ADVANCED THREAT HUNTING OVER SOFTWARE-DEFINED NETWORKS IN SMART CITIES

By

Steven Schmitt

Farah Kandah
Professor of Computer Science
(Chair)

Li Yang
Professor of Computer Science
(Committee Member)

Anthony Skjellum
Professor of Computer Science
(Committee Member)

ADVANCED THREAT HUNTING OVER SOFTWARE-DEFINED NETWORKS IN SMART CITIES

By

Steven Schmitt

A Thesis Submitted to the Faculty of the University of Tennessee at Chattanooga in Partial Fulfillment of the Requirements of the Degree of Master of Science: Computer Science

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

December 2018

ABSTRACT

The emergence of Software-Defined Networking (SDN) has brought along a wave of new technologies and developments in the field of networking with hopes of dealing with network resources more efficiently and providing a foundation of programmability. SDN allows for both flexibility and adaptability by separating the control and data planes in a network environment by virtualizing network hardware. Threat hunting is a technique that allows for the detection of advanced network threats through forensic analysis. We present an advanced threat hunting model by combining the SDN infrastructure with threat hunting techniques and machine learning models aiming to intelligently handle advanced network threats such as lateral movement. We found that our approach outperforms current threat hunting models in vital areas such as the detection to mitigation time. Our results show that we are able to detect advanced threats with 93.4% accuracy and begin mitigation within 10 seconds of detection.

ACKNOWLEDGMENTS

My research and academic career in both my years as a graduate and undergraduate student could not be possible without the ongoing support from incredible professors. First, my professor and advisor Farah Kandah has been invaluable with his continuous support in research and academic guidance. With his guidance I have become more prepared to approach future research work and a career. Second, my professor Li Yang has been a constant support system in pointing me in the right direction in my career and academic pursuits. I can't say I would be where I am now without the continued help from both professors and many more faculty at The University of Tennessee at Chattanooga. I would also like to thank Anthony Skjellum for both serving on my committee while also providing great insight into the field of computer science.

DEDICATION

This work is dedicated to my friends and family who supported me throughout my academic career.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The emergence of Software-defined Networking (SDN) has brought along a wave of new technologies and developments in the field of networking with hopes of dealing with network resources more efficiently and providing a foundation of programability. SDN allows for both flexibility and adaptability by separating the control and data planes in a network environment by virtualizing network hardware [1]. SDN architecture has been adopted by many companies such as Google and Amazon [2], which shifted to SDN in order to create a network that is able to handle dynamic routing changes more efficiently while maintaining fault tolerance.

1.1 State of Software-defined Networking

This level of control adds flexibility to the infrastructure and enhances the network performance in high demand fields such as smart cities and connected autonomous vehicles that require a high quality of service while processing a large amount of network traffic in real-time. The ability to process more data while maintaining data integrity and quality of service is essential to the emergence of smart cities and autonomous vehicles. These fields need to maintain a quick reaction time to threats targeting the security in the network, thus leading to network outages. Adopting SDN will significantly decrease the reaction time along with allowing for a more dynamic approach to control the network [1]. Separating the control and data planes, adds a number of potential attack vectors and threats. Some of these attack

vectors targeting the data plane are inherited from the traditional network model, which consist of unauthorized access to the network or disruption the network performance such as a denial of service (DoS) attack. Attacks such as these can affect the network infrastructure where the network switch's ability to process traffic.

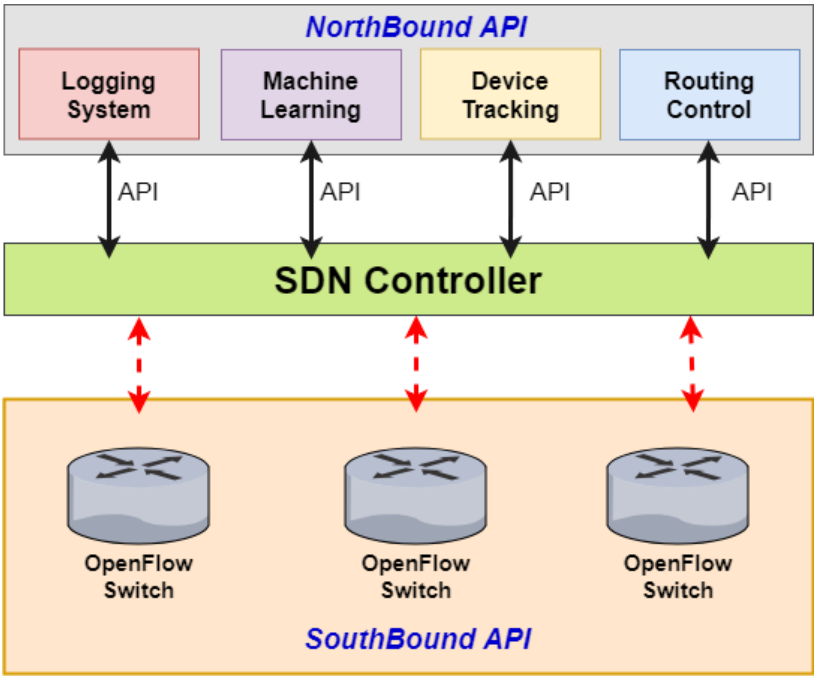

Figure 1.1 A diagram showing the general architecture and planes separation used in software-defined networking

Data plane attacks can also target the protocols themselves. Common southbound communication protocols such as OpenFlow (OF) [3] and Open vSwitch (OVS) [4] can be exploited by attackers to instantiate new flow rules in the network. Since these protocols are relatively new, they come with the growing pains of development. The leverage of these protocols can bring common attacks such as man in the middle (MITM) [5] and other spoofing attacks replay attacks [6].

The northbound and southbound APIs serve as bridges in the network architecture. This is how the control and data planes can work together in the SDN environment. The northbound API is how network applications are able to communicate with the SDN controller.

Applications that handle network functions such as logging, routing, and device tracking use this API. The southbound API is the main method for the SDN controller to assert changes to the network using the OpenFlow protocol on the network switches. These updates contain the logic that the network switches should follow when handling traffic.

The most attractive component of SDN that is highly targeted by attacks is the SDN controller. A simplified view of a software-defined network architecture is given in Fig. 1.1. Attackers can seek to spoof both northbound or southbound messages to gain control over the network. Spoofing southbound messages can lead to the instantiating of flows that allow attackers to bypass security measures in the network [6]. Northbound message also can be spoofed to bypass application layer security that may be present [7].

Along with these issues, the SDN controller can be targeted by DoS attacks, where a flood of Packet In and Packet Out messages can be forced onto the controller by attackers. If the controller is unable to process these messages in a reasonable time frame, it may prohibit legitimate traffic from being processed. Another point of note that the virtualization of the controller allows it to be run on a large variety of operating systems. This however causes the SDN controller to inherit any vulnerabilities that may be present on the operating system. Lastly, attackers have the ability to bring up a rogue controller. If this rogue controller is able to authenticate with the network switches it would then be able to push malicious flows into the network. Another common attack target would be the northbound application programming interface (API) that can be implemented by the SDN controller. These APIs can use a variety of tools such as Java, Python, REST, and JSON to carry out network tasks. The downside being that they are likely become exploitation targets. Unprotected or sparsely protected REST APIs can lead to attackers gaining network controls. All these issues need to be addressed to fully adopt the SDN paradigm for future use.

Current detection and mitigation methods implemented by companies are unable to scale to the amount of traffic that is being processed. Current research reports showed that the

average company takes 170 days to detect an advanced threat, 39 days to mitigate, and 43 days to recover, according to the Ponemon Institute and the average attacker persists on the network for up to 99 days before detection [8]. Software-defined networking offers a potential solution to these flaws by having a centralized control plane that lies separate from the data plane. This network paradigm allows for novel solutions to be created and spread throughout a network from a central point of control. By combining the idea of threat hunting with the functionality allowed by software-defined networking, a new novel solution for intrusion detection and mitigation can be created.

The often costly process of threat hunting requires the parsing of a high volume of network logs and records. These logs are often very noisy due to the logging of highly repetitive normal behavior on the network. Growing network traffic rates have led to the workload of parsing these logs to be tedious and require too much time to have an effective incident to mitigation turnover rate. The programmability of software-defined networks also allows for machine learning models to be trained based on normal network behavior that can then be used to make real time predictions on the overall integrity of the network. These models have a broader impact where they can be extended to other aspects of network management including; quality of service, load balancing, and network slicing. Once predictions are made on current traffic, new rules and updates can be made from the SDN controller to the network switches in order to mitigate threats on the network while upholding maximum network up time.

We, in this work, present an intelligent threat hunting system using the advantages of software-defined networking in conjunction with machine learning. The end goal is to offer a network infrastructure that better suits advancements in current technologies such as smart cities and autonomous vehicles. The adoption of threat hunting in the field of cyber security has lead to more systematic approach to finding new threats on internal networks that allows for a novel perspective on answering SDN security issues. The questions we seek to answer

with this research revolve around if there are ways to reduce threat detection and mitigation times, new ways of handling mitigation using SDN, and can automation be used to improve mitigation and detection.

CHAPTER 2

RELATED WORKS

Software-defined networking has gained ground since its introduction in the field of network architecture and design [9]. The use of southbound APIs can leverage a centralized network controller to dynamically implement network changes from a top-down view of the network as a whole. The gain in flexibility and programmability due to this change has allowed novel solutions to be created for common networking problems. Contrary to these advancements, many new threats and issues have been found in the software-defined networking paradigm. An example of this is new denial of service based attack to target the centralized control point of the network [10]. These new threats leverage the fact that the network has a single point of failure at the SDN controller. Along with these new attacks have come new security measures that leverage the flexibility of SDN to isolate affected targets from the network [11]. These solutions adapt common defense strategies for traditional networking to software-defined networks. Despite the fact that these strategies are effective, they also can affect the end-user quality of service. In [10] the authors proposed to limit the overall transfer rate dynamically to prevent denial of service attacks. The strategy does mitigate the threat, but it also has a negative impact on the quality of service for the entire network. In [11] a different approach is taken to neutralize an incoming attack. The strategy consists of isolating the control plane in an attempt to remove it from the attack surface. This defense strategy still leaves network switches exposed allowing for means of attacking the control plane.

The programmability of SDN also allows for opportunities in machine learning to be

applied to network-based problems. Current research has been done to implement machine learning solutions to design next generation networks. An example of this is the use of classification and feature extraction to build high-density WLANs using machine learning and SDN [12]. The idea behind this is to train network models to classify and extract features that will allow an ideal configuration of the network under high load. By taking advantage of SDN flexibility, these configurations and optimization can be applied dynamically throughout the network. Other research implements dynamic routing using neural networks and an implementation of Dijkstra's algorithm to improve quality of service for end users [13]. In essence it has a common strategy as the previously mentioned research to identify optimization in the network through machine learning. Although both studies show positive results in the combining of machine learning and SDN, they suffer from common machine learning problems and also some problems that are unique to SDN, such as data management and controller strain [14].

One issue with machine learning implementations in SDN is the high variety of data that is processed by a network. This creates issues when training accurate models in a SDN environment. Another side effect is testing the accuracy of these models can become different because of the inconsistency in the data sets collected. Targeting of machine learning models using adversarial training effects such situations as a Bayesian Network implemented in [15] that can cause predictions to become skewed in favor of hiding an attacker. In [16] a direct benefit of using gradient boosted trees to detect threats by allowing for a high degree of dimensions. They show that with this model in specific use cases can reach 99.8% accuracy. The downside of this model relies on high volumes of labeled data for the supervised learning process. We observed that this type of supervised learning is not applicable in real world networks that demand a large throughput of traffic. We found that a solution that allows for an unsupervised learning approach is more effective in offering a real world solution to detection.

2.1 Motivations

To solve aforementioned the issues found in this related work, we applied the ideas of threat hunting along with tested network logging methods to train and harden SDN machine learning models while also aiming for a shorter response time than usually attributed to threat hunting. The benefits and strengths of threat hunting can be leveraged to cover the weaknesses of traditional machine learning models [17] [18]. This in contrast to a more traditional approach to threat detection where signature-based systems are used to identify attackers by specific actions. The hands off data collection approach in threat hunting strengthens it against obfuscation methods used by attackers. The collected network behavior can then be used to identify infrequent behavior that can be flagged as suspicious. The downside of threat hunting is that it requires a large amount of time and labor to perform forensics on the logged data to both discover threats and add new functionality in logging to cover blind spots. Our work leverages SDN and machine learning in this regards to take the human out of the loop to increase threat mitigation response time while also allowing the network to self harden based on its needs. In comparison, our network will naturally become more resilient than the work in [15] by treating network data without bias to allow accurate training of our machine learning models.

2.2 Contributions

In this section we describe the contributions made by our research. These contributions focus on improving the overall efficiency and accuracy of threat detection. The following contributions are made in this research:

- Accurate Detection Model An accurate machine learning threat detection model using frequent pattern mining and gradient-boosted trees that is able to process unlabeled network data. This includes the ability to parse SDN-based events.

- SDN-based Mitigation A SDN-based mitigation system that is able to dynamically push network update events throughout affected parts of the network upon detection.
- Dynamic Logging Rule Creation A dynamic logging rule creation system that allows for the network to harden itself to future threats that share similar behavior to past threats to improve detection speed and accuracy.

CHAPTER 3

BACKGROUND

This chapter serves the purpose of outlining and describing technologies and methods used in this research. The following sections contain all relevant systems and definitions for our work. Each subsection contains a summary of the detailed system along with its implementation in our Intelligent Threat Hunting System (ITHS).

3.1 Definitions

The following definitions and terms are used throughout the work.

Definition 1 - Openflow switch is a software-based implementation of a hardware switch in a network. It often serves as a node to route traffic between hosts on a network.

Definition 2 - SDN controller is a software-based implementation of a hardware controller in a network. A controller often serves as a primary point of control that can designate work to network switches.

Definition 3 - Network link is a route between any two virtual switches that allows for the transmission of traffic.

Definition 4 - Flow is defined by the source host, the destination host and the requested bandwidth, which is defined by the amount of traffic being generated during the flow.

Definition 5 - Lateral movement is a strategy that abuses slow threat detection to spread across the network environment using mean that are often hard to detect through traditional intrusion detection methods.

3.2 Floodlight SDN Controller

Floodlight is an open source SDN controller developed by Big Switch Networks. It allows module based development to extend and create new features. Along with this it supports both physical and virtual Openflow switches [1].

In our work we use the Floodlight SDN controller that is developed in open source. Since the Floodlight controller is open source it allows for modification to integrate the ITHS. The controller is used in both topology and link discovery as well as device and host tracking. All network updates and management will be processed through the Floodlight controller to the Openflow switches on the network. Our ITHS will query the Floodlight controller to dynamically push network updates upon detection of a threat.

Floodlight also provides an environment where a multiple controller implementation can be used. This helps mitigate the single point of failure problem that SDN faces. Being implemented in Java also allows the controller to be hosted on a large variety of devices.

3.3 Openflow Protocol

The Openflow protocol is the standard protocol for software-defined networking [1]. It originally defined how the communication channel in SDN environments enables the controller to directly interact with the data plane of network devices. This allows the network to better adapt to any changing environment. Packet forwarding and other traffic management tasks are handled through flow tables maintained on each network switch by the

protocol. These flow tables contain all routing rules that can be utilizes by the switch when routing traffic. If no rule is present for a given packet the controller will then be queried to decide further routing. In order to detect threats to the Openflow Protocol we use custom logging rules to support the logging SDN network events such as flow table updates.

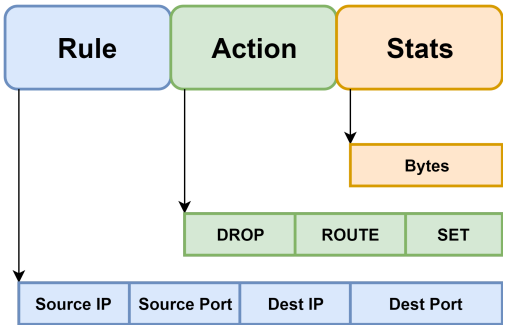

Figure 3.1 An example diagram of the structure of an Openflow protocol message

In order for a device to be used in a SDN environment it has to support the Openflow protocol. The Openflow protocol does come with new vulnerabilities and attacks unique to SDN that we also address in our work. These include attacks such as flow table poisoning and switch masquerading. The heart of these attacks lie in how updates are pushed to the Openflow protocol. Figure 3.1 shows an example of the Openflow update structure.

3.4 Kafka Streaming

Kafka Streaming is a framework for data handling across a network [19]. Kafka accomplishes this by establishing a shared data stream where different systems can both produce and consume data within the stream. Along with this, Kafka has the capability to timestamp data and track changes to the data in the stream. We use Kafka Streaming as a data stream for all logging and SDN Response system updates. This allows for the centralization of data to create a data set for the network as a whole as well as individual partitions. We also gain the ability to visualize our system through the use of an Elastic Search, Logstash

and Kibana Stack (ELK) in conjunction with Kafka. An ELK Stack combines Elastic Search, Logstash, and Kibana to visualize data streams and historic data. Both of these are useful when tuning our system for better results.

Kafka also serves as a long term log processing system. This allows us to store previous machine learning models and data sets. These data sets can be useful when computing the decay rate of the accuracy of our system.

3.5 Bro IDS

The Bro IDS is used as the logging framework for our system [20]. Bro serves as a logging system that logs common network events that occur. It does not distinguish between malicious and non malicious events. Instead, Bro treats all events equal allowing for it to be a more complete set of logs of a network environment. It allows for custom scripting to support the Openflow protocol and dynamic rule creation. We developed custom scripts to detect SDN network events such as flow table updates, controller connections, and switch connections. Both the data and control planes will be tracked by Bro. With this we can track network events on both the control and data planes.

We are also able to partition the logs based on the nature of the log. In this way we can separate differing network events such as connection logs and DNS. We can then train separate models for each log type. We can then test the accuracy of the partitioned models in comparison to a general overarching model for all network event types.

3.6 Spark

Spark is a machine learning framework that natively supports Kafka Streaming [21]. It includes standard libraries for common machine learning algorithms that can be modified

and integrated with custom functionality. In our case, we integrate Spark to act on the data we store in the Kafka data stream. With Spark we develop the frequent pattern mining and gradient boosted tree models that will be used to determine if new network events are suspicious. Spark also allows for us to store and retrain models on demand. This will ensure that the models are trained on the most current state of the network. Along with this we can test the accuracy of our models using a couple different methods. The first method is by pre-labeling data and then comparing it to the trained model labeling. The second being calculating the test error of each run through using out-of-bag observations. Out-of-bag observations use sampling to determine the error rate in each training sample.

CHAPTER 4

SYSTEM ARCHITECTURE

In this chapter we discuss the overall system architecture for our research. This includes major components such as SDN and threat hunting. The ending section also details the physical test bed used to analyze the performance of our systems.

4.1 Software-defined Networking

Software-defined networking refers to a network architecture where the forwarding plane is separated from the data plane in the network. This is combined with three other core features can be used to describe a SDN [3]. The first core feature as explained above is that the data plane and the control plane are separated from each other in the network architecture. This differs from traditional networking architecture where the network devices control both routing decisions and actual packet routing. The second major feature of an SDN is that routing decisions are flow-based. A flow is defined by the source host, the destination host and the requested bandwidth, which is defined by the amount of traffic being generated during the flow. Packets in a flow all receive the same policies allowing for the abstraction of the flow which gains flexibility in control of the network. These flows are then stored on a flow table which is hosted on each Openflow switch.

The third core feature of a SDN is that control logic is moved to a external device on the network. This is the heart of SDN architecture. By separating the control logic from the routing devices a more advanced control will become available on the network. This feature

allows for the fourth major feature in SDN which is that the network becomes programmable. By implementing applications that interface with the external network control point the SDN can be controlled and adapted based on software. This centralized control point allows for the abstraction of the topology as a whole which lets programming for the network become a simple task. These four major features help define the software-defined network architecture.

Another benefit of having a centralized control point is that it becomes easier to push network policy updates without producing errors using high-level languages. Policies can also be maintained on the fly based on network changes. A global view is also gained by using a SDN architecture. This view makes programming for the network easier since applications can share the same network data. The same applications can also push network policy updates from anywhere on the network as long as they can access the SDN controller. Applications can work in sync such as load-balancing, routing, and other applications by introducing a priority system where higher priority has precedence over lower priority applications on the network.

Another concept that exists in SDN is the Northbound and Southbound APIs. The Northbound API gives network applications a more general interface to use when developing for the network. This API exists between the SDN controller and network application. The Southbound API serves as an interface between the SDN controller and the Openflow switches on the network. This gives a formal language for interaction between the control and data planes. SDN has also brought the idea of network virtualization. SDN naturally melds with network virtualization with the separation of data and control planes and allows for multiple SDN switches to share the same physical resource. Both flexibility and portability is gained from this method. To summarize, SDN allows for more flexibility and programmability in network architecture that allows us to leverage these tools for advancements in security.

4.1.1 Controller Functionality

The SDN controller is routinely explained as a network device that serves the basic functionality of a network. This includes functions such as routing, topology control, statistics, and device management [4]. On top of these basic functions more advanced functionality such as security measures and other critical components. To summarize, the SDN controller should act as an operating system for the network as a whole. All network decisions and management should be handled by the controller.

4.2 Threat Hunting

Threat hunting is the act of preemptively performing forensics on a network to identify threats. In order to achieve this, anomalous behavior should be identified and then further inspected. In contrast to signature-based detection systems, where a security team waits for a signal to be triggered based on an attackers actions on the network. Threat hunting was introduced to solve a problem in network security. The problem was that, on average, attackers were spending 99 days on the network undetected while only needing less than three days to obtain administrative credentials [8]. Along with this on average 53 percent of these attacks are only detected because of signalling by a external third-party source.

The general rule of threat hunting is to log any and all network events for future analysis. This gives forensic experts more to work with when trying to discover behavior patterns that may be considered malicious. This comes with the flaw that the amount of data becomes cumbersome and hard to manage. In environments such as smart cities where the amount of traffic may be too much for a single security team, other solutions are needed. Threat hunting's answer to this has been through developing machine learning models that can label data on the network vastly reducing the threat window that needs to be analyzed. Classifiers such as gradient-boosted trees have been used with good result in identifying

anomalous behavior in network traffic [16].

In the end threat hunting is a passive system that solely relies on forensics to identify threats. It also leaves any mitigation to the network operators. This can lead to a similar problem as before where attackers have free reign on the network for a large amount of time. Another work reports that the average company takes 170 days to detect, 39 days to mitigate, and 43 days to recover from a network attack [8]. Our work seeks to not only use machine learning and threat hunting to cut down on detection time, but it also actively mitigates the attack using software-defined networking. This gives a more encompassing security solution that can be used to benefit security teams of large networks such as smart cities.

Insight can also be gained from the logs generated through threat hunting. These can include inferences on potential future vulnerabilities and network bottlenecks. Having a system that models behavior of the network can lead to further developments even outside of security.

4.3 Logging System

The purpose of threat hunting is to create a system that can more easily detect advanced network attacks that are hard to track using traditional methods. An example of this kind of attack is lateral network movement. Lateral movement on a network allows an attacker to pivot to easily accessible devices to gain persistence on the network. Since this movement does not involve further exploitation or privilege escalation traditional signature-based detection struggles to track lateral movement on a network. Threat hunting solves this issue by taking a more verbose logging approach. In threat hunting, network events are not treated as malicious or non-malicious, instead they are all logged equally. The goal is to allow forensic investigators and systems to make insightful observations of the logs

to discover advanced network threats. Although this process does allow the detection of advanced network threats, it does not scale with larger networks. Larger networks such as smart cities and enterprise networks produce an exponentially larger amount of traffic which in turn increases the workload and time it takes to mitigate. To improve this, machine learning can be adapted to lessen the investigation window for each case. This can be done using common machine learning models that will be discussed in further detail in the following sections.

4.4 Intelligent Threat Hunting

The SDN architecture supports flexibility in the form of programmability of the control plane. Using this flexibility, we create a network architecture that uses the behavior modeling of machine learning with dynamic network updating. Fig. 4.1 outlines the general layout of our work. The supporting systems of our work include a network logging system that collects network events, a machine learning system consisting of both frequency pattern mining and gradient-boosted tree models, and finally a response system that leverages the SDN environment to push changes. These four pieces create a continuous cycle that can monitor the network.

4.4.1 Logging System

The backbone of threat hunting is the logging of any and every network occurrence. For this we use the Bro Intrusion Detection System in conjunction with SDN devices. In our network each openflow switch and SDN controller hosts a Bro client which logs network events and sends them to a centralized repository on the primary SDN controller. These logs contains events from both the control and data planes allowing us to track attacks.

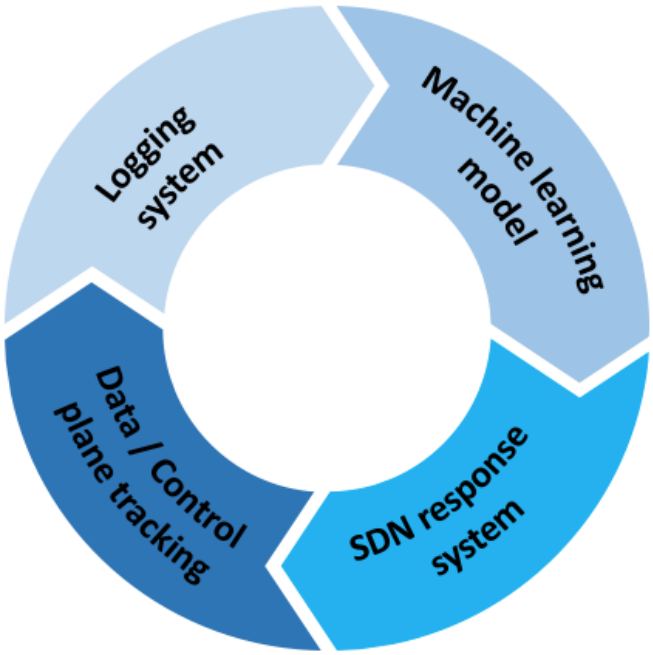

Figure 4.1 A diagram detailing the four stage layout of the ITHS framework

4.4.2 Machine Learning Model: Frequent Pattern Mining

The Frequent Pattern Mining model (FPM) is the first stage of the machine learning system. For input, the model is trained on unlabeled network data from our logging system. The data is considered unlabeled until each record can be flagged as frequent or infrequent by the FPM model. The FPM model is trained based on the frequency of events occurring on the network. This frequency includes the occurrence of items in an event such as IP addresses or action types along with how often they occur with each other. Once the model is trained, we can use it to label the network events as frequent or infrequent to serve as input for the gradient-boosted tree model. Infrequent items that are flagged in this model can be treated as suspicious according to a threshold that is based on the ratio of normal traffic on the network. The threshold is computed using the total amount of traffic on the network in comparison with the frequency of the relationship between features. Unlike signature-based detection [22], a single event does not constitute malicious activity. Instead a grouping

of these behaviors gathered by the SDN Response system can be used to further classify malicious behavior. Following this, a threat hunting approach will enable the flexibility in detecting complex network threats.

4.4.3 Machine Learning Model: Gradient-Boosted Tree

To predict future classifications of network events, we use a Gradient-Boosted Tree model (GBT) [23]. This model combines a large group of weak decision trees to create a stronger prediction model. Each node represents a state where each feature can be in. Based on the historical network data, the decision trees can be constructed to predict the outcome of new incoming events. This model is useful when ranking the network events based on the probability of being suspicious to the network. The input to train this model is the labeled data set created by the FPM model. Once trained, new unlabeled data can be taken as input by the model to make an accurate prediction on whether the event is suspicious or not. The output of this model will include the subset of events that were flagged as suspicious that will then be formatted and sent as messages to the SDN Response System.

4.5 SDN Response System

The SDN Response System (SRS) receives messages from the GBT model and builds a queue of actions to be taken on the network such as routing control, traffic dropping, or blacklisting. This system will track devices on the network, marking frequent devices as trusted based on historical data from the machine learning models. Events that have been flagged as suspicious will be mined for important information such as affected devices and actions taken. From this data, we can build network update messages in the form of SDN flow tables updates. With the SDN flow updates, we have the ability to facilitate network

flows, blacklisting, forwarding, and other useful actions to maintain quality of service in the network. Severity of each event can be determined by the scale of the network affected along with if any critical devices were involved such as network controllers.

Once actions have been taken, the SRS system will also update the logging system with new rules for behavior. These new rules will attempt to log affected devices more verbosely in order to target previously threatened devices. This step is taken in order to harden the network to future attacks using the same methods.

Our proposed system consists of two models that come together to create an intelligent detection and mitigation system that functions on the network automatically. The first model is shown in Fig. 4.2 which involves training a machine learning model to detect changes in the network environment along with suspicious activity that occurs on the network. To achieve this we use a frequent pattern mining (FPM) model that is trained on common network attributes and occurrences. Frequent pattern mining is a useful algorithm for determining if a new event can be considered a normal occurrence or not.

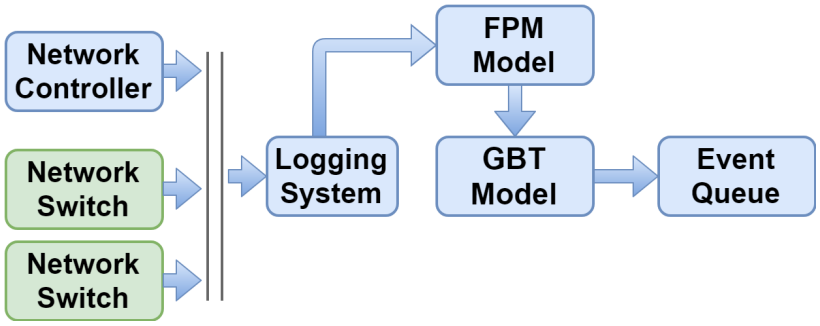

Figure 4.2 The flow of events when training the ITHS machine learning models

In our case we are training this model to build an item set of frequent items that can be used to compare in real time against new network events. Adapting this to software-defined networking also involves making the distinction between control plane traffic and data plane traffic. In SDN these two types of traffic always happen on a separate interface isolated from each other. With this, we can better control data classification and also identifying access

to the control plane coming from external sources. This model will return a labeled data set that classifies data based on its occurrence. This can then be used to train a gradient-boosted decision tree model to predict the label of future traffic logs. The two machine learning models can then be evaluated based on their accuracy over time by calculating the decay rate. The decay rate is defined as the rate in which accuracy is lost due to a new normal network behavior being established over time. Once a unacceptable level of inaccuracy is reached based the nature of how critical the network is, the model will then need to be retrained to form a new baseline.

The second model is shown in Fig. 4.3 which involves incorporating this newly trained models into the software-defined networking environment. New network traffic will produce logs of events occurring on the network. These logs are then fit to the gradient-based decision tree model that was trained on normal network behavior for the control plane and data plane. The trained models will compute a probability that an event can be considered suspicious.

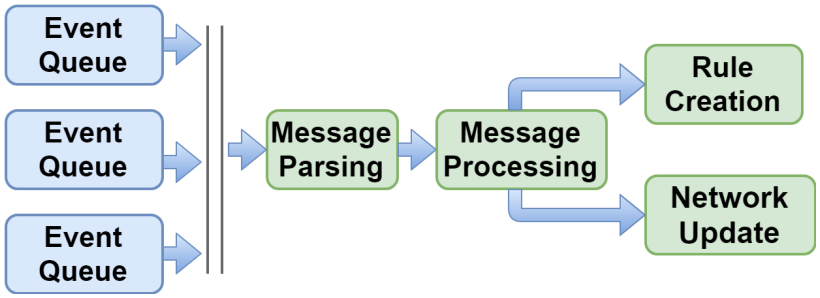

Figure 4.3 The flow of events when updating the software-defined network when an event is detected

By setting a threshold for the value we can determine when action is needed for a given event. This threshold is computed by creating a ratio of how often the event has occurred when compared to other events. Once the threshold is triggered, we start building a network update event in order to mitigate the action being taken against the network. This event is composed of any useful data that can be ascertained for the log events that triggered as suspicious. This event is then queued with the software-defined network controller and

awaits processing. The software-defined network controller then completes two tasks for each network update event message. The first task is mitigating the detected threat on the network. Based on what the event was that triggered the model, the controller will take actions to update any switch flow tables, drop affected traffic, or blacklist affected hosts. Since software-defined networking allows for centralized updating of the network, we can quickly and efficiently adapt the network to prevent further threats. The second task the controller takes is to create new rules that specifically targets the type of behavior that triggered the event. This makes future detection of similar threats take less processing time to prevent and mitigate attacks.

4.6 Network Test Bed

In order to test the intelligent threat hunting system we implemented a network test bed using Raspberry Pi computers. This allowed for an accurate experimentation of the performance and accuracy of our system. We can separate our test bed into two components based on SDN architecture. The first component is the control plane. As detailed before, the control plane hosts our SDN Floodlight controllers which are ran on Raspberry Pi 3s. These controllers also will perform all data handling and machine learning and passing the resulting actions to the second component of the network, the data plane. The data plane consisted of both inner switches and outer switches on the network. Inner switches are devices that only contain switch to switch connections while outer switches contain switch to host connections. All network switches are hosted on Raspberry Pi 3s using USB to gigabit adapters for link connections.

Hosts on our test bed consisted of both regular PCs and Raspberry Pi computers. Hosts performed a variety of common network tasks to create a baseline for traffic analysis. During testing, we introduced devices that will act as malicious on the network.

CHAPTER 5

RESULTS AND ANALYSIS

In this chapter we outline a use case scenario for our system that steps through the detection and mitigation process. Along with this we analyze and gather performance metrics and compare them to recent research in the field.

5.1 ITHS Example

Below we present a use case for our system. This use case outlines the general flow between detection and mitigation. Each section will further detail the steps taken and provide insight in to possible benefits and flaws. Figure 5.1 outlines an example network used in testing. The C label denotes a network controller while a S label denotes and Openflow switch. We will walk through how our system manages the network during the occurrence of a network compromise.

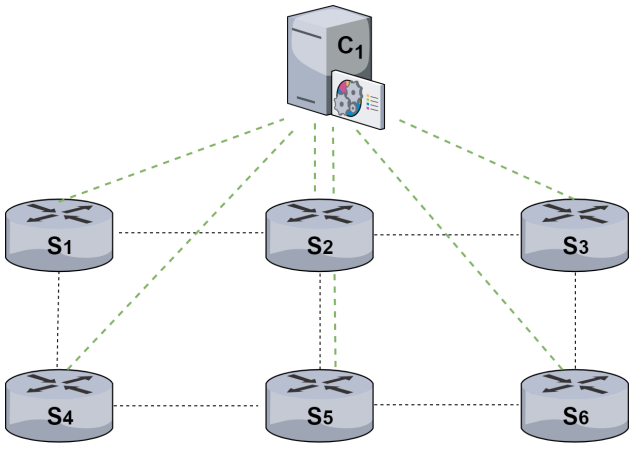

Figure 5.1 Our network topology used during testing of the ITHS system

5.1.1 Log Collection

The first step in the Intelligent Threat Hunting System is to deploy a logging framework to all available Openflow switches and network controllers. These serve as workers to report on any network event that occurs. This data is important in building an accurate behavior base to train the machine learning models. A log is created on any network event, and it includes any relevant event data.

In order to capture Openflow events such as flow table updates, we created a custom rule script to recognize the protocol. SDN-based threats such as flow table poisoning and controller-based attacks can be mitigated in an efficient manner by parsing the Openflow protocol.

Once all the logging systems are in place, network devices push their messages to a single shared data stream on a set window interval. Messages can then be identified with a tag indicating which network device the message originated from. This window allows for the network as a whole to be processed giving a more complete picture during detection and mitigation.

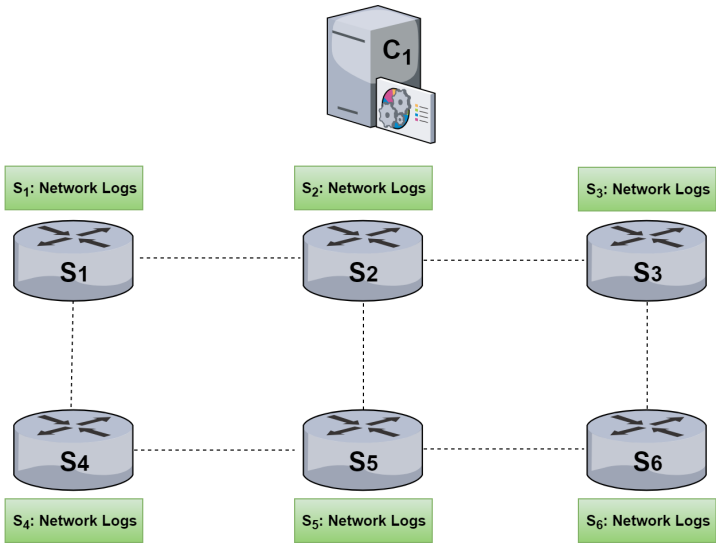

Figure 5.2 How logs are collected on the network and passed between devices

5.1.2 Data Processing

Once logs are in the centralized data stream they are processed for feature extraction. Identifying fields such as log type and switch network ID are separated from data fields. The data fields are then used to build feature sets. The created feature sets are then used to train a frequent pattern mining model. This is useful in labelling the initially unlabeled data set based on frequency of occurrence on the network. We can assume that infrequent events on a network can be viewed as suspicious with a sufficient amount of data based on network size.

The resulting feature vector from the FPM model is used to train the next machine learning model. The FPM model is also associated with a decay factor in the network. This decay factor is defined as the rate at which the FPM model becomes inaccurate when labeling the network data. For this and based on how critical the network is, a threshold can be set to signal the model to be retrained. Each training for the network model should be completed on new data to ensure that an accurate account of event frequency can be gained.

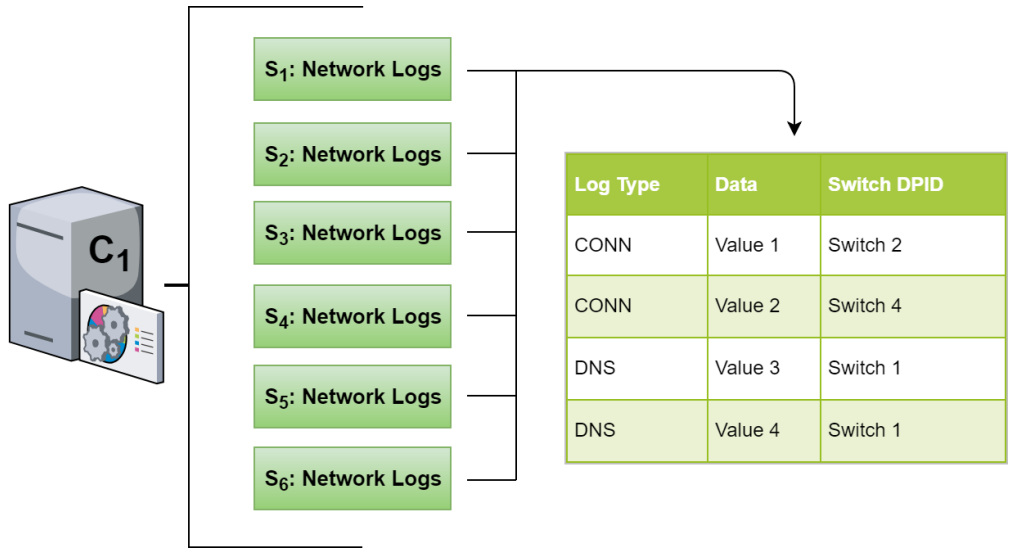

Figure 5.3 How collected logs are processed on the central device once received

5.1.3 Behavior Modeling

The GBT is trained using the newly labeled data from the FPM model. The input for this model is the feature vector that was previously created with the appended label of being frequent or infrequent. Once this model is trained on the FPM data set, it can then be used to label new network events. Each new network event is then passed to the GBT model which will append a prediction and probability. The prediction is the frequency label based on the similarities in the feature set attached to the event. The probability is based on the level of similarity between the GBT model and the network event being labeled.

The GBT model has a similar decay rate based on how much normal network behavior changes. Similar to the FPM model, based on the how critical the nature of the network is, would determine how often the model should be retrained. In the case of smart cities, critical infrastructure will be hosted on the network such as utilities and vehicle communications. A network with such importance demands a higher accuracy rating. The next step in the process is to build a network update event based on the findings of the machine learning models. These update events are based on affected devices, threat scope, and rate of occurrence. Affected devices are any network device such as switches and controllers that were the origin on the produced network log. This is also used to define threat scope which determines how much of the network is at risk. Risk is determined by the number of devices affected and the importance of the device to the network such as the network controller.

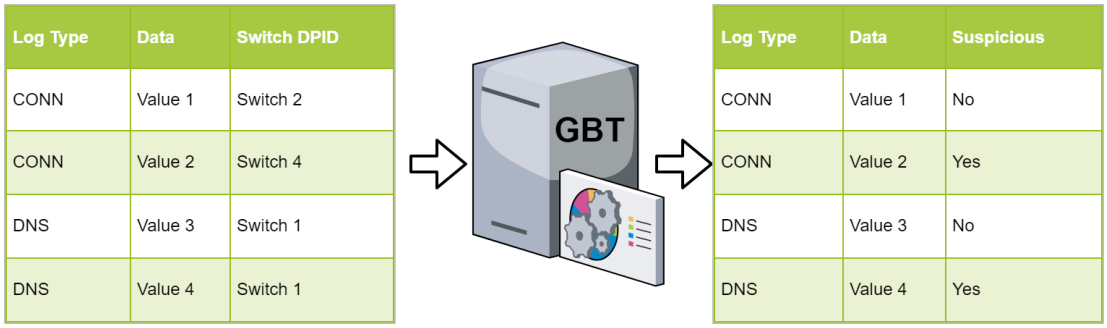

Figure 5.4 A diagram detailing the before and after data sets after classification using GBT model

5.1.4 Event Processing

The next step marks the end of the detection phase and the move to the mitigation phase. In order to begin mitigation, data mining must be preformed to extract relevant data that will be useful during mitigation. Unlike signature-based detection, where strict actions can be taken on detection, the events that are flagged as suspicious cannot be treated as malicious. Instead a risk value is computed by the assumed risk of each detection. This value is computed based on the amount of the network it affects and the potential impact of the network itself. An example being that threats that involved a master controller will be treated more strictly than that affecting a single switch. This also pushes the threat hunting idea that the system is for mitigation and the finding of threats on the network before they escalate.

In Figure 5.5 below we see an example of a network update message. The first set of fields detail the accused devices found in the threat along with the action that the SDN controller should take. The second set of fields details where the SRS should send the updates to. The idea is to only update network devices that are affected by the threat to limit strain put on the SDN controller. Once the affected controllers receive the SRS message they can begin to send updates to all affected network devices using the functionality allowed by SDN.

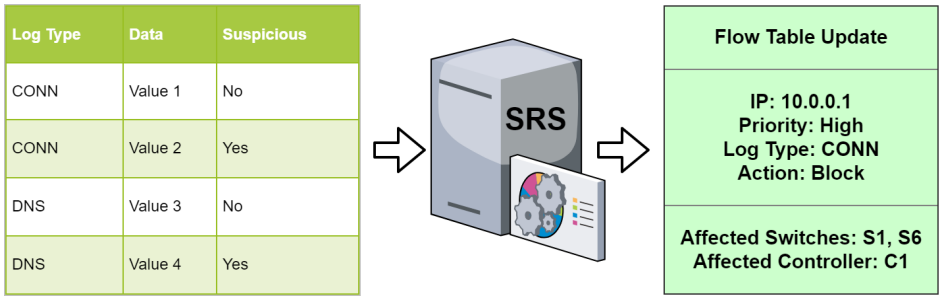

Figure 5.5 How a network event is processed and prepared for distribution throughout the network

5.1.5 SRS Mitigation

The final step of the mitigation phase is to implement the network updates received from the SRS. Each affected switch will receive from the controller a flow table update message that is crafted from the SRS update message. Before updating the flow table, previous flows will be cleared to ensure that any malicious flow table entries are removed. Once removed, flagged host devices will have the decided action taken against them. In the example, Figure 5.5, we see that the host at IP 10.0.0.1 will be blocked from the network. These actions can range from limiting bandwidth on a link from a given source to blacklisting the source of the threat. These actions are meant to slow the lateral movement and spread of a malicious host on the network.

Another function that is provided by SDN is the ability to quarantine when available. For example, the network can seek to quarantine a hosts scope by routing traffic through a specified slice of the network. This in turn prevents the host from performing lateral movement across previously unaffected devices. This completes the mitigation phase and transitions to the dynamic rule creation phase.

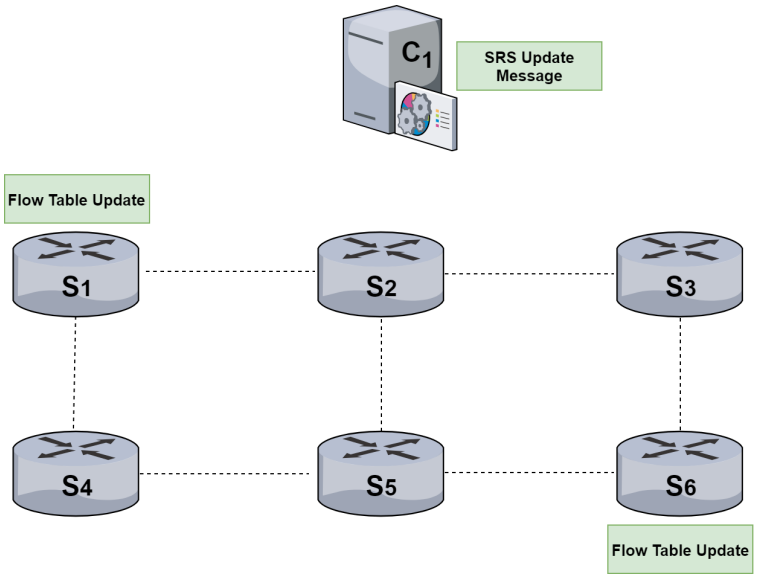

Figure 5.6 How network events are passed throughout the network once an event has been processed

5.1.6 Dynamic Rule Creation

To prevent loss of detection rate when retraining on new data, the SRS also pushes new rule updates to the logging system. This allows previously detected threats to more easily be identified on the network during the logging phase. This behaviour becomes similar to a signature-based system. The new rule contains the SDN action taken on detection and also the feature set used in detection.

This system also removes the same threat being acted upon multiple times by the SRS. The process gains a natural speedup by using previously gained results to skip the machine learning models if the detection can be made by the logging system. Another benefit of updating the logging system is to potentially find similar behavior on the network across different host devices. Previously undetected hosts can then be identified that share the same suspicious behavior but were missed by the behavioral models. This creates a cycle that will continue processing data until it is retrained.

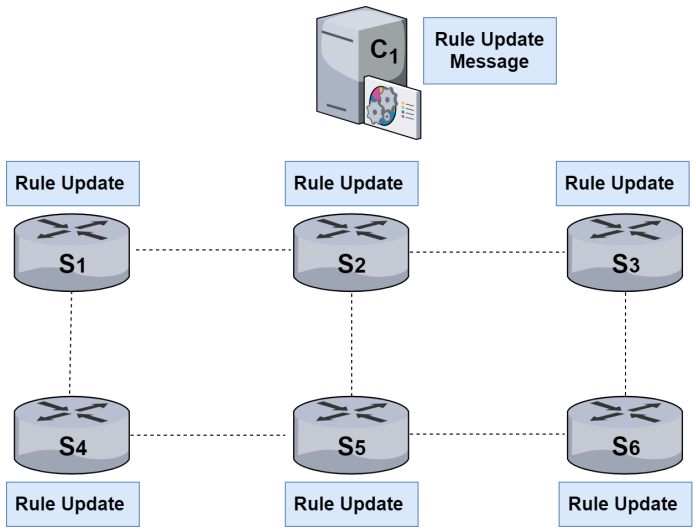

Figure 5.7 How new dynamic rules are created and distributed throughout the network devices

5.2 Numerical Results

In this section we compute the performance metrics used to measure our machine learning models. These metrics include accuracy, specificity, sensitivity and training time. True positive, true negative, false positive, and false negative are used in the computations. Accuracy is computed based on the proportion of the total number of predictions that were performed correctly. Specificity is the proportion of predicted negative cases that were correctly identified by the model. Finally sensitivity is the proportion of the positive predicted samples that are correctly identified by the model.

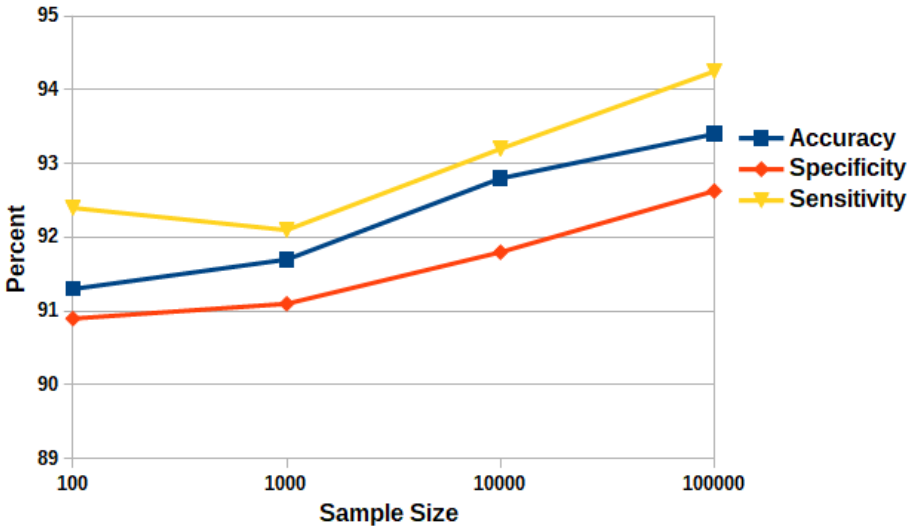

Figure 5.8 A graph detailing the performance metrics of our GBT model across four different sample sizes

Along with these metrics the training time for four data sets is documented for 100, 1000, 10,000, and 100,000 records. To gauge the performance of our model against current standards we make the comparison with current research that implements a GBT classifier for intrusion detection [16]. This GBT classifier is used to determine malicious nodes based on previously labeled data sets. Table 5.1 shows our comparison.

Table 5.1 A Comparison Between the Performance Metrics of Our ITHS System and a GBT Model Using Labeled Data

Algorithm	**Accuracy %**	**Sensitivity %**	**Specificity %**
Intelligent Threat Hunting	**93.4**	**93.43**	**92.63**
GBT IDS	**99.8**	**99.83**	**99.78**

As seen, our model is slightly less accurate, but remains competitive in all categories. The discrepancy in the metrics has to do with the data set that is processed by both algorithms. The GBT IDS research uses a preprocessed data set with malicious data already labeled. We believe that this is not feasible in actual implementation due to the amount of data that needs to be labeled. In comparison, we implement a FPM model to label our data set for use in our GBT model. This allows network data to make a detection on unlabeled data sets. This both improves efficiency and speed of detection for our model. We observed an upward curve in all metrics with an increase in data set size. From this we can argue that given a large enough data set that our accuracy metric will reach that of other GBT classifiers.

Our next performance metrics we analyze are those attached to the mitigation phase of our system. We compare our metrics with that of Fire Eye's annual report on global security environment [8]. This report details global averages on detection times within businesses and networks. They report on average that a network threat has 101 days of dwell time on a network.

Dwell time is defined as the time from the point of exploitation up until detection. The range of these attacks in report include common exploitation techniques such as denial of service [22], man in the middle [5], and replay attacks [6]. Our system drastically cuts down on this dwell time. Using the functionality of SDN to push out network updates and collect flow data it is able to react to a threat at the moment of detection. This allows for mitigation to begin on our network in under 9 seconds on average with a data set 100,000 records. In

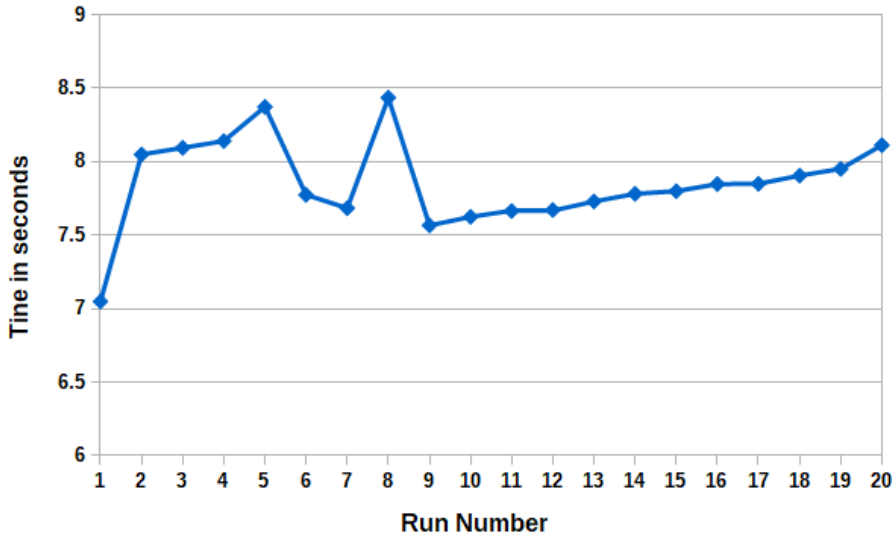

Figure 5.9 A graph of the detection to mitigation time across 20 runs of our system

Figure 5.9 we can see that across 20 runs of our system that the mitigation time remains under 9 seconds on average. A run is consists of set predetermined network traffic that is propagated throughout the network.

Along with the benefit of beginning mitigation immediately without human interaction, the output of the system can be further analyzed using network forensics. Compared to a normal threat hunting environment which can overwhelm a business due to the amount of network records; Our system offers a greatly reduced set of records that focuses the attention on suspicious behavior. The dynamic rule creation system also reduced detection time on future runs when behavior has already been identified in past runs. In Figure 5.10 we see a steady decrease in detection to mitigation time when using the dynamic rule creation on second set of 20 test runs.

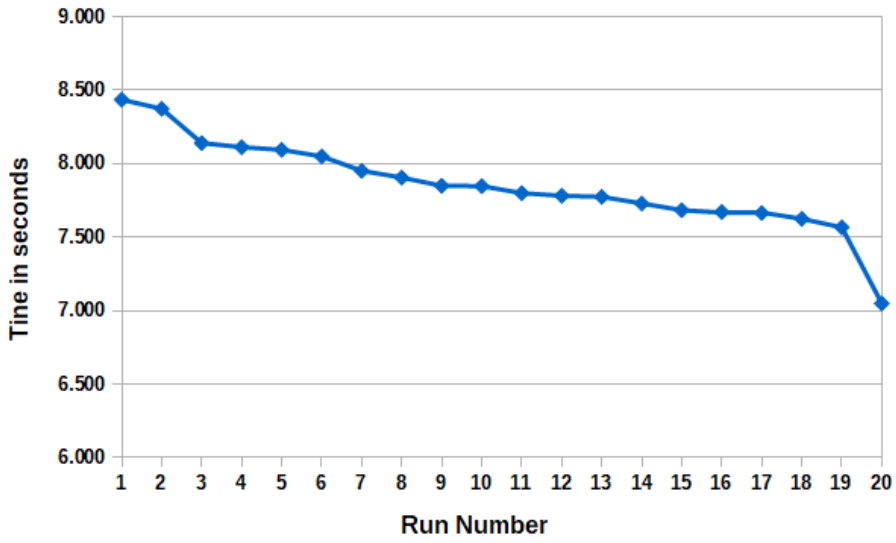

Figure 5.10 A graph of the detection to mitigation time across 20 runs when using our dynamic rule creation scheme

5.3 Lateral Movement Detection

The main purpose of our system was developed for the detection and mitigation of advanced threats such as lateral movement on a network. During our tests malicious actions consisted of simulated lateral movement that would be performed by an attacker. These actions are based on common actions found in threat hunting research. Our system identified 100 percent of the lateral movement, but also falsely labeled additional network actions. This leads to a lower accuracy rating long term. We believe this is due to shared features between normal and malicious behavior sets. In order to decrease this false positive rate, the features selected would need to be further refined to identify solely suspicious behavior. These false positives also are reduced on larger data sets. This is due to the behavior model becoming more refined with larger sets of data. Based on these results we can confidently state our system delivers on the three contributions we state of performing accurate threat detection and SDN-based mitigation.

CHAPTER 6

CONCLUSION

In this chapter we discuss the future work that can be accomplished by building off of our system. We also discuss the benefits of our system and how it can be used to increase current security standards.

6.1 Future Work

Based on our results, we can see that our system offers a novel solution to detection and mitigation using SDN that remains competitive. Further work can be done to research feature sets in network traffic that allow for better sensitivity when labeling suspicious behavior. SDN can also be further utilized to combine load-balancing schemes with the mitigation phase to better handle denial-of-service based attacks. We believe the system serves as a foundation for further development in threat hunting on SDN.

6.2 Conclusion

Advancements in fields such as smart cities and autonomous vehicles will require new infrastructural designs that allow the network to handle new traffic demands more efficiently. Our Intelligent threat hunting system fits the need through facilitating the handling of large throughput data while maintaining integrity and quality of service. In contrast, the traditional threat hunting models are based on performing forensics on large data sets by hand

which does not scale for larger networks. Our work accomplishes the goals of providing a high accuracy threat hunting detection model, automating SDN-based mitigation, and network hardening through dynamic rule creation.

Our system provides an answer to current networking security issues that are being found in adapting to smart cities and autonomous vehicles. We offer an intelligent threat hunting system that allows for dynamic updating of network devices to better serve users of the network. The system takes a necessary step to advance network architecture to adapt to future threats and problems.

REFERENCES

[1] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.

[2] J. H. Cox, J. Chung, S. Donovan, J. Ivey, R. J. Clark, G. Riley, and H. L. Owen, "Advancing software-defined networks: A survey," *IEEE Access*, vol. 5, pp. 487–526, 2017.

[3] W. Wassapon, P. Uthayopas, C. Chantrapornchai, and K. Ichikawa, "Real-time monitoring and visualization software for openflow network," in *2017 15th International Conference on ICT and Knowledge Engineering (ICT KE)*, Nov 2017, pp. 1–5.

[4] M. S. Olimjonovich, "Software defined networking: Management of network resources and data flow," in *2016 International Conference on Information Science and Communications Technologies (ICISCT)*, Nov 2016, pp. 1–3.

[5] M. Conti, N. Dragoni, and V. Lesyk, "A survey of man in the middle attacks," *IEEE Communications Surveys Tutorials*, vol. 18, no. 3, pp. 2027–2051, 2016.

[6] P. Syverson, "A taxonomy of replay attacks [cryptographic protocols]," in *Proceedings The Computer Security Foundations Workshop VII*, June 1994, pp. 187–191.

[7] D. M. Kristian Slavov and M. Pourzandi, "White paper: Identifying and addressing the vulnerabilities and security issues of sdn," Ericsson, Ericsson SE-164 83 Stockholm, Sweden, Tech. Rep., 2015.

[8] Madiant, "M-trends 2018]," in *Fire Eye Special Report*, May 2018, pp. 4–10.

[9] W. Braun and M. Menth, "Software-defined networking using openflow: Protocols, applications and architectural design choices," *Future Internet*, vol. 6, no. 2, pp. 302–336, 2014.

[10] S. Singh, R. A. Khan, and A. Agrawal, "Prevention mechanism for infrastructure based denial-of-service attack over software defined network," in *International Conference on Computing, Communication Automation*, May 2015, pp. 348–353.

[11] T. Sasaki, A. Perrig, and D. E. Asoni, "Control-plane isolation and recovery for a secure SDN architecture," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, June 2016, pp. 459–464.

[12] Á. López-Raventós, F. Wilhelmi, S. Barrachina-Muñoz, and B. Bellalta, "Machine learning and software defined networks for high-density wlans," *CoRR*, vol. 3 abs/1804.05534, 2018.

[13] A. Azzouni, R. Boutaba, and G. Pujolle, "Neuroute: Predictive dynamic routing for software-defined networks," *CoRR*, vol. abs/1709.06002, 2017. [Online]. Available: http://arxiv.org/abs/1709.06002

[14] T. N. Nguyen, "The challenges in SDN/ML based network security: A Survey," *CoRR*, vol. 7 abs/1804.03539, 2018. [Online]. Available: http://arxiv.org/abs/1804.03539

[15] S. Nanda, F. Zafari, C. DeCusatis, E. Wedaa, and B. Yang, "Predicting network attack patterns in sdn using machine learning approach," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2016, pp. 167–172.

[16] K. G. P. Gulati, "Intrusion detetction system using gradient boosted trees for vanet]," vol. 5, August 2017, pp. 187–191.

[17] Sqrrl, "White paper: A framework for cyber threat hunting," Sqrrl: Target. Hunt. Disrupt., Tech. Rep., 2016.

[18] R. Bejtlich, "Your threat hunting knowledge compendium]," in *Huntpedia*, August 2017.

[19] G. P. T. Narkhede, Neha. Shapira, in *Kafka - the Definitive Guide: Real-Time Data and Stream Processing at Scale*. O'Reilly Media, 2017.

[20] T. B. Project, in *Bro Documentation*, 2016, https://www.bro.org/documentation/.

[21] I. Foster, in *Big Data and Social Science: a Practical Guide to Methods and Tools.* CRC Press/Taylor Francis Group, 2017.

[22] CERT-UK, "White paper: Denial of service attacks: what you need to know," CERT-UK, Tech. Rep., 2004.

[23] S. Si, H. Zhang, S. S. Keerthi, D. Mahajan, I. S. Dhillon, and C.-J. Hsieh, "Gradient boosted decision trees for high dimensional sparse output," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 3182–3190. [Online]. Available: http://proceedings.mlr.press/v70/si17a.html

VITA

Steven Schmitt was born and raised in Nashville, Tennessee. Later, he moved to Chattanooga, Tennessee to pursue a Bachelor's degree in Computer Science at The University of Tennessee at Chattanooga. After receiving his Bachelor's degree, he was accepted into the Master's program in Computer Science at UTC. During this time he became a SFS scholar and published his research with plans on graduating December 2018.