REFERENCES

- [1] D. E. Denning, "An Intrusion Detection Model," in *1986 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 1986.
- [2] B. Lokesak, "A Comparison Between Signature Based and Anomaly Based Intrusion Detection Systems," 4 December 2008. [Online]. Available: [www.iup.edu](http://www.iup.edu).
- [3] M. Stamp, "A Revealing Introduction to Hidden Markov Models," 12 January 2018. [Online]. Available: <https://www.cs.sjsu.edu/~stamp/RUA/HMM.pdf>.
- [4] M. Schonlau, "Masquerading User Data," [Online]. Available: <http://www.schonlau.net/intrusion.html>.
- [5] S. Maurer, "Intrusion Detection: An Analysis of User Behavior," Hiram College, [Online]. Available: <https://src.acm.org/binaries/content/assets/src/2010/stefan-maurer.pdf>.
- [6] J. P. Anderson, "Computer Security Threat Monitoring and Surveillance," James P. Anderson Co., Fort Washington, 1980.
- [7] D. E. Denning, "An Intrusion Detection Model," SRI International, Menlo Park, 1983.
- [8] P. Innella, "The Evolution of Intrusion Detection Systems," Tetrad Digital Integrity, LLC, 2001.
- [9] L. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wood and D. Wolber, "A network security monitor," in *Research in Security and Privacy, 1990. Proceedings*, Oakland, 1990.
- [10] M. Laureano, C. Maziero and E. Jamhour, "Intrusion detection in virtual machine environments," in *Proceedings. 30th Euromicro Conference*, Rennes, 2004.

- [11] E. Kang, "Hidden Markov Model," 31 August 2017. [Online]. Available: <https://medium.com/@kangeugine/hidden-markov-model-7681c22f5b9>. [Accessed 18 March 2018].
- [12] B.-J. Yoon, "Hidden Markov Models and their Applications in Biological Sequence Analysis," *Current Genomics*, vol. September, 2009.
- [13] L. Huang, "A Study on Masquerade Detection," December 2010. [Online]. Available: <https://pdfs.semanticscholar.org/44ec/1df941770decfdd6e3bfe5085f5a984d3118.pdf>.
- [14] Sysdig, "Sysdig," Sysdig, [Online]. Available: <https://sysdig.com>.
- [15] "mysqslap," MySQL, 2018. [Online]. Available: <https://dev.mysql.com/doc/refman/5.7/en/mysqslap.html>.
- [16] J. H. M. Daniel Jurafsky, "Hidden Markov Models," 7 August 2017. [Online]. Available: <https://web.stanford.edu/~jurafsky/slp3/9.pdf>.
- [17] sqlmap, "sqlmap," sqlmap, [Online]. Available: <http://sqlmap.org>.
- [18] M. Stamp, Introduction to Machine Learning with Applications in Information Security, Los Gatos: CRC Press, 2017.
- [19] scikit-learn, "Receiver Operating Characteristic (ROC)," 2017. [Online]. Available: [http://scikit-learn.org/stable/auto\\_examples/model\\_selection/plot\\_roc.html](http://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html).
- [20] M. S. Lin Huang, "Masquerade detection using profile hidden Markov models," *Computers and Security*, vol. 30, no. 8, pp. 732-747, November 2011.
- [21] A. Mahajan, "Masquerade Detection Based On UNIX Commands," 2012. [Online]. Available: [http://scholarworks.sjsu.edu/etd\\_projects/273](http://scholarworks.sjsu.edu/etd_projects/273).

- [22] D. F. V. H. Dae-Ki Kang, "Learning Classifiers for Misuse and Anomaly Detection Using a Bag of System Calls Representation," in *Proceedings of the 2005 IEEE*, New York, 2005.
- [23] F. V. G. V. C. K. Darren Mutz, " Anomalous system call detection," in *ACM Transactions on Information and System Security (TISSEC)*, New York, 2006.
- [24] AWS, "Amazon Machine Learning," Amazon, [Online]. Available: <https://docs.aws.amazon.com/machine-learning/latest/dg/binary-classification.html>.