Analysis and Management of Security State for Large-Scale Data Center Networks

by

Abdulhakim Sabur

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2018 by the
Graduate Supervisory Committee:

Dijinag Huang, Chair
Yanchao Zhang
Paulo Shakarian

ARIZONA STATE UNIVERSITY

December 2018

ABSTRACT

With the increasing complexity of computing systems and the rise in the number of risks and vulnerabilities, it is necessary to provide a scalable security situation awareness tool to assist the system administrator in protecting the critical assets, as well as managing the security state of the system. There are many methods to provide security states' analysis and management. For instance, by using a Firewall to manage the security state, and/or a graphical analysis tools such as attack graphs for analysis.

Attack Graphs are powerful graphical security analysis tools as they provide a visual representation of all possible attack scenarios that an attacker may take to exploit system vulnerabilities.The attack graph's scalability, however, is a major concern for enumerating all possible attack scenarios as it is considered an NP-complete problem. There have been many research work trying to come up with a scalable solution for the attack graph. Nevertheless, non-practical attack graph based solutions have been used in practice for realtime security analysis.

In this thesis, a new framework, namely 3S (Scalable Security Sates) analysis frameworkis proposed, which present a new approach of utilizing Software-Defined Networking (SDN)-based distributed firewall capabilities and the concept of stateful data plane to construct scalable attack graphs in near-realtime, which is a practical approach to use attack graph for realtime security decisions. The goal of the proposed work is to control reachability information between different datacenter segments to reduce the dependencies among vulnerabilities and restrict the attack graph analysis in a relative small scope. The proposed framework is based on SDN's programmable capabilities to adjust the distributed firewall policies dynamically according to security situations during the running time. It apply white-list-based security policies to limit the attacker's capability from moving or exploiting different segments by only

allowing uni-directional vulnerability dependency links between segments. Specifically, several test cases will be presented with various attack scenarios and analyze how distributed firewall and stateful SDN data plan can significantly reduce the security states construction and analysis. The proposed approach proved to achieve a percentage of improvement over 61% in comparison with prior modules were SDN and distributed firewall are not in use.

To My Brother, Mohammad, Whom I Owe All My Success...

ACKNOWLEDGMENTS

Successful writing and completion would not be possible without the guidance, support, and encouragement from my Suporvisor, Dr. Dijiang Huang. He is a great mentor for graduate students, and a great resource and expert in CyberSecurity era. Thank you for teaching me how to think critically, and most of all, how to become a good researcher.

I would like to give a special gratitude for my thesis committee members, Dr. Yanchao Zhang and Dr. Paulo Shakarian for their guidance and support. Thank you for allocating me part of your valuable time to defence my thesis.

Moreover, I am grateful to all of those whom I had the honor and pleasure of working with in Secure Networking and Computing (SNAC) lab, especially Ankur Chowdhary and Adel Alshamrani for their insightful input, revisions, helping me in conducting and successful completion of this research work.

Above all, my friend, partner, and wife, Maryam Hafiz, who stood by me every moment and gave me her endless constant support. Thank you for being in my life, without your existence, I would not have been able to finish my degree on time, thank you for your understanding and patience.

Last, but not least, my family, special thanks and gratitude for my father, Mansour, and my mother, Samar, for your lovely prayer, effort in raising me, and honest advice and guidance to become today who I am.

I gratefully acknowledge the financial support and scholarship I received from Taibah University through Saudi Arabian Cultural Mission (SACM).

TABLE OF CONTENTS

Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Background and Motivation

Data centers and networking systems continue to expand and increase in both size and complexity at a rapid pace. With such expansion, security concerns rise as attackers' capability improves and the number of vulnerabilities grow. In order to have a better understanding and management of the security situation of the system, strong and efficient analysis tools are needed to assist administrators in protecting the critical assets. Graphical security analysis (*Attack Graph*) tools are one of them being used as methods to understand the weakness of a system. Security state management is conducted by monitoring and evaluating system's security components such as Firewall and Intrusion Detection Systems(IDS). It is critical that a system administrator is able to not only inspect and deploy security rules, but also analyze and evaluate the current situation and whether or not an improvement is needed to increase the protection level of the system. Coming back to attack graph as a well-known tool for security analysis, attack graph's scalability is still considered as *NP-Complete* problem Greiner *et al.* (2006). Amman *et al.* Ammann *et al.* (2002) show that attack graph scalability problem limits its scope and applicability. There has been no effective solutions to overcome this problem. Moreover, attack graph-based security scenario analysis approaches carry the problem of states explosion as described in Ammann *et al.* (2002), in which the attack graph represents all the possible attack scenarios (or states) and it can be exponentially complicated.

To mitigate the state explosion problem, most of existing solutions either tried to

reduce the dependency among vulnerabilities Ou *et al.* (2005) or apply an hierarchical strategy Hong and Kim (2013) to reduce the computing and analysis complexity of constructing and using attack graphs. For example, Hong and Kim Hong and Kim (2013) proposed a hierarchical approach for constructing and analyzing full attack graph by simulation. They compared their model with simplified attack graph, which represents the network structure only and not the full attack graph information, where the proposed approach was linearly better than the simplified attack graph. In Kaynar and Sivrikaya (2016), a framework was proposed for distributed attack graph generation from hardware aspect through utilizing memory pages to ensure smooth distribution of the attack graph. In Hong *et al.* (2013), the authors presented an approach for attack graph reduction using reduction technique. Specifically, they showed two formal methods, one is by calculating the full attack path, while the other method is by increasing an existing path. However, they did not show how effective their proposed solution is through a real-time solution or experiments, besides it scales poorly to $O(2^n)$ for full path calculation and $O(n!)$ for incremental path calculation.

Using attack graph is essential to enumerate all the possible attack scenarios, or it can be called security states, for a given computer networking system. None of previous solutions can effectively address the state explosion issue nor considered large data center networks with emerging technologies such as *SDN* and *stateful Distributed Firewall (DFW)*. To further highlight and expound the state term; state is meant to be the entire flow state among SDN environment. In the past, state was referred to the header information being examined. However, state in this thesis describe the complete implication of the entire flow, from header to application data where a comprehensive inspection is conducted based on predefined state for each flow to prevent the attacker from tricking the firewall and bypass protection. The idea of the proposed research work is to provide security state management along

with a graphical analysis of critical paths in the system. Therefor, system admin can make appropriate decision to protect the system. The proposed approach guarantee that the attacker may not bluff the firewall, since we used a methodology to link the communication path with destination port and host IP.

There are many research work proposed virtual Large Data Center Networks to enhance network performance and bandwidth allocation Landis *et al.* (2010); Guo *et al.* (2014). DFW approaches enforces security policy at a different level and components of the network. These security policies are deployed (on demand) and change frequently. A centralized firewall in SDN is not effective as it may block legitimate traffic according to Hu *et al.* (2014a). DFW is more reliable when used in SDN (Software Defined Networking) environment to ensure the ease of use of different policies and configurations for each network segment or Tenant. However, using SDN controller to maintain firewall managed security states is not an effective approach. To address this problem, in this work, we propose to develop an SDN-based stateful DFW by incorporating OpenState Bianchi *et al.* (2014) to manage the system security states in SDN networking environment.

Software Defined Networking (*SDN*) is an emerging technology aiming to enhance the current networking protocols by separating control-plane from data-plane. SDN breaks down the switch and routers controlling functionality and forwarding functionality to erase the vertical complications according to Lantz *et al.* (2010). This method of separation converts the switches in the network into simple forwarding devices controlled by central controller that coordinates and manages the forwarding rules (flow rules), which will reduce the complexity of applying and managing different flow policies and help in minimizing conflict between those policies.Kreutz *et al.* (2015). The separation operation is conducted via software capabilities, application

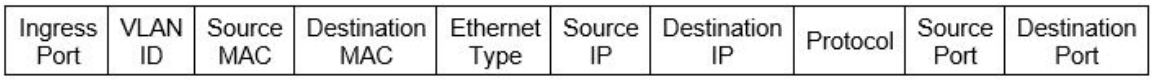

Figure 1.1: OpenFlow Protocol Header Fields

programming interface (APIs) particularly, in which the SDN controller takes control of the data-plane states according to Kreutz *et al.* (2015). The method or protocol of deploying SDN controller was studied widely, starting from McKeown *et al.* (2008) in which OpenFlow protocol was proposed as a standard protocol to implement SDN functionalities. The power of OpenFlow is the dynamic property of adding and removing flow rules in switches, without creating conflicts between the rules. Moreover, the centralization of controller allowed for higher performance and traffic throughput Bianco *et al.* (2010). OpenFlow datapath design include flow table and an action associated with each flow. This exact property will help in designing the *DFW* as we will see later.

OpenFlow switches will act based on the flow rules specified by the controller. Instead of making decision for each packet individually, the controller check for the first packet in each flow, and generalize the rule for the rest of the flow. This way the number of packets transmitted over the network is reduced significantly. As for the basic actions each switch must have; they are 1) Forward the flow. 2) Encapsulate and forward. 3) Drop the flow packet. McKeown *et al.* (2008)
Currently, there are several SDN controllers, and many studies conducted comparing controllers performance and usage Khondoker *et al.* (2014); Nunes *et al.* (2014),the most popular and widely used is OpenDayLight Controller. Figure 1.1 shows the header fields of OpenFlow protocol flow.

As for SDN-based data center, it is worth noting that many companies started to deploy SDN in their infrastructure. For instance, Google announced in 2017 their plan to introduce SDN to the public Network, and 20% of Google's traffic is managed using SDN. Also, CISCO, the largest networking devices manufacturing and provider, has a dedicated plan and products designed specifically for SDN-based data centers. All of these news and efforts prove that SDN is becoming the new networking technology every organization will deploy soon or later, and hence, many researcher are trying to conduct different type of studies either on SDN or using SDN technology.

As of security of SDN, firewall functionality is an essential component of any security system. There are many challenges in deploying distributed firewall in SDN-based environment, especially a stateful one. For instance, conflict resolution as indicated by Dixit *et al.* (2018), where certain rules are overlapping with one another or once. Firewalls shall be capable of handling policy conflict when a flow rule is pushed or upon path update in the system. Also, for large scale data center, it is of paramount importance for a firewall to provide support for multi-tenant architecture, and differentiate between address domain for each sub-network.

Back to attack graphs, previous work Cook *et al.* (2016) worked on generating an attack graph in distributed manner. The method of parallelism depends on breaking down the algorithm loops into parallel processes, and the complexity of generating the attack graph is scaled down by a factor of n or t, where n is the number of nodes and t is the number of edges to the node. Moreover, Kaynar *et al.* Kaynar and Sivrikaya (2016) focused on partitioning the memory of the system where the attack graph computation occur to add parallelism to attack graph generation. The authors also provided a virtual shared memory abstraction across distributed agents to avoid multiple expansion of nodes. The shared virtual memory is used to store the reacha-

bility graph, hence, any two nodes adjacent will be put in the same memory page to the distributed agents, which will lead to processing the nodes in the same page with the same agent without assistant or the need to transfer the page to another agent. The complexity analysis of the proposed algorithm is based on the messages transfer and execution time. The maximum number of memory pages faults encountered by the search agents is $O(N * H$, where H is the number of edges and N is the number of hosts in the network. Overall complexity of the proposed system is determined by $O(P + N * H * log(P))$, where P is the number of processors or the search agents for the memory pages. As can be seen from previous work, attack graph scalability is still a major issue that has not been addressed effectively. As large data center network grow in both size and complexity, it is of paramount importance to come up with a solution to make networking management easier for network administrator. SDN emerged as a solution to solve such a complication, but it also have much more higher capabilities to enhance the security and provided more accurate and effective security analysis. To the best of our knowledge, SDN and distributed firewall have not been used before to address the attack graph scalability issue and address the state explosion problem though controlling the reachability between the nodes in the system.

1.2 Contribution

This thesis aim to provide a solution for the complication behind managing and analyzing security state for large data center networks. The main contribution is that a SDN-based environment was designed to deploy vulnerability-based stateful distributed firewall, by utilizing SDN controller and OVS. Moreover, the presented framework provide scalable graph-based security analysis by incorporating vulnerability information and flow information to get the exact connectivity between the

hosts inside the data center system. The framework achieve a better performance by over 60% in comparison with prior models that utilizes attack graph for security analysis. Other contribution of the work is that it was completed using container based approach, which is an emerging technology and competitor to traditional virtual machines. The work also provide an intuition on how to connect multiple container to OVS in order to allow for local communication as well as communication with the public network for each container.

1.3 Organization of Thesis

In the following chapters, we will explain more in details on the theory and applications of designing stateful distributed firewall *DFW*, the approach used for design, and finally, the method we followed to utilize distributed firewall functionality to generate attack graph that is scalable, and in much lower computation time. we will present related work in Chapter 2. Chapter 3 will provide the details of system design and implementation. In chapter 4, we explain our work about attack graphs and how we were able to combine firewall flow polices with vulnerability scanning results in order to be able to generate a scalable attack graph. Finally, conclusion and future work is presented in Chapter 5 and Appendix section is followed after that

Chapter 2

RELATED WORK

Managing security state for any system is crucial to safeguard the confidentiality, integrity, and availability of data. According to Symantec Internet Security Threat Report sym (2018), the wide spread of software vulnerabilities and ransomewares O'Gorman and McDonald (2012) create many challenges for system administrator to protect and defend against. Our goal here is to offer a scalable approach to manage the security situation and provide an efficient approach of analysis to allow and assist the administrator in performing the effective decision. SDN offers great opportunities to manage the networking interactions between the control-plane and the data-plane's switches using an API. However this new technology eliminate complications Alkhulaiwi *et al.* (2016) between switches and controller, there are many security considerations and challenges that need to be addressed. Administrators now are capable of controlling a specified network segment or isolate is easily, thanks to SDN. The question is how and when this isolation should occur, and most importantly, what criteria or other countermesures can be selected to avoid losing data availability, or confidentiality lose. According to Scott-Hayward *et al.* (2013), one of the main concerns in the industry is how satisfactory the level of audit process, which device is connected to which switch, what what level of access the device can have or reach. SDN security can be categorized into three main points:

- OpenFlow controller related security challenges.
- Switch related security challenges.
- Communication channel between controller and switch security challenges.

Graph-based Security Models	**Tree-based Security Mode**
Sheyner *et al.* (2002)	Saini *et al.* (2008)
Swiler *et al.* (2001)	Bistarelli *et al.* (2006)
Sheyner (2004)	Aliari Zonouz (2012)
Ingols *et al.* (2006)	Roy *et al.* (2012)
Jajodia *et al.* (2005)	Edge (2007)
Ou *et al.* (2005)	Roy *et al.* (2010)

Table 2.1: Classification of Some of Graphical Security Analysis Tools

In the following sections, we present some of the related work in regards to communication channel security and how a distributed firewall was implemented to address those issues. We also show the current work on graphical security analysis and what tools and methodologies are used evaluate the security state in the system and how they can be used to identify the critical paths in the system.

2.1 Graphical Security Analysis

Graphical security analysis model can be divided into two main category: 1- Attack graphs and 2- Attack Trees. The formal is the main approach for graphical security analysis, whereas the later is the basis for tree structured model. Many research work was conducted in this area, such as Attack defense trees to study the effect of hierarchical attack model, and to improve the security assessment capability Hong *et al.* (2017). Hong *et al.* conducted a comprehensive survey on the usability and application of graphical security models. They classified the existing models to study its performance in terms of complexity and and life cycle, as well as compare the availability of tools and what contribution they offer. Chowdhary *et al.* Chowdhary

et al. (2018) presented an approach of modeling Markov game to defend and select the optimal countermeasure selection. The author's idea was to design a model in which a reward will be given to each player (attack and defender) for their action, optimal reward would result in the higher points for defender and lower one for attacker, and vice versa.

Graphical security analysis models can be classified into 2 main categories as we mentioned earlier, however, each category has a sub category, which we show in table 2.1. There exist other types of security analysis models such as stochastic Petri Net Peterson (1981) and stochastic Reward Net Hirel *et al.* (2000). Since they suffer from state explosion problem and they do not follow graphcial approach, the authors of Hong *et al.* (2017) did not consider them for comparison with other modules.

Attack Graph Scalability: Attack Graph Toolkit Sheyner *et al.* (2002) presents an automated way of generating attack graph. Authors considered attack graph generation as a minimum set-cover problem and utilized vulnerability information from Nessus to generate a graph of 19 nodes and 28 edges. The graph generation algorithm, will however not be scalable on a large cloud network. Multi-prerequisite attack graph generation presented by Ingols *et al* Ingols *et al.* (2006) utilizes efficient data-structures to achieve the worst-case attack graph generation complexity of order O(E+NlgN), where N is the number of nodes, and E is the number of edges. The performance results have been evaluated over an attack graph with a few hundred nodes. Additionally, using a greedy algorithm for the generation of the entire graph is a slow process. We utilize a distributed graph generation algorithm to a achieve better scalability on a large network. Lee *et al* Lee *et al.* (2009) presented a cut and divide algorithm to achieve scalable attack graph generation on a large network. The authors, however, assume the graph structure to balanced, to facilitate division. The real world networks, however, rarely balance, i.e., some segments may have a large

number of vulnerabilities, while others may be relatively secured. Albanese *et al* Albanese *et al.* (2012) use exhaustive search algorithm for attack graph generation, and estimate the cost for network hardening. The search algorithm can have exponential complexity in the worst case on a large network. We utilize segment aware attack graph generation algorithm, which will have linear complexity, O(n), where n is the number of attack graph nodes.

To show an example of Attack graph scalability, we present the following figure 2.1:

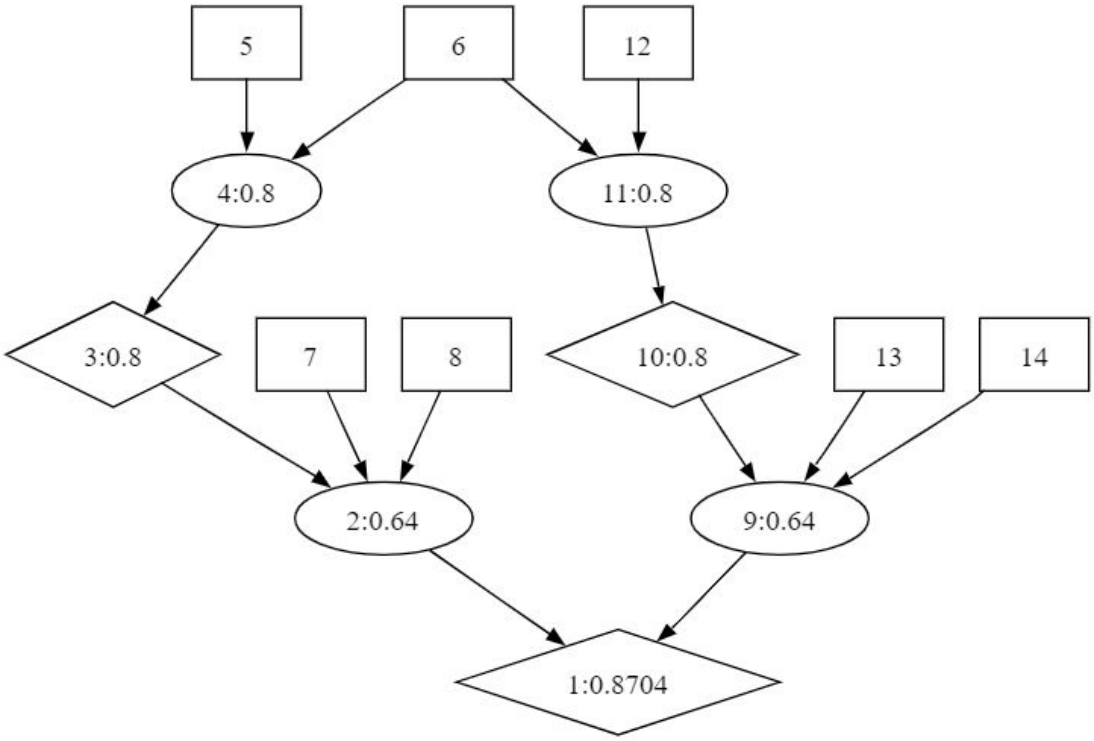

Figure 2.1: An Example of Attack Graph

Where the labels for each shape in the graph mean:

1,"execCode('192.168.1.6',someUser)","OR",0.8704
2,"RULE 2 (remote exploit of a server program)",

```
"AND",0.64
3,"netAccess('192.168.1.6',tcp,'22')","OR",0.8
4,"RULE 6 (direct network access)","AND",0.8
5,"hacl(internet,'192.168.1.6',tcp,'22')",
"LEAF",1.0
6,"attackerLocated(internet)","LEAF",1.0
7,"networkServiceInfo('192.168.1.6',openssh,tcp,
'22',someUser)","LEAF",1.0
8,"vulExists('192.168.1.6','CVE-2008-5161',openssh,
remoteExploit,privEscalation)","LEAF",1.0
9,"RULE 2 (remote exploit of a server program)","AND",0.64
10,"netAccess('192.168.1.6',tcp,'9090')","OR",0.8
11,"RULE 6 (direct network access)","AND",0.8
12,"hacl(internet,'192.168.1.6',tcp,'9090')"
,"LEAF",1.0
13,"networkServiceInfo('192.168.1.6',safari,tcp,
'9090',someUser)","LEAF",1.0
14,"vulExists('192.168.1.6','CVE-2013-2566',
safari,remoteExploit,privEscalation)","LEAF",1.0
```

The above figure show an instance of a simple graph with two vulnerabilities only. In Chapter 4, we show a more sophisticated graph, and how our solution reduced the complexity of interpreting the graph meaning.

SDN based Attack Graph Generation: SDN allows centralized management and orchestration in a cloud network. The network controller having a centralized

view of network traffic and vulnerabilities makes it an ideal candidate for attack graph generation. Scalable MTD solution presented by Chowdhary *et al* Chowdhary *et al.* (2016) use graph partitioning for attack graph scalability and proactive security. We build on similar principles, but the sub-attack graph generation and merging process used in our current work achieve faster graph generation. Chung *et al* Chung *et al.* (2015, 2013) use SDN based attack graph generation and countermeasure evaluation framework for analyzing the impact of different security countermeasures on the security state of a network. The graph generation algorithm is polynomial in terms of the number of network nodes, which can limit the scalability of the attack graph generation process. We use SDN controller to centrally compose segment graphs, and achieve 75% reduction in the overhead associated with single attack graph generation for the entire network.

2.2 SDN-Based Distributed Firewall

Firewalls are one of the essential security components that exist in many systems. Firewall are capable of filtering or blocking certain type of traffic, whether it belongs to layer two, layer three, or application layer. There are mainly three type of firewall:

1. Stateful firewall: which allow for state-based inspection of the traffic, and state here correspond to the traffic state (New or Established for instance).

2. Stateless firewall: in which the firewall act as a static filtering mechanism based on pre-defined rules.

3. Application firewall: basically an application oriented filter that run in the application domain and defend against rule violation to a specified filter.

Some of the research work that built SDN-based firewall include Pena and Yu (2014), where they implemented a simulated environment by using Mininet Team

(2012) and connecting multiple switches together then testing the **ping** command, nevertheless, their approach depend on installing the firewall rule as a flow rule to show the distributed property of the firewall. Installing some rules as a flow rule does not effectively address many security challenges and may not defend against web server vulnerability for example since the attacker may simply act as a normal user and then compromise that vulnerable server. Moreover, since the controller is the one responsible for generating and installing the rules, the term distributed has been eliminated and the controller now is the main engine in the system for defense.

SDN Reactive Stateful Firewall zerkane *et al.* Zerkane *et al.* (2016) introduced a stateful reactive firewall by incorporating the firewall into the SDN architecture. The authors claim it is stateful since it monitor the connection status and react based on the state. There are three main components in their design, which is: 1- an orchestrator that run in the application layer, 2- firewall application running on top of SDN controller and 3- OpenFlow policies installed in OpenFlow data-plane devices. The purpose of the orchestrator is to allow the administrator to specify security policy in the high level, and then those security policies are "propageted" to the controller. Each firewall instance has a state table to keep track of the states of the flow. Firewall specify which action the controller should take, which in this regards ease the load on the controller and only send to it what action to be performed. The authors claim this will mitigate some attack such as DDoS and SYN Flood attack. This stateful firewall design lack auto priority handling, multi tenant support, and violation resolution according to Dixit *et al.* (2018). Their evaluation is implemented using Mininet Team (2012) and the results show that the firewall is capable of processing 1000 connection requests in about 0.7 ms.

FLOW GUARD According to Hu *et al.* (2014b), there are many challenges in building a SDN firewall, such as

- Examining Dynamic Network Policy Updates
- Checking Indirect Security Violations
- Architecture Option
- Stateful Monitoring

In order to address these challenges, The authors proposed a robust firewalls that enable effective network-wide access control, namely *FLOWGUARD*. The framework accurate detection and policy violation in SDN environment. FLOWGUARD has several components that configure, verify, and manage the flows through network state configuration, flow packet violation detection and flow rejection in case of violation modules. The authors ensured that the framework have enough flexability, and efficiency to dynamically adopt to network state changes. Evaluation of FLOWGUARD was performed using *FLOODLIGHT* SDN controller Floodlight (2012). The experiments were conducted using real world network topology, and they achieved a detection and rejection strategy in 0.03 milliseconds (ms).

SDN-Oriented Stateful Hardware Firewalls Collings and Liu Collings and Liu (2014) presented a hardware approach into designing a stateful SDN firewall, where they incorporated the flow rules in both, OpenFlow switches and firewall controller, where the latter is responsible for making control decisions for unknown flows. The main idea is to utilize SDN controller to insert and remove rules based on predefined security polices. The authors evaluated their design using GENI test-bed which provide real world network simulation model. They achieved an overall latency of

about 30 ms for 300 flows with 1000 rules, however, they did not test their approach for distributed multiple network segments. Moreover, the authors did not consider evaluating the security state after deploying the proposed firewall.

Multi-level Stateful Firewall Mechanism for SDN Naif and Kotulski Nife and Kotulski (2017), presented an approach to design a reactive stateful firewall based on OVS data. They relied in their work upon having the SDN controller to specify the pre-defined state, which is a simple "match action paradigm ". The firewall application is centralized above the control plane. They compared their work with other SDN security module that either offer protection against DoS attack or having a simple filtering mechanism. Authors claim that their proposed solution only need 26-bytes for each flow entry in the STable, where they store the states; however, there is no real evaluation nor implementation for their solution on a real system, nor the authors identified how the OpenFlow devices will communicate with each other to ensure there is no redundancy or conflict between them.

Chapter 3

SYSTEM DESIGN, MODELING, AND MANAGEMENT

In this Chapter, we present the proposed system architecture design in terms of: 1- System components and how they are connected. 2- Details of traffic flow from source until destination 3- Different stages of security states and how they are handled in such SDN-based environment. A comprehensive explanation of the environment walk-through is presented as well to show a proof of concept for our proposed framework.

3.1 System and Architecture Components

The proposed system rely heavily upon Software Defined Networking architecture, especially SDN Controller. There are many types of controllers that follow OpenFLow protocol specifications such as OpenDayLight Controller, Pox, etc. a comparative study was conducted to emphasize on each one by Khondoker *et al.* (2014).
In addition to SDN controller, We use OpenState Bianchi *et al.* (2014), which is a proposed open source tool that utilizes Mealy-based eXtended Finite State Machine (XFSM) to model and handle flow states in SDN environment. The goal of Open-State is to allow the programmer not only to include states in the OpenFlow device, but also the ability to manage those states and the devicse shall be able to handel the state without controller assistant.

The goal is to design a stateful firewall capable of distinguishing benign flow from malicious one by monitoring the current state of the traffic and the vulnerabilities in the system. Open Virtual Switch (OVS) Lantz *et al.* (2010), is used as OpenFlow

data-plane device to receive and execute controller's commands and flow rules. OVS is essential for open state as the XFSM tables actions are being pushed in there and it is responsible to forward the traffic for the destination or to Intrusion Detection System (IDS) to inspect the malicious traffic. Next, the ongoing security vulnerability and critical paths in the system are modeled using attack graph. A proposed modeling approach is being used to reduce the scalability of the graph via controlling reachability by distributed firewall policies.

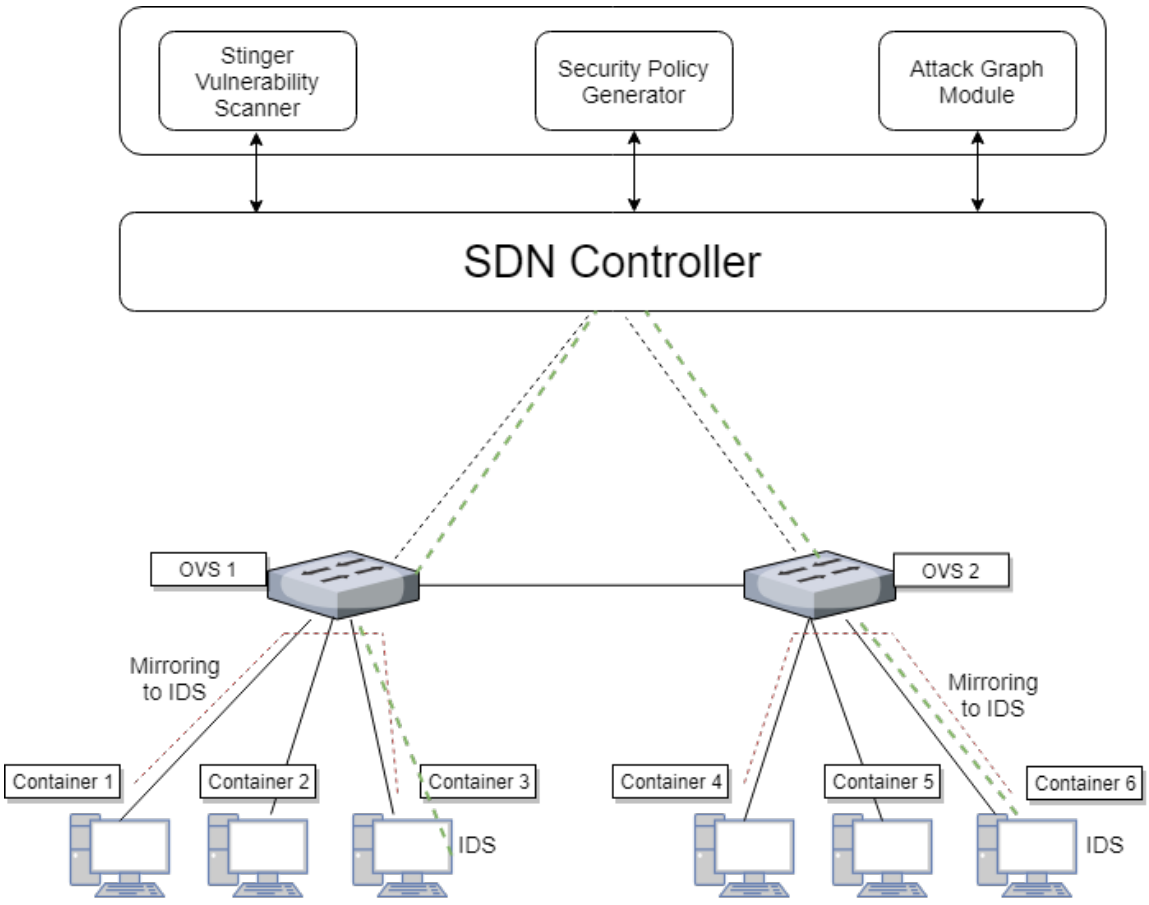

Figure 3.1: System Architecture of The Proposed System

The proposed framework is presented in Figure 3.1, where the application layer has the vulnerability scanner, the security policy generator based on OpenState, and

finally the attack graph module that will compute and generate the attack graph for the system. The next layer is the Control layer in which the SDN controller resides and act as a meditate between the upper modules and the data-plane layer. Controller job is to manage the connectivity across the system, by receiving the security policy from the policy generator, converting it into a flow rule, and push it to OVS. The next layer is the data-plane layer where OVS is responsible to execute the flow rules as well as monitoring the state of the connected machines. If there exist a connection to a vulnerable machine, OVS forward the traffic to an IDS for further inspection. Once an alert is generated, it is sent to the security policy module generator to execute and update the current security policy and block the malicious flow.

3.1.1 OpenState

OpenState Bianchi *et al.* (2014) was introduced by Bianchi *et al.* as a solution to bring and provide states for SDN data-plane. The authors utilized eXtended Finite State Machines (XFSM) as the main technique to allow switches at the data-plane level to be programmable. The goal was to add intelligent techniques to the switches and reduce the load and decision making from being done in the controller. Moreover, allowing switches to maintain states is not enough, it is necessary to provide state management for the devise itself Bianchi *et al.* (2014). Therefor, Finite State Machines were selected to fulfill this requirement.

OpenState was implemented using two main tables: 1- State tables; which hold the state for each flow, and 2- XFSM table; where each state is linked with a state key, triggering event, associated action, and next state label. Figure 3.2 shows an example of mealy type state machine. Each state is represented by a circle. To have a valid

transition from one state to another, a valid triggering event must occur, otherwise, the next state is still the current state. We explain more in details about OpenState in a later section 3.1.4.

To highlight on different state management schemes for states in SDN data-plane layer, we show a comparison in table 3.1. The table emphasized the scalability of the approach, what data structure it used, and what is the type of the scheme. We choose OpenState for our design for it's simplicity and availability of code in comparison to the other modules.

Scheme	**Scalability**	**Type**	**State Storage**
OpenState	Can support multiple XFSM tables	Platform	Hash Table + TCAM
Fast	Scalable	Platform	Hash table
SDPA \cite{zhu2015sdpa}	Scalable	Platform	TCAM + SRAM
SNAP \cite{arashloo2016snap}	Evaluation show scalability when topology size increase, but not when No. of flow policy increase	Framework	Hash Table
Event-Driven Network Programming \cite{mcclurg2016event}	No evaluation Result provided	Framework	Registers

Table 3.1: Comparison Between Some of The Famous Platforms for State in SDN

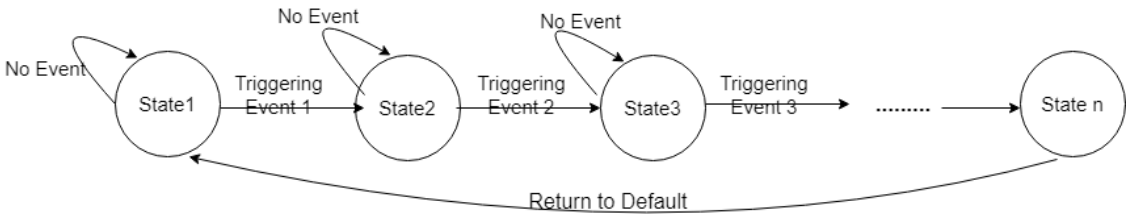

Figure 3.2: An example of Mealy Finite State Machine

3.1.2 Distributed Firewall (DFW)

Firewalls are one of the important security elements in any networking system. They can control the flow of packets from one node to another by inspecting the traditional five tuples (source and destination IP address, source and destination port number, and protocol). There are three types of firewalls, stateless, stateful, or application firewall. Application firewall basically is an application oriented filter that run in the application domain and defend against rule violation to a specified filter. It control the input/output and access to and from an application by blocking any unmatched traffic. For example, an organization specify an access policy to whom can access to sensitive data servers which will help in preventing unwanted and unauthorized access. Stateless firewall is essentially a filtering mechanism to remove (or drop) unwanted traffic based on static information such as IP address, access control list (ACL). They do not account for traffic state nor they monitor network status. A stateless firewall can also serve as an access gateway to allow or deny certain type of traffic or users from entering to some unauthorized areas by comparing the pre-specified rule sets and check for a match. On the other hand, the stateful firewall has the capability to monitor network traffic in order to inspect any path change in the network. Moreover, a stateful firewall can watch specific connection stage in TCP protocol (SYN, SYN-ACK, etc) Mojidra (2016). They add

to the filtering mechansim monitoring functionality to watch for the newly opened ports by any connection. In essence, stateful firewall add Layer-4 realization to the basic filtering model.

Modern DFW Architectures

There has been a shift in paradigm from host-centric model to the data-centric model. The network services and computation capacity is available closer to the users. One of the emerging solution to prevent lateral movement of attack in the network is the usage of microsegmentation via a distributed firewall. The distributed firewall model proposed by microsegmentation allows segmentation of the network at various layers of abstraction - layer 2,3,4 or segmentation of application workloads within the same layer, e.g., web-application layer, database layer, etc. A segmented network thus protects workloads against attacks even when the attacker has footprint within a network segment.

Existing microsegmentation solutions such as VMWare NSX Ferrari (2014) satisfy recommendations listed in NIST 800-125b Chandramouli and Chandramouli (2016) guidelines (VM-FW-R1-3). We use object oriented microsegmentation model to create security policies at the abstraction level of security and user-level groups, thus satisfying all the recommendations above. Additionally, both VMWare ESX and CISCO ACI Morgan (2014) frameworks allow the creation of microsegmentation within a multi-segment cloud network.

The drawback of such an architecture is that SDN architecture forwards every new traffic request to the SDN controller. With the security policies at the granularity of per-application, workload, flow-state, SDN controller may be quickly overwhelmed, and make the security assessment quite slow. We incorporated a light-weight state monitoring capability in our architecture to achieve the same capabilities as modern

DFW architectures while limiting the impact on the network performance. If a centralized firewall is used in an SDN environment and the SDN controller is enforced to track every connection, the attacker can launch a saturation attack as described in AVANTGUARD Shin *et al.* (2013). Also, a centralized firewall can not detect and defend against attacks in data-plane layer Dixit *et al.* (2018).

In order to restrict traffic between different segments, it is important to have some sort of mechanism that allows us to control the flow through the communication paths. The obvious solution is to use a firewall. Nevertheless, deploying a central firewall will suffer from a single point of failure issue. Moreover, different segments will have different security requirements, which is impossible for a central firewall to accommodate all at once. The policies for distributed firewall are centrally generated and managed, nevertheless, those policies are pushed into the OpenFlow devices (OVS) to be maintained.

In our case where we utilize the SDN, DFW will resolve the problem of flow policy violation by setting up an individual firewall for each entry Hu *et al.* (2014b). The several firewalls will be synchronized by maintaining connectivity with the SDN controller, which is responsible for generating the state tables as we will explain in the next sections.

Some researcher addressed DFW in SDN Hu *et al.* (2014b) Satasiya *et al.* (2016) Pena and Yu (2014). However, they only consider stateless firewall which does not leverage the full advantage of both SDN and DFW. VMware has proposed a distributed firewall for their NSX model, by using a central object that manages the distributed firewall's policies Mojidra (2016). Unfortunately, this architecture is only applicable to the NSX model and cannot be adopted to OpenFlow standards, because NSX comprises of stateful and stateless components. The firewall rules of the host machines are also controlled by the NSX manager. Whereas the OpenFlow imple-

ments a stateless firewall. Also, NSX follows a distributed firewall model, and SDN is a centralized controller model, which is another difference.

There are many challenges and research questions that need to be addressed to consider stateful firewall implementation. Initially, all the intelligence work in SDN was designated to the controller Dargahi *et al.* (2017), and switches were dedicated to maintaining stateless forwarding tables based on the controller. However, this is not the case anymore as stateful SDN introduces the concept of empowering switches to partially control incoming flows. For the case of a multi-segment data center, which is the focus of this thesis, network configuration will change dynamically as well as the flow states. This alternation raises the need for stateful rules that can be configured inside the switch to satisfy each segments' requirements, either from security or networking point of view. As we will see later, utilizing the state of the SDN flow and the state of the existing system's vulnerabilities can result in such alike stateful firewall.

3.1.3 System Design

In the previous sections, the system architecture and components were introduced. The rest of the chapter will show the detailed flow of the system components and what are the dependencies between them.

To illustrate the flow of the system, consider Figures 3.3 and 3.4, which shows the flow chart of the system. The first step is to conduct vulnerability scanning results, which is the base to build the stateful firewall and evaluate the security situation of the system. The scanning results are sent to the controller in order to generate the XFSM table with the assistance of OpenState. XFSM table include information about the vulnerable service in a certain sub-domain. This domain shall be under OVS observation since OVS will be responsible to forward the flow to IDS/IPS or block it if an intrusion was identified. The procedure on how to generate the state table will be explained shortly. The SDN controller will now push the generated state tables to the OVS, and OVS will track flow based on the specified flow key and associated state. If a state indicate a flow is suspicious, then the flow is forwarded to IDS/IPS for further inspection and examination. Next, a decision will be made to determine whether the flow will be forwarded to the destination or get blocked. If the flow is benign, it will be returned to OVS to be forwarded to final destination, otherwise, it gets dropped. If no state was found, OVS send the flow headers and wait for forwarding rule decision.

The previously described procedure show how to combine both, centralized decision making and distributed rule enforcement, which is crucial for Distributed Firewall (DFW) to have. SDN controller will manage and push the flow rules based on OpenState module, each OVS will enforce those rules and maintain the security state of each network segment it is attached to. Hence, distributed functionality is main-

tained in addition to adding intelligent property for the OVS devices. The purpose of designing a DFW is that:

- It is a software-based firewall, which essentially means it can be easily enabled or disabled on any network segment or interface.
- The design needs to support stateful firewall to handle the situation if a malicious traffic (i.e. traffic associated with known vulnerability and port number) is detected, which results in enabling detailed packet filtering policies according to the flow states.

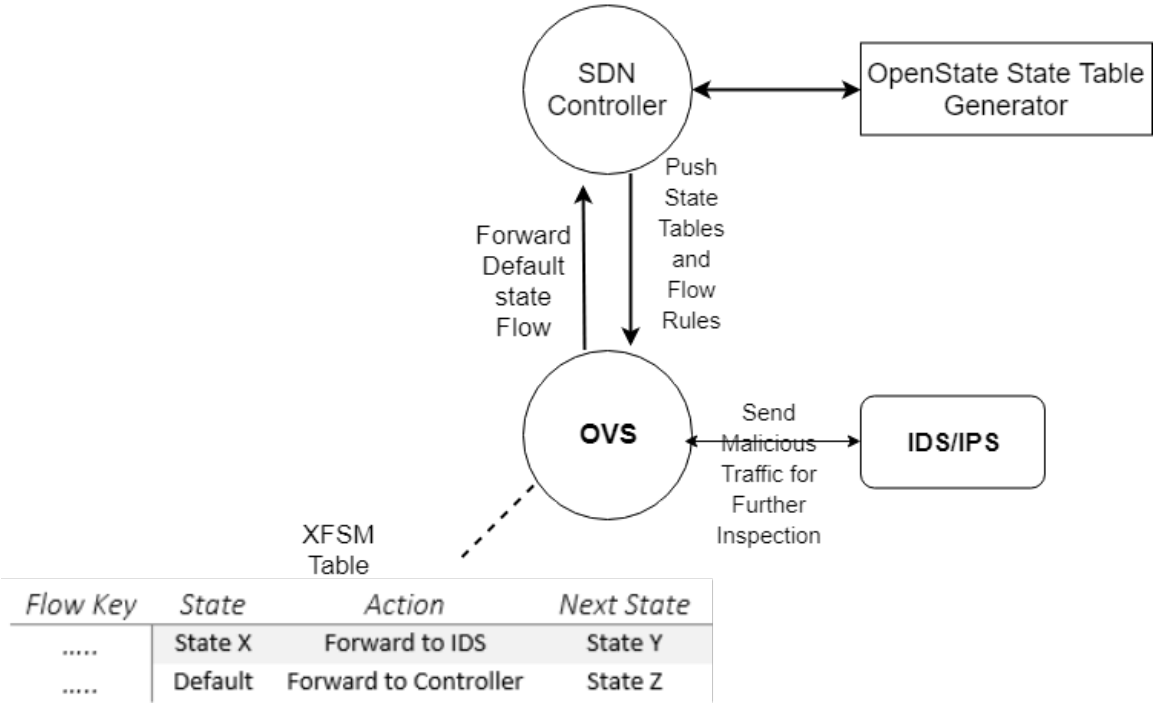

Figure 3.3: Flow Chart Diagram of The Proposed System

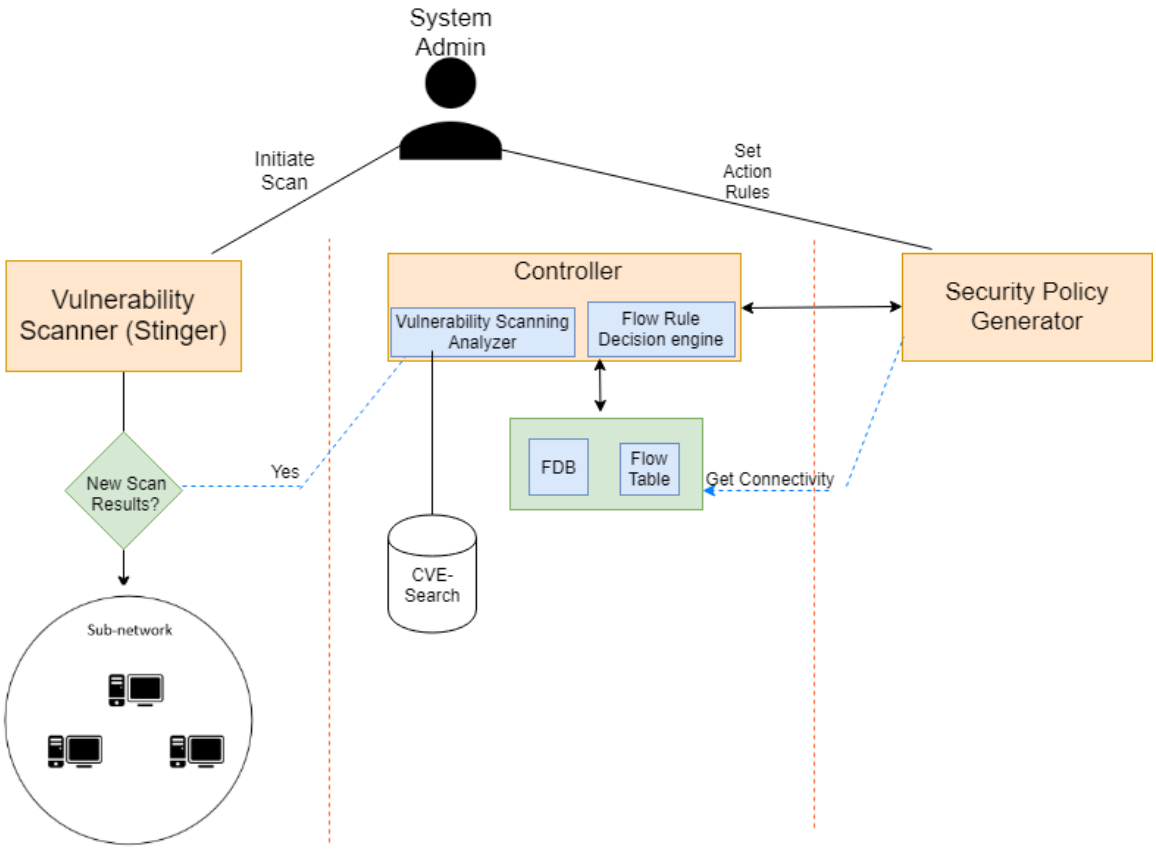

Figure 3.4: Flow Diagram of The Proposed System

3.1.4 OpenState Tables Generation

OpenState was proposed by Bianchi *et al.* Bianchi *et al.* (2014) to support state management in OpenFlow enabled switches. OpenState relys on Mealy eXtended Finite State Machine *(XFSM)* to describe the current flow state, trigring event for each flow based on flow key, what action is executed for each event, and finally what is the next state. To illustrate on OpenState XFSM, let's consider the following example:

Concrete Example:

Suppose we have a sub-net with the following hosts and vulnerability information shown in table 3.2:

Host	Vulnerability	CVE ID	Port
192.168.1.10	WebDAV vulnerability in IIS	CVE 2009-1535	135
192.168.1.11	Squid port scan	CVE 2001-1030	200
192.168.1.12	None	NA	NA

Table 3.2: Sub-net Configuration Example

We assume there is a dedicated agent for each network segment responsible for vulnerability scanning, and later on attack graph computation. The agent will provide the information shown in the table above to controller. The controller will query for the mentioned *CVE IDs* and get more detailed information, which then sent to security policy module to examine and generate the distributed firewall rules.
to elaborate on OpenState we explain the requirments for generating the tables as follows:

- Flow Key: which is defined by Destination IP and Port number
- State: The state is assigned depending on whether a vulnerability exist or not.
- Event: Incoming Connection from another node to a specific port.
- Action: The action is set either forward to IDS/IPS, wait for security clearance, or forward to Controller.
- Next State: What state should the rest of the flow get?

Figure 3.5 shows the State table in the top and the XFSM table at the bottom of the figure. once a flow come in, OpenState module locate the associated state label for this flow. Next, the module check XFSM table to see if the associated triggering event for this state is present for in the flow, if so, the action will be associated and the next state label will be assigned for the flow and get updated in the upper state table.

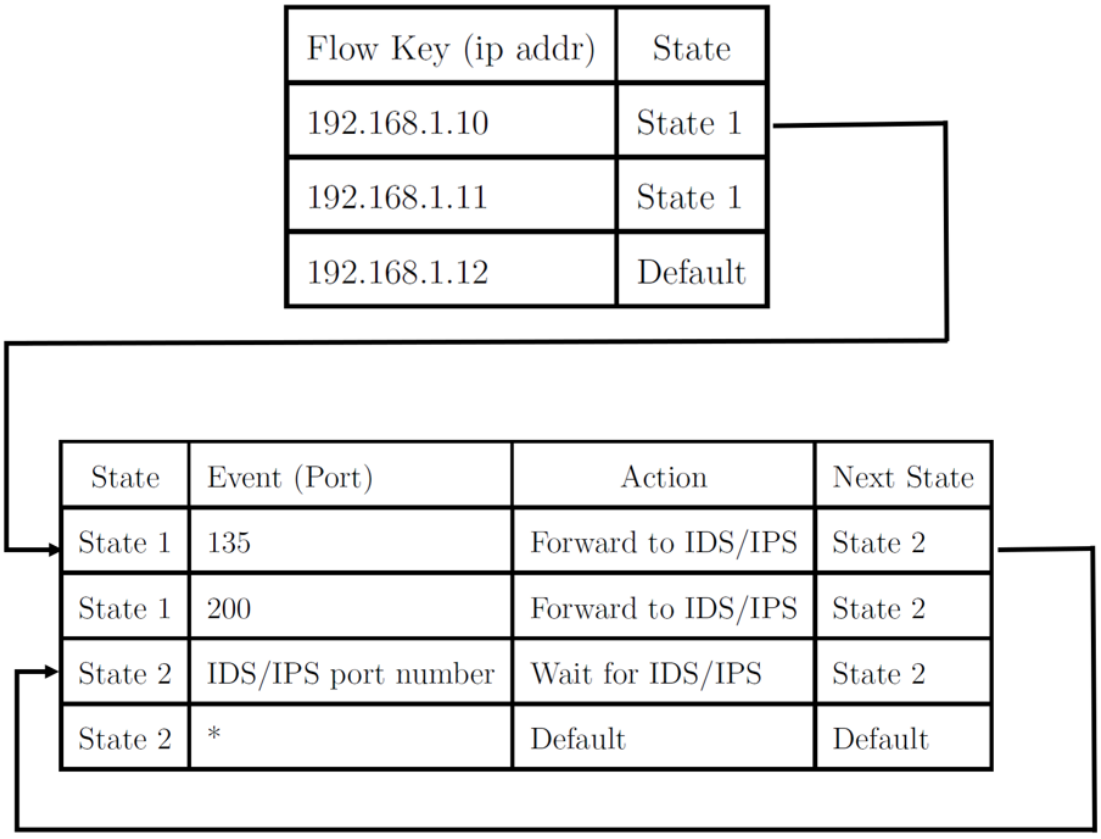

Figure 3.5: State and XFSM Tables of OpenState

3.2 Stateful Distributed Firewall Implementation Details

In this section, we will present the approach to implement the DFW functionality, as well as what exact software/components are being used and utilized for this purpose. Before carrying on with the rest of the chapter, it is important to point out that these component are being developed and tested using a limited processing hardware device, and the idea can be replicated for each network segment to build the entire DFW. The purpose is to provide a proof of concept through real implementation of the proposed systems.

3.2.1 Operating System and Networking virtualization

To start with the implementation, we used Dell laptop with Intel *I-7* 2.6 GHz processor with 16GB RAM. The running operating system on the machine is Windows 10. Next, we installed Oracle VirtualBox Version 5.2.6 as a base hypervisor to host virtual machines. We installed Ubuntu 14.04 there to run and do all the experiments.

The required main components for our environment testing include:

- Open Virtual Switch (OVS)
- SDN Controller
- Linux Containers (lxc)
- Snort IDS/IPS

Inside ubuntu terminal, we begin by installing the above requirements using the following commands:

```
1- installation of OVS:
$ apt-get update
$ apt-get install -y git automake autoconf gcc uml-utilities
libtool build-essential git
$ git clone https://github.com/openvswitch/ovs.git
$ cd ovs
$ ./boot.sh
$  ./configure --with-linux=/lib/modules/ uname -r  /build
$ make && make install
$ insmod datapath/linux/openvswitch.ko
$ mkdir -p /usr/local/etc/openvswitch
$ ovsdb-tool create /usr/local/etc/openvswitch/conf.db
vswitchd/vswitch.ovsschema
$ ovsdb-server -v --remote=punix:/usr/local/var/run/openvswitch/
  db.sock \
--remote=db:Open_vSwitch,manager_options \
--private-key=db:SSL,private_key \
--certificate=db:SSL,certificate \
--pidfile --detach --log-file
$ ovs-vsctl --no-wait init
$ ovs-vswitchd --pidfile --detach
$ ovs-vsctl add-br ovsbr1
$ ovs-vsctl show
```

Figure 3.6 shows successful installation of OVS after executing the above commands.

```
hakem@ubuntu:~$ sudo ovs-vsctl show
6e268fe3-3062-44ac-897e-096a4823afc8
    Bridge "br0"
        Port "br0"
            Interface "br0"
                type: internal
hakem@ubuntu:~$
```

Figure 3.6: Successful Installation of OVS

The next part now is to install Linux container, and for that, we will allocate a dedicated section as follows:

Linux Containers

Linux containers Rosen (2014) which was introduced in August 2008 Wikipedia contributors (2018) and developed by number of developers from IBM and Google, is designed to enable virtulization for Linux Operating system, where all containers share the same kernel as the host. Containers are considered exactly as a regular virtual machine, with the exception of they use and share the same resources (memory and kernel) as the host does. We choose to continue with our implementation and testing using containers because of the simplicity of managing the containers and attaching them to OVS, all using the host terminal command.

In order to install Linux containers (lxc), some dependencies are required to run it efficiently, which are:

- One of glibc, musl libc, uclib or bionic as your C library
- Linux kernel >=3.8
- libcap (to allow for capability drops)
- libapparmor (to set a different apparmor profile for the container)

Next, run the following command to install lxc:

$ sudo apt-get install lxc

To create Linux containers (virtual machines) for our experiments, we use the following code:

$ sudo lxc-create -t ubuntu -n u1

which indicate ubuntu version container with name u1 is created. Figure 3.7 show successful creation of this container. We created 3 containers, two are acting as normal clients (web server and FTP server), while the last one is acting as IDS/IPS components.

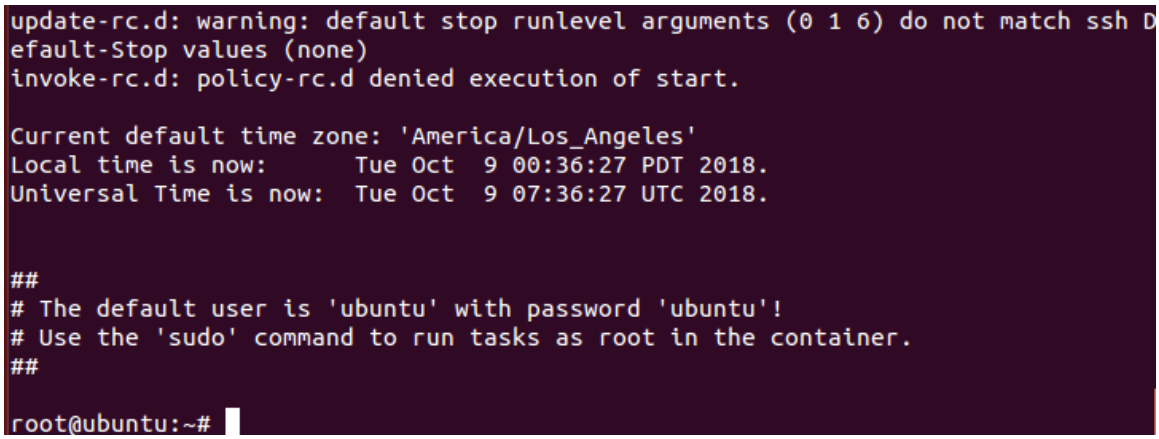

Figure 3.7: Successful Creation of LXC Ubuntu Container

3.2.2 Network Configuration

In this section, we will explain the networking environment setup for our proposed work. First of all, as mentioned earlier, the work is dependent on SDN environment, which include SDN controller and data-plane switches. For the purpose of implementation and testing, we used POX controller Kaur *et al.* (2014), which is python-based controller that has several modules, including but not limited to: layer-2 switching, layer-3 switching, etc.

The next step is to start linking the OVS to each container. It is critical for the containers not only be able to communicate with each other, but also to have access to the internet. For such a purpose, the switch need to have the capability to forward each container's traffic to the internet and to other containers as well. Therefor, the following configuration is used to allow container to be connected dynamically to the OVS. Once a container start, it will be assigned a physical interface, and the logical port of it will be connected to the OVS. Moreover, if the container shutdown, the created logical port will be deleted from the OVS connected port, to no flood the OVS with unwanted bridges and ports.

In order to connect the container to OVS, the following must be done:

1- Edit the file in /etc/lxc/default.conf as follows:

2- After creating a container, edit it's configuration file as follows:

This step is essential after creating any container, otherwise, it will not connect to the OVS and will fail in starting.

Figure 3.8: OVS Main Configuration File

3- Create the following new two files as follows:

```
$ sudo touch /etc/network/if-up.d/lxcora02-asm2-ifup-ovsbr1
$ sudo touch /etc/network/if-up.d/lxcora02-asm2-ifdown-ovsbr1
```

4- Edit the created files as follows:

a) sudo vim /etc/network/if-up.d/lxcora02-asm2-ifup-ovsbr1

```
#!/bin/bash
BRIDGE="ovsbr1"
sudo ovs-vsctl --may-exist add-br $BRIDGE
sudo ovs-vsctl --if-exists del-port $BRIDGE $5
sudo ovs-vsctl --may-exist add-port $BRIDGE $5
```

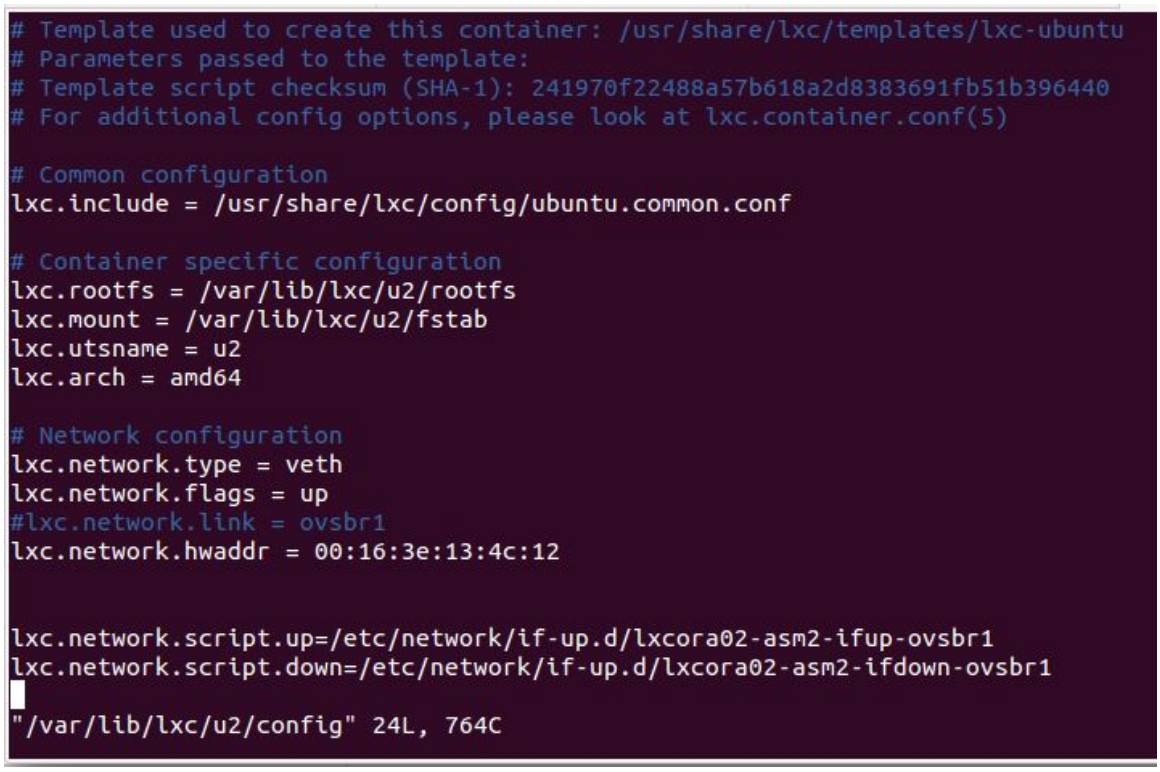

Figure 3.9: Container Main Configuration File

b) sudo vim /etc/network/if-up.d/lxcora02-asm2-ifdown-ovsbr1

```
#!/bin/bash
BRIDGE=‘ovsbr1‘
#ovsBr=  ovsbr1
#ovs-vsctl  if-exists del-port ${ovsBr} $5
NET_CONFIG=/etc/network/interfaces
ovs-vsctl --if-exists del-port $BRIDGE $5
sed -i -n '/allow-'$BRIDGE' '$5'/{s/.*//;x;N;N;N;N;d;}
;x;p;${x;p;}'
$NET_CONFIG
sed -i '/./,/^$/!d' $NET_CONFIG # remove leading blank lines
```

At this stage, OVS and containers are connected and container should be able to communicate with public network.

In order to proceed with the experiments, as shown in Figure 3.3, an Intrusion detection system (IDS) and Vulnerability Scanner module (Stinger) are needed. The Stinger module installation and preparation is explained in the Appendix chapter of this Thesis. As of IDS configuration, we used Snort Roesch *et al.* (1999) intrusion detection system for it's popularity and ease of configuration. Snort was installed in container 3 to inspect any abnormal behavior for the traffic that is identified as vulnerable by Stinger module. To apply specific traffic forwarding, we used OVS port mirroring, which is basically a function to copy all traffic as it is from one port to another one.

To do this, we used the following commands:

```
sudo ovs-vsctl -- --id=@m create mirror name=mirror0 --
add bridge ovsbr1 mirrors @m
sudo ovs-vsctl set mirror mirror0
output_port=d11cc9f9-1287-464f-8454-68205e896fd7
sudo ovs-vsctl set mirror mirror0
select\_dst\_port=c524609b-4830-4bc8-bbe7-4ddfbcc29b7c
```

where *output* is the uuid of the ovs port that we want to send the traffic to, and *dst_port* is the uuid port the we want to listen to it's ongoing traffic (any traffic going to that port). in order to get the uid of a port, we list all ports and their ids by the follwing command:

```
$ sudo ovs-vsctl list port
```

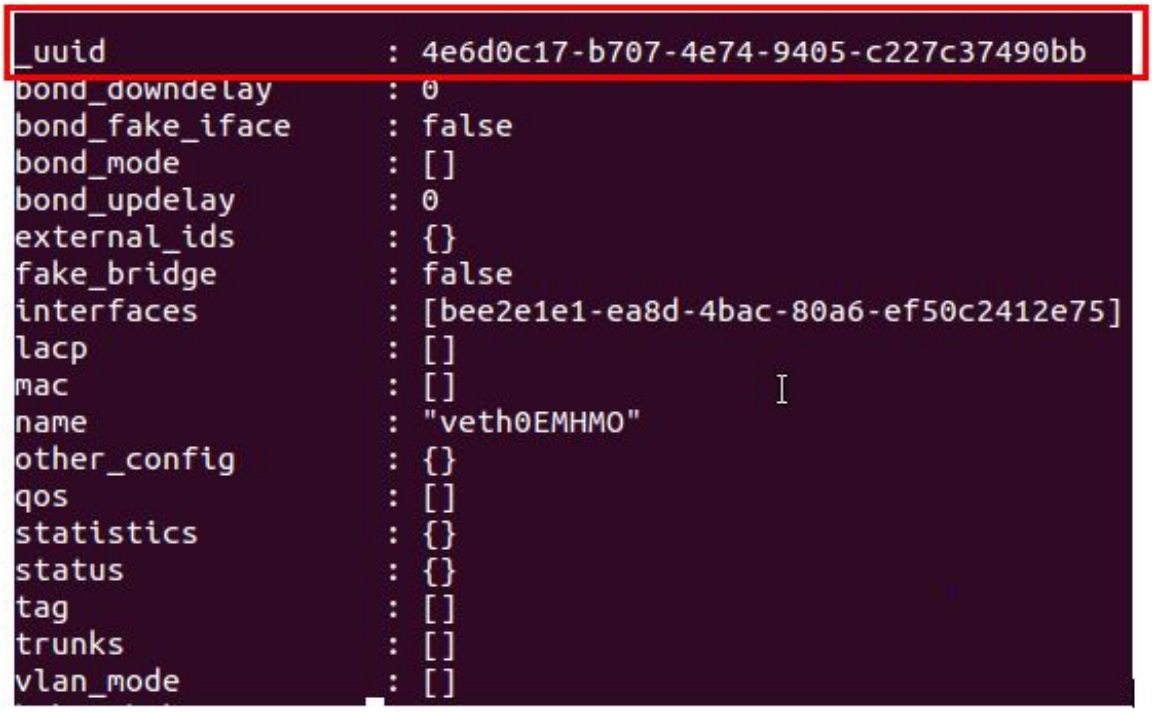

Figure 3.10: Example of *uuid* for an OVS Port

Port mirroring prevent any usage of the port that is receiving packets. Therefor, we added an additional port to container 3 in order to be able to communicate with other containers and the public network as well. Adding a physical port is done by changing the container main configuration file as follows:

```
# Template used to create this container:
/usr/share/lxc/templates/lxc-ubuntu
# Common configuration
lxc.include = /usr/share/lxc/config/ubuntu.common.conf
# Container specific configuration
lxc.rootfs = /var/lib/lxc/u3/rootfs
lxc.mount = /var/lib/lxc/u3/fstab
```

```
lxc.utsname = u3
lxc.arch = amd64

# Network configuration
#lxc.network.name = eth0
lxc.network.type = veth
lxc.network.flags = up
#lxc.network.link = ovsbr1
lxc.network.hwaddr = 00:16:3e:9d:35:47
lxc.network.script.up=
/etc/network/if-up.d/lxcora02-asm2-ifup-ovsbr1
lxc.network.script.down=
/etc/network/if-up.d/lxcora02-asm2-ifdown-ovsbr1

#lxc.network.name =eth1
lxc.network.type =veth
lxc.network.flags = up
lxc.network.hwaddr = 00:16:3e:9d:35:10
lxc.network.script.up=
/etc/network/if-up.d/lxcora02-asm2-ifup-ovsbr1
lxc.network.script.down=/etc/network/if-up.d/
lxcora02-asm2-ifdown-ovsbr1
```

Finally, we are now able to get IDS generated alerts by designing an API that will return the source IP and source port number, destination IP and destination port number. This information is the goal for the controller to create a flow rule that will

drop any traffic having the same attributes. Also, this information will be used by the attack graph module to compute the overall system attack graph. Thus, the goal was achieved by ensuring only desirable flow will pass through the controller, after inspection by IDS, and based on the original vulnerability scanning result.

Chapter 4

SCALABLE ATTACK GRAPH GENERATION

In this chapter, we present our methodology and approach on how we are able to generate a scalable attack graph after utilizing distributed firewall capability and SDN environment. The main ideas is to monitor the vulnerabilities in the system and then embed a path to the graph whenever a vulnerability exist and an active communication to that vulnerability is being established. Our approach relied on MulVal tool Ou *et al.* (2005) to compute the attack graph, and finally we draw it using *d3 javascript* library. The evaluation result show an improvement with over 60% in comparison with prior modules .

4.1 Motivation

Attack graph has been used as a modeling tool for the study of multi-hop attacks in a network. In addition, it has application in a number of areas of network security such as vulnerability analysis where a system administrator can understand a collective impact on network security. However, one of the greatest challenge making the usability of the attack graph unpractical is its scalability. The relationship between the generation of attack graph and its scalability is a direct relationship. Thus, in a large data center network, generating attack graph can exceed our ability to analyze and understand the relationships between vulnerabilities and system flow's. The main motivation is to solve the attack graph's scalability issue and enhance the attack graph's usability and visualization. For this purpose, we utilize the *SDN* and *DFW* technologies for the purpose of controlling the reachability between the data

center segments by allowing uni-directional links between the segments only, and by obtaining real-time reachability information from the controller.

The decoupling of control-plane and data-plane in SDN provides more flexibility to run multiple applications on the SDN controller where each of which has complete knowledge of the controlled environment. In this case, it is conceivable to design a model that periodically runs to discover the network topology and fetching newly added devices, services, or links that can go up or down. Therefore, the output of this model can be input to the attack graph's Host Access-Control Lists (HACL). This will result in constructing real-time attack graph. Although the current deployment of attack graph proposes to use HACL and obtain such information from a firewall management tool, it is static and considers any to any relationship for connectivity between nodes and vulnerabilities Ou *et al.* (2005).

Our methodology of examining each segment individually in order to construct a global view attack graph helps us to enforce security policies at a very granular level to reduce the access policy space and limit the trusted zone between multiple components in the network. Deploying DFW with the SDN-based environment will fully automate the policy configuration. Later, we will explain how reducing the access policy and integrate it into the attack graph HACL will help in solving the attack graph's scalability issue.

4.1.1 Attack Graph Background

The attack graph is a graphical representation of the vulnerabilities in the system. It shows all the possible paths an attacker may take to compromise the system and gain the desired level of privileges, taking into account vulnerabilities dependency, pre-conditions, and post-conditions for building the graph and successful exploita-

tion.

Previously, Chung *et al.* (2013), extended the definition of MulVAL attack graph as Scenario Attack Graph (SAG), as follows:

SAG: SAG =(V,E), where: Vertices $V = N_{\mathrm{C}} \cup N_{\mathrm{D}} \cup N_{\mathrm{R}}$,
such that C is conjunction node representing exploit, D is disjunction node representing results of exploit and root node R for initial step of the attack scenario.
Direct edges E: $E_{pre} \cup E_{post}$, where an edge, $e \in E_{\mathrm{pre}} \subseteq N_{\mathrm{D}} \times N_{\mathrm{C}}$ means that $\mathrm{N_D}$ must exist to reach to $\mathrm{N_C}$. an edge $e \in \mathrm{E_{post}} \subseteq \mathrm{N_C} \times \mathrm{N_D}$ means that the output shown by $\mathrm{N_D}$ can be reached if $\mathrm{N_C}$ is satisfied Chung *et al.* (2013).

The scalability issue of attack graph is a major concern for researchers. It is of paramount need to design an approach that scales well, especially for large data center networks. In an attack graph, a cycle might appear where the attacker is able to exploit vulnerability more than once. This issue will be discussed afterwards.

MulVALOu *et al.* (2005) is a well-known open source tool to generate an attack graph. It uses datalog and logic programming as its modeling language. The input to MulVAL is vulnerability information, which includes but not limited to; Common Vulnerabilities and Exposures (CVE-ID) Vulnerabilities (2007), the affected application or service, vulnerability consequences (whether the vulnerability results in data loss, remote exploitation, data integrity, etc). In addition, network reachability information, which include for each host: IP address, the vulnerable service or application, port, and protocol. Vulnerability information, network service information as well as Host access level in MulVAL are represented as follows:

```
VulExists ('IP Address','CVE ID',Vulnerable service)
NetworkServiceInfo('IP Address',Vulnerable service,
                   Protocol,Port)
```

HACL(src_address, dst_address, Protocol, port)

where *port* means the vulnerable port at the destination node.

MulVAL uses host access-control level (HACL) tuples to model network and firewall configuration, the authors of MulVal used a general rule to test and specify reachability information (any host can access any host using any port and protocol). Figure 4.1 shows a simplified version of an attack graph. Let's consider the attacker's

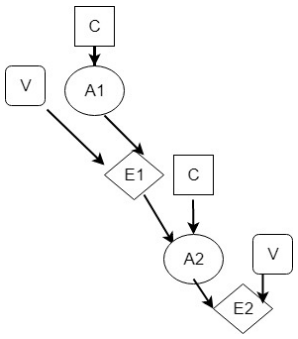

Figure 4.1: Sample Attack Graph

goal is to compromise node *E2*. As can be seen from the Figure, in order to exploit node *E1*, the attacker has to gain access to node *A1* first through condition *C*. An example of pre-condition is:

vulExists (10.0.0.1, CVE-4545, apache1.3.4).

In the second-stage, the pre-condition of exploiting *E1* is both access to *A1*, and vulnerability exist *V*. Note that the required connectivity has to be the same as the one related to the vulnerable service *V*. In order to exploit node *E2*, the attacker first has to compromise node *E1*, which will lead to a communication link to be created and hence, continue to the path to *E2*. Thus an attacker can launch a mulit-stage attack in order to reach goal node E2, starting from *V* and *C*.

The proposed approach computes an attack graph for each sub-network (seg-

ment). In order to construct the global view of attack graph for the entire system, we need to consider the post-condition of the resultant sub-attack graphs. If there is a dependency between these post-conditions (specifically, connectivity between the segments), then we take these post-conditions and make them as pre-conditions in the global view attack graph of the system.

After combining post-conditions, state and post-condition tables are created, the first is to be into the data-plane switches, while the last is for the controller to keep track of post-conditions of vulnerabilities. This is essential for the controller to generate the global-view attack graph.

For example consider a segment having three machines, *202.0.0.1*, *203.0.0.1*, and *203.0.0.2*. The first machine has

```
vulExists('202.0.0.1','CVE-1999-0045',http_server),
networkServiceInfo('202.0.0.1',http_server},tcp,'80').
```

Which has the specified CVE-ID, service, and the associated reachability information which indicates that the *httpserver* service is running on port 80 using protocol tcp. Also, Machine *203.0.0.2* has:

```
vulExists('203.0.0.2','CVE-1999-0511',windows_2000).
networkServiceInfo('203.0.0.2',windows_2000,tcp,'70').
hacl(any,any,any,any)
```

The original attack graph as computed in Ou *et al.* (2005), will consider reachability relationship between the machines as *any* to *any* as shown above where there is no added security policy as well. We want to specify the exact connectivity between the 2 machines by modifying the reasoning rules and add what we consider as the major variables which are *src_ip_address*, *dst_ip_address*, *dst_port*, and *protocol* as indicated by the vulnerabilities information. In addition, add firewall rules to relate

vulnerability information to flow rules. This is accomplished by specifying states for each flow. In order for the attacker to compromise segment *203.0.0.2*, they have to go through *202.0.0.1* and *203.0.0.1* first. Therefore, the post-condition of exploiting *203.0.0.1* becomes pre-condition of *202.0.0.1*.

Post-Condition Table

Source IP	Destination IP	CVE ID	Severity	Port
203.0.0.1	202.0.0.1	CVE-1999-0045	6.4	80
202.0.0.1	203.0.0.2	CVE-1999-0511	6.4	70

Figure 4.2: Post-Condition Table for The Controller

Tables shown in Figure 3.5 show an example of specifying each flow rule for the above vulnerability information. The first step is to initiate state lookup by using the source and destination IP and port number (since we are considering incoming traffic from outside the segment). If the switch finds the corresponding state, it will go to XFSM table to take the required action (forward to DPI in state1 for instance). Otherwise, it will treat the incoming flow by the default state and check the original flow table maintained by the controller to make the appropriate decision. To illustrate more on the reachability information, *HACL* should be modified according to the exact topology fetched from the SDN controller. for instance, machine *202.0.0.1* can reach machine *203.0.0.2* using port 70 and TCP protocol. *HACL* should look like:

hacl('202.0.0.1','202.0.0.1',tcp,'70')

Firewall starts by looking for a match into state table for the incoming traffic. State table contains the exact source and destination IP and port, it will then go to XFSM table to check what action should be taken.

4.1.2 Parallel Attack Graph Computing

In order to collect vulnerability information and scanning results, we assume that a dedicated agent *(Stinger)* inside each segment exist to perform scanning operation, and local attack graph generation. The method of scanning is out of scope of this thesis, we assume that once scanning is done, vulnerability information will be sent to SDN controller's state table module to generate the state and post-condition tables for both, local and global attack graph. Next, the machine will fetch connectivity information from the controller, along with the resultant vulnerability information and compute the attack graph for that particular segment.

Scalable Attack Graph

After computing an attack graph for each segment. We now have small, multiple sub-attack graphs. We want to inspect how to combine all components to generate the overall graph for the system. The obvious next step now is to combine all smaller sub-graphs into a bigger one using divide and conquer. However, it is not that easy due to merging problem. Specifically, when we create a bi-directional link between two segments, which makes the segment-based partitioning useless and as a result, we have to re-inspect all the vulnerability dependency among two segments, which is a non-trivial task especially if there are multiple and dependable vulnerabilities in the two segments. This leads to the following two claims:

Claim 1: *Uni-direction communication between two segments allows us to merge sub-attack graphs, taking into account the ability of the attacker to reach from one host to another through direct communication. Specifically, this direct communication is the one required to compromise the vulnerability residing in the victim machine.*

Here we must note that the link between segments meant the reachability between

vulnerabilities. For example, when VM A from segment 1 explores a vulnerability in VM B on an open TCP port n, and if the DFW allows the connection from A to B on TCP port n, then we say there is a link from segment 1 to segment 2, in which the link is directional and the vulnerability is reachable. Using the stateful DFW, we can easily create uni-directional vulnerability exploration links between segments since the DFW can maintain the connection' states, and thus, the DFW can block unwanted connection requests.

Claim 2: *The different vulnerabilities between the segments might have a dependency between them. This dependency might create a cycle where the attacker is able to reach revisited segment. In order for the attacker to compromise a node E2 from node E1, they have to exceed the exploitability threshold T, which is setup by the system user. The threshold here is the number of nodes that the attacker need to bypass. If the attacker can not exceed this threshold, they should not be able to reach to node E2.*

To realize both Claim 1 and Claim 2, we present the following algorithm 1:

Algorithm 1 Segment Attack Graph construction

```
1: procedure SEGMENT ATTACK GRAPH GENERATION
2:     for all Segments do
3:         Vul_Scan ← Conduct Vulnerability Scanning
4:         Monitor Vulnerable target through IDS.
5:         if new IDS alert from VulScan is found: then
6:             Block attacker's attempt.
7:             Send Vulnerability and connectivity info to AG_Analyzer.
8:             Compute Segment's Local-View AG.
9: procedure SYSTEM ATTACK GRAPH GENERATION
10:    for all segments do
11:        Fetch segment's Vulnerability & Connectivity info.
12:        if Link s1 to vulnerability ∈ s2 is found: then
13:            Add link from s1's AG to s2's AG.
14:    if Global-View AG exist: then
15:        find attack graph cycles().
16:        if cycle C is detected and |C| < T then
17:            prune(C).
18:            Construct-Global View Attack Graph.
19:            Redraw system's attack Graph.
```

Algorithm 1 explains the procedure of constructing a local-view attack graph for each segment in the system, and finally generating the global view attach graph for the entire system by examining the post-conditions resulted from compromising each vulnerability (gaining connectivity to a target node having a vulnerability). The first step of the algorithm is to conduct a comprehensive vulnerability scanning on the target system. Next, Intrusion Detection System *(IDS)* is configured to monitor the targeted vulnerable system. This step is essential in order to prevent blocking legitimate flow originated from normal users, and only detect the malicious flow originated from an attacker. If the IDS generate and alert, then the DFW block that communication and the vulnerability and connectivity information is sent to the attack graph analyzer module. A local-view attack graph is generated for each segment in the system after words, and finally, all segments' local-view attack graphs is sent to global-view attack graph generation module to analyze the dependency between those sub-graphs and generate the entire system attack graph without the need of recomputing the attack graph at each time.

If the segments have no vulnerability dependency between them, specifically, a network connectivity from the source segment to the target segment, then the system user can only see and inspect each segment's local attack graph. On the other hand, after constructing the global-view attack graph, a cycle might appear which will allow the attacker to go back to another node by compromising an intermediate node.
The Algorithm will run for each segment t in the overall network segment n in the system. In order to evaluate the occurrence of a cycle after the global-view attack graph is generated, we use Tarjan's strongly connected components algorithm Tarjan (1972) to implement the *cycles()* to find all cycles in the graph. If a cycle is detected, then we set the exploitability threshold T. T can be set as the number of hops or exploitability of a given attack path computed based on path probability that

is exploitable by the attacker. For a longer cycle, we assume that the attacker has negligible chance to deploy attacks, and thus, we do not need to consider it to reconstruct the attack graph. If the cycle is less than the threshold T, then we apply the *prune* function. In practice, the *prune* function can be achieved by simply selecting a link on a cycle to disable the link by changing DFW filtering rules to break the cycle. The complexity of Algorithm 1 is dominated by the *cycle()* function, which is the complexity of running Tarjan's algorithm: $O(E+V)$ as well as the complexity of MulVal that is $O(n^2)$. Therefore, the worst-case complexity time analysis is $O(n^2)$ where n is the number of nodes in the graph. The algorithm approve to have a better performance in comparison with older approaches, where DFW and SDN are not utilized. On the other hand, the best case scenario complexity is determined if the number of segments in the system is large. We divide the total number of nodes N by the number of segments, and therefor, the best case complexity is determined by $O(n^2/S)$, where S is the number of segemnts in the system.

The attack graph can be generated by utilizing network topology and the vulnerability information for each segment. In order to generate the global view attack graph, the post-condition of exploiting VM1 becomes pre-condition for VM3, and the post-condition for that is the pre-condition for VM5. Figure 4.8 shows the global view of the system's attack graph after using 3S and Figure 4.3 shows the attack graph without *3S*, and it is clear how complicated the previous existing approach even with a small number of vulnerabilities and nodes in the system. It is important to emphasize the DFW role here. The uni-direction communication is maintained by the DFW to ensure the attacker can only advance to one segment, without the capability of coming back to the same point of origin. This connectivity information is synchronized with the controller. As we mentioned earlier, the controller now is able to generate the global view of the attack graph, with the help of uni-direction commu-

nication enforced by DFW. For simplicity, we purposely tested our approach on small number of segments. However, once we compute the attack graph for one segment which could be the difficult part due to large number of vulnerabilities. Thus, if there are multiple segments, then we only have to merge the sub-attack graph by checking for the vulnerability dependency, and do computation only once for the sub-attack graph.

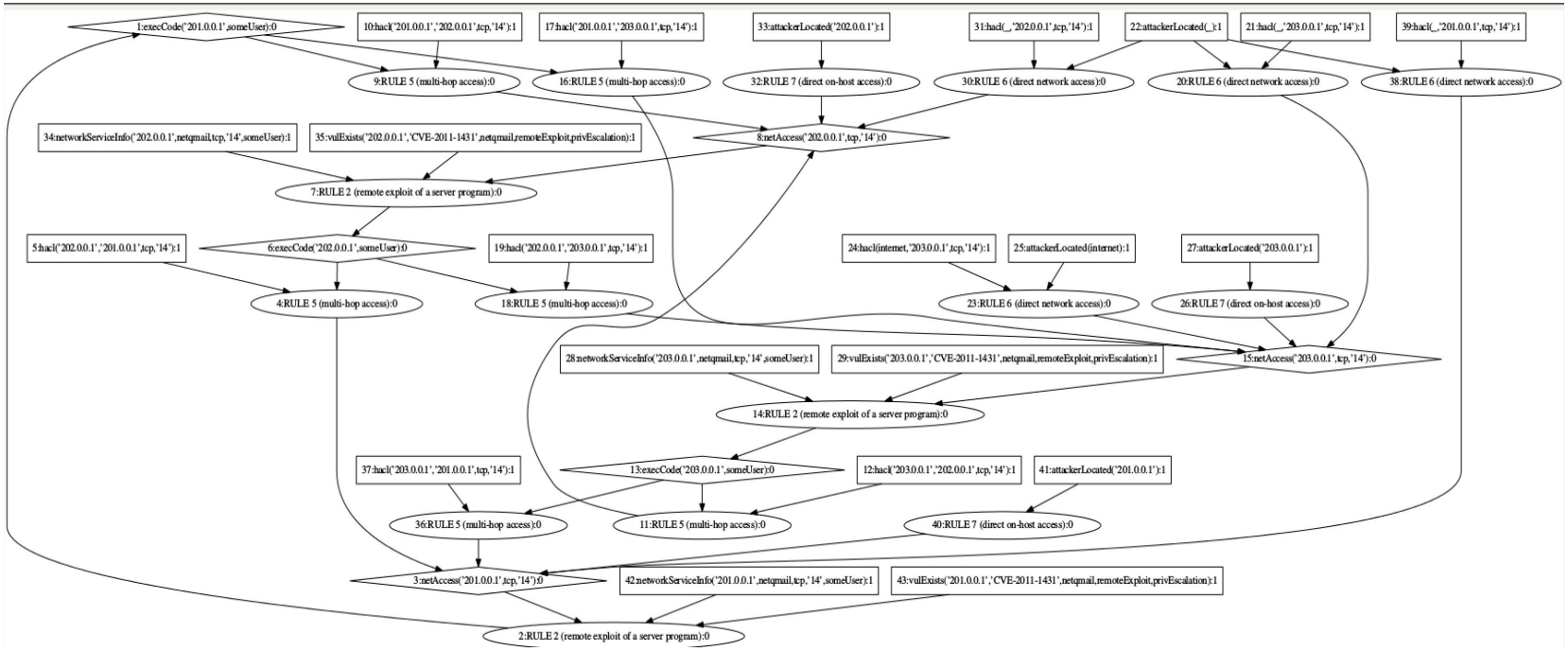

Figure 4.3: Attack Graph Without Using 3S.

4.1.3 Results

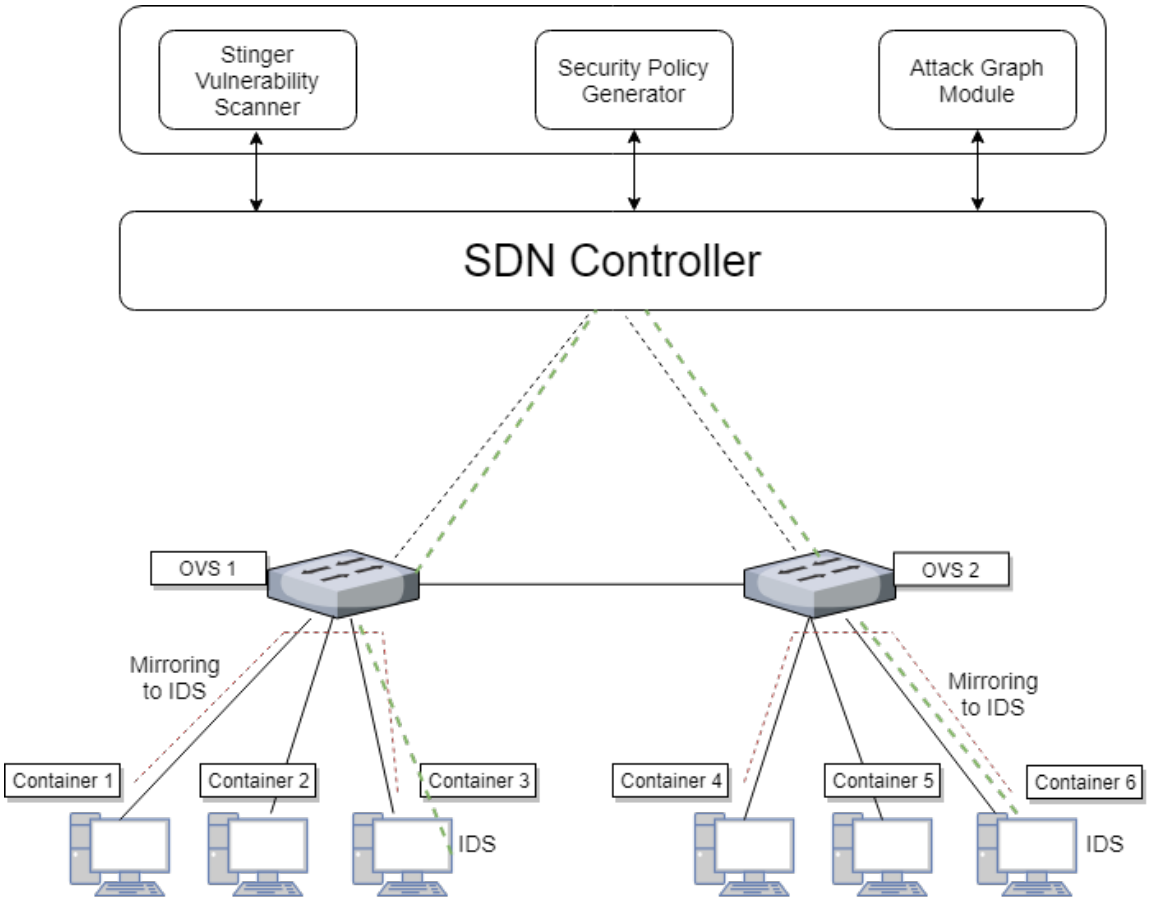

Figure 4.4: System Architecture of The Proposed System

In order to evaluate and measure the performance of our proposed approach. First, we present the following evaluation equation:

Let S be the number of segments in the system and H is the total number of Hosts in the system. The asymptotic complexity of Mulval Ou *et al.* (2005, 2006) complexity is bounded by: $O(H^2)$. Using our proposed approach, we intend to divide the network into a total number of segments T, where each segment T_i having the same number of hosts H. To analyze the best case and worst case complexity of the proposed research, we calculate the best case performance by:

$$Cost(S) = \frac{H}{T}^2 \tag{4.1}$$

On the other hand, the worst case complexity is where the number of segments in the segment is equal to one (no segmentation is occurring), thus, the worst case complexity remain as it is (equal to $O(H^2)$). The larger the number of segmentation, the closer the complexity is becoming linear, which shows how significantly the proposed approach is better and more efficient than the MulVal approach. Figure 4.5 show the line plot of equation 4.1, the complexity is approaching to linear scale as the number of segments increases.

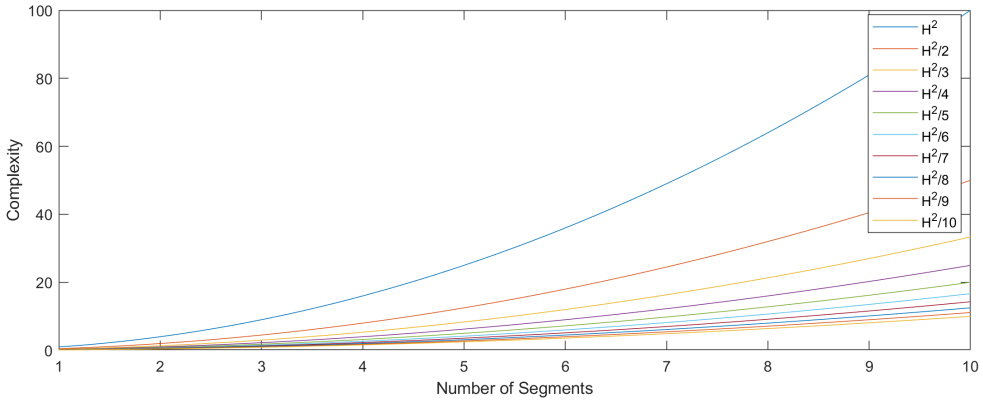

Figure 4.5: Complexity Analysis

As of the machine used for graph computation, we used the Intel I7 2.6-GHz CPU machine with 16GB RAM. Next, we created several attack scenario cases with a different number of vulnerabilities for each segment, and by using a system similar to the one shown in Figure 4.4 to test the approach with the generated test scenarios. Figure 4.6 shows the number of nodes and edges with and without using *3S*. It should be noted that those test cases are for the global view attack graph and does not reveal any information about the local attack graph. Initially, we begin by considering a

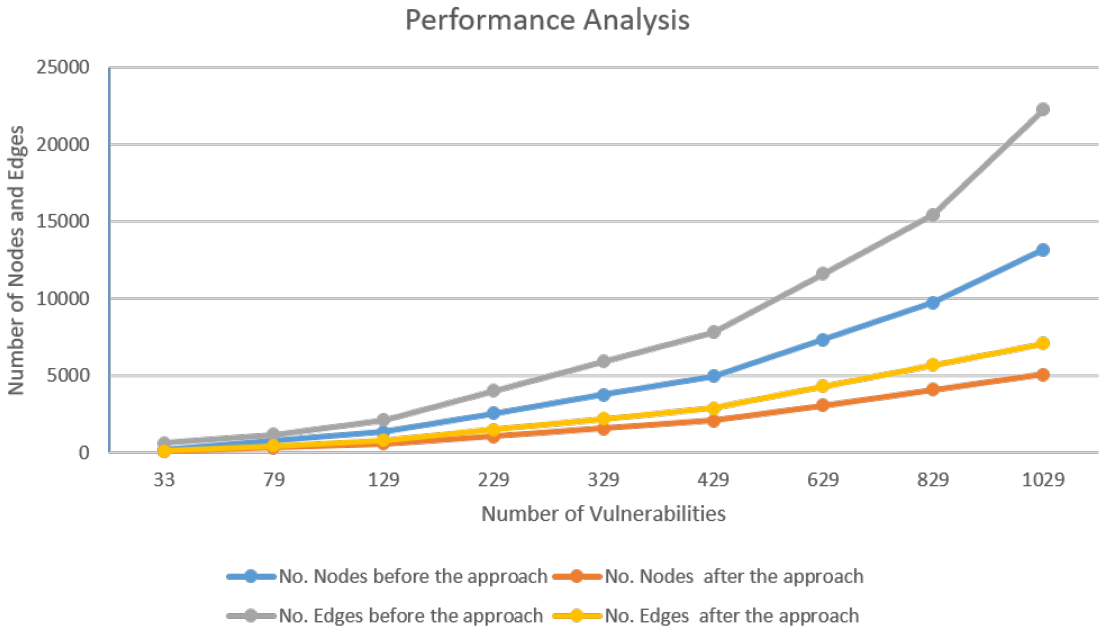

Figure 4.6: Performance Analysis

few number of vulnerabilities. As the number of vulnerabilities increase, the number of nodes and edges in the graph increase with and without applying 3S. When the number of vulnerabilities is about 30, number of nodes dropped from 187 to 91, the percentage of improvement is over 50%. The percentage continues to increase as we go to the right of the graph up until when the number of vulnerabilities is 1029, the number of nodes decline from over 13 thousand nodes to 5 thousand and number of edges decreases from over 22 thousand to 7 thousand with 61% percentage improvement. This is due to a large number of vulnerabilities in the system overall, and the absence of control over the connectivity in the system (the attacker may transfer from any machine to another). Specifically, the graph now contains the actual connectivity information obtained by the controller. In addition, uni-directional links allow removing several nodes and edges from the graph without affecting the actual connectivity between the segments.

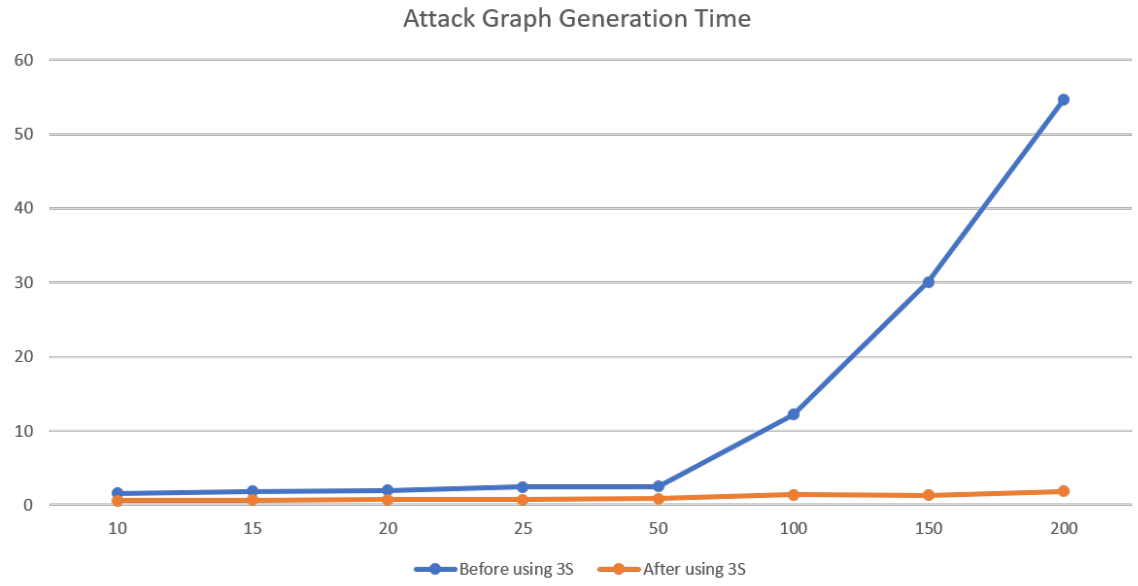

Figure 4.7: Attack Graph Generation Time with and without 3S

The time required to generate attack graph without using our proposed solution scale at very large pace ($O(H^2)$). On the other hand, after deploying *3S*, the time required to generate an attack graph is much less than that and could go to as low as a few seconds. For security risk assessment, it is necessary to construct the attack paths from the source to the target node. However, in a large enterprise network, constructing the attack paths can take a long time, as shown in Figure 4.7. Nevertheless, after deploying our proposed solution, the needed time is reduced significantly from thousands to milliseconds, which shows how efficient the proposed system is.

As for graph clarification and interpretation, we used javascript *d3* library to redraw the attach graph and output a Bayesian based shape. As an example, the resultant graph is shown in Figure 4.8 contains information about the vulnerable

node, as well as the conditional probability of compromising a node in the graph if the link was not blocked and if the attacker was able to satisfy the pre-condition and post-conditions of exploiting the vulnerabilities. The probability formula is derived from Chung *et al.* (2013), we refer the reader to that paper for further explanation. The green color of the node mean the node has a low probability of getting compromised, while the red one has more likelihood of being attacked by adversary. If the mouse over the node occur, the exact text of the node will be shown, however, for sack of simplicity, we did not include the Figure with its text to not disturb the reader with the interconnected texts.

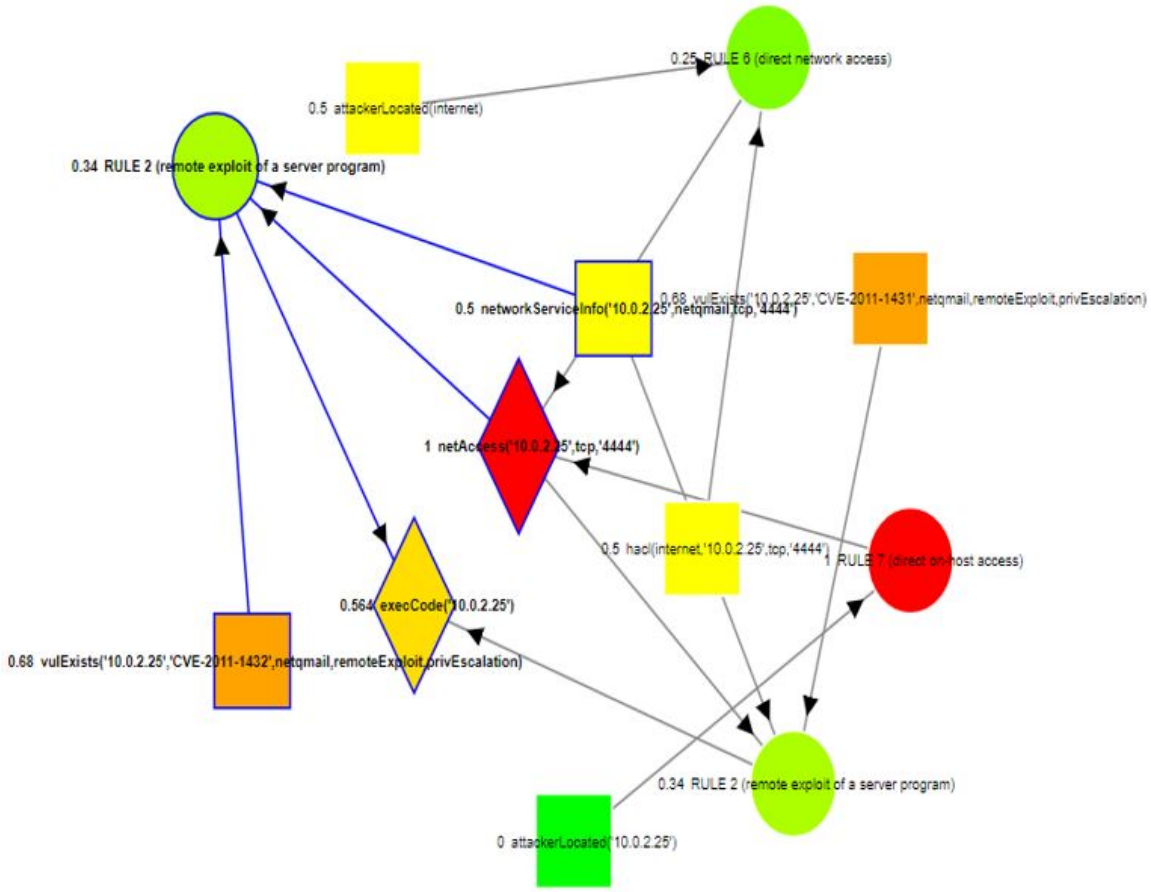

Figure 4.8: Bayesian-based Attack Graph

Chapter 5

CONCLUSION AND FUTURE WORK

In this thesis, we presented a new framework that allow system administrator to manage and analyze the security state for a large data-center network. Firewalls are a powerful component of any system, which ensure smooth flow for network traffic. Every system will need some way to analyze its security state. For that purpose, we proposed to combine both, firewalls and attack graphs as a visual representation of vulnerabilities and critical paths an adversary may take to exploit the system. Software Defined Networking are an emerging technology that enable the decoupling between the control-plane and data-plane, which will allow a great utilization of network devices, as will as reducing the conflict and ease the management of the networking system. our goal in the thesis was to utilize the SDN capabilities and increase the system security situation. Usage of SDN controller allow me to cordinate the security state between the switches, where the goal is to come-up with a distributed firewall to limit the attacker capability from exploiting multiple nodes in the system. DFW will help in reducing the number of nodes that are generated by the attack graph module. Specifically, the exact connectivity between the network segment turned out the play a major rule to generate a scalable attack graph in realtime fashion.

Future work will include incorporating a module in the SDN controller to control and insert a host-level firewall rules in the container machines directly. Also, building an UI portal to allow the administrator to examin exact security polices deployed in the system, insert rules directly into the host-based firewall, and most of all, keep generating the attack graph continuously and enhance the visualization for easier interpretation.

REFERENCES

"2018 internet security threat report", URL `https://www.symantec.com/security-center/threat-report` (2018).

Albanese, M., S. Jajodia and S. Noel, "Time-efficient and cost-effective network hardening using attack graphs", in "Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on", pp. 1–12 (IEEE, 2012).

Aliari Zonouz, S., *Game-theoretic intrusion response and recovery*, Ph.D. thesis, University of Illinois at Urbana-Champaign (2012).

Alkhulaiwi, R., A. Sabur, K. Aldughayem and O. Almanna, "Survey of secure anonymous peer to peer instant messaging protocols", in "Privacy, Security and Trust (PST), 2016 14th Annual Conference on", pp. 294–300 (IEEE, 2016).

Ammann, P., D. Wijesekera and S. Kaushik, "Scalable, graph-based network vulnerability analysis", in "Proceedings of the 9th ACM Conference on Computer and Communications Security", pp. 217–224 (ACM, 2002).

Bianchi, G., M. Bonola, A. Capone and C. Cascone, "Openstate: programming platform-independent stateful openflow applications inside the switch", ACM SIGCOMM Computer Communication Review **44**, 2, 44–51 (2014).

Bianco, A., R. Birke, L. Giraudo and M. Palacin, "Openflow switching: Data plane performance", in "Communications (ICC), 2010 IEEE International Conference on", pp. 1–5 (IEEE, 2010).

Bistarelli, S., F. Fioravanti and P. Peretti, "Defense trees for economic evaluation of security investments", in "Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on", pp. 8–pp (IEEE, 2006).

Chandramouli, R. and R. Chandramouli, "Secure virtual network configuration for virtual machine (vm) protection", NIST Special Publication **800**, 125B (2016).

Chowdhary, A., S. Pisharody and D. Huang, "Sdn based scalable mtd solution in cloud network", in "Proceedings of the 2016 ACM Workshop on Moving Target Defense", pp. 27–36 (ACM, 2016).

Chowdhary, A., S. Sengupta, A. Alshamrani, D. Huang and A. Sabur, "Adaptive mtd security using markov game modeling", arXiv preprint arXiv:1811.00651 (2018).

Chung, C.-J., P. Khatkar, T. Xing, J. Lee and D. Huang, "Nice: Network intrusion detection and countermeasure selection in virtual network systems", IEEE transactions on dependable and secure computing **10**, 4, 198–211 (2013).

Chung, C.-J., T. Xing, D. Huang, D. Medhi and K. Trivedi, "Serene: on establishing secure and resilient networking services for an sdn-based multi-tenant datacenter environment", in "Dependable Systems and Networks Workshops (DSN-W), 2015 IEEE International Conference on", pp. 4–11 (IEEE, 2015).

Collings, J. and J. Liu, "An openflow-based prototype of sdn-oriented stateful hardware firewalls", in "Network Protocols (ICNP), 2014 IEEE 22nd International Conference on", pp. 525–528 (IEEE, 2014).

Cook, K., T. Shaw, P. Hawrylak and J. Hale, "Scalable attack graph generation", in "Proceedings of the 11th Annual Cyber and Information Security Research Conference", p. 21 (ACM, 2016).

Dargahi, T., A. Caponi, M. Ambrosin, G. Bianchi and M. Conti, "A survey on the security of stateful sdn data planes", IEEE Communications Surveys & Tutorials **19**, 3, 1701–1725 (2017).

Dixit, V. H., S. Kyung, Z. Zhao, A. Doupé, Y. Shoshitaishvili and G.-J. Ahn, "Challenges and preparedness of sdn-based firewalls", in "Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization", pp. 33–38 (ACM, 2018).

Edge, K. S., "A framework for analyzing and mitigating the vulnerabilities of complex systems via attack and protection trees", Tech. rep., AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING AND MANAGEMENT (2007).

Ferrari, M., "Release: Vmware nsx 6.1", (2014).

Floodlight, P., "Project floodlight open source software for building softwaredefined networks", (2012).

Greiner, R., R. Hayward, M. Jankowska and M. Molloy, "Finding optimal satisficing strategies for and-or trees", Artificial Intelligence **170**, 1, 19–58 (2006).

Guo, C., G. Lv, S. Yang and J. H. Wang, "Virtual data center allocation with bandwidth guarantees", US Patent 8,667,171 (2014).

Hirel, C., B. Tuffin and K. S. Trivedi, "Spnp: Stochastic petri nets. version 6.0", in "International Conference on Modelling Techniques and Tools for Computer Performance Evaluation", pp. 354–357 (Springer, 2000).

Hong, J. B. and D. S. Kim, "Performance analysis of scalable attack representation models", in "IFIP International Information Security Conference", pp. 330–343 (Springer, 2013).

Hong, J. B., D. S. Kim, C.-J. Chung and D. Huang, "A survey on the usability and practical applications of graphical security models", Computer Science Review **26**, 1–16 (2017).

Hong, J. B., D. S. Kim and T. Takaoka, "Scalable attack representation model using logic reduction techniques", in "Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on", pp. 404–411 (IEEE, 2013).

Hu, H., G.-J. Ahn, W. Han and Z. Zhao, "Towards a reliable sdn firewall.", in "ONS", (2014a).

Hu, H., W. Han, G.-J. Ahn and Z. Zhao, "Flowguard: building robust firewalls for software-defined networks", in "Proceedings of the third workshop on Hot topics in software defined networking", pp. 97–102 (ACM, 2014b).

Ingols, K., R. Lippmann and K. Piwowarski, "Practical attack graph generation for network defense", in "Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual", pp. 121–130 (IEEE, 2006).

Jajodia, S., S. Noel and B. OBerry, "Topological analysis of network attack vulnerability", in "Managing Cyber Threats", pp. 247–266 (Springer, 2005).

Kaur, S., J. Singh and N. S. Ghumman, "Network programmability using pox controller", in "ICCCS International Conference on Communication, Computing & Systems, IEEE", vol. 138 (2014).

Kaynar, K. and F. Sivrikaya, "Distributed attack graph generation", IEEE Transactions on Dependable and Secure Computing **13**, 5, 519–532 (2016).

Khondoker, R., A. Zaalouk, R. Marx and K. Bayarou, "Feature-based comparison and selection of software defined networking (sdn) controllers", in "Computer Applications and Information Systems (WCCAIS), 2014 World Congress on", pp. 1–7 (IEEE, 2014).

Kreutz, D., F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-defined networking: A comprehensive survey", Proceedings of the IEEE **103**, 1, 14–76 (2015).

Landis, J. A., T. V. Powderly, R. Subrahmanian and A. Puthiyaparambil, "Virtual data center that allocates and manages system resources across multiple nodes", US Patent 7,725,559 (2010).

Lantz, B., B. Heller and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks", in "Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks", p. 19 (ACM, 2010).

Lee, J., H. Lee and H. P. In, "Scalable attack graph for risk assessment", in "Information Networking, 2009. ICOIN 2009. International Conference on", pp. 1–5 (IEEE, 2009).

McKeown, N., T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, "Openflow: enabling innovation in campus networks", ACM SIGCOMM Computer Communication Review **38**, 2, 69–74 (2008).

Mojidra, N., "Stateful vs. stateless firewalls", URL `https://www.cybrary.it/0p3n/stateful-vs-stateless-firewalls/` (2016).

Morgan, S., "Cisco aci fabric simplifies and flattens data center network", (2014).

Nife, F. and Z. Kotulski, "Multi-level stateful firewall mechanism for software defined networks", in "Computer Networks", edited by P. Gaj, A. Kwiecień and M. Sawicki, pp. 271–286 (Springer International Publishing, Cham, 2017).

Nunes, B. A. A., M. Mendonca, X.-N. Nguyen, K. Obraczka and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks", IEEE Communications Surveys & Tutorials **16**, 3, 1617–1634 (2014).

O'Gorman, G. and G. McDonald, *Ransomware: A growing menace* (Symantec Corporation, 2012).

Ou, X., W. F. Boyer and M. A. McQueen, "A scalable approach to attack graph generation", in "Proceedings of the 13th ACM conference on Computer and communications security", pp. 336–345 (ACM, 2006).

Ou, X., S. Govindavajhala and A. W. Appel, "Mulval: A logic-based network security analyzer.", in "USENIX Security Symposium", pp. 8–8 (Baltimore, MD, 2005).

Pena, J. G. V. and W. E. Yu, "Development of a distributed firewall using software defined networking technology", in "Information Science and Technology (ICIST), 2014 4th IEEE International Conference on", pp. 449–452 (IEEE, 2014).

Peterson, J. L., "Petri net theory and the modeling of systems", (1981).

Roesch, M. *et al.*, "Snort: Lightweight intrusion detection for networks.", in "Lisa", vol. 99, pp. 229–238 (1999).

Rosen, R., "Linux containers and the future cloud", Linux J **240**, 4, 86–95 (2014).

Roy, A., D. S. Kim and K. S. Trivedi, "Cyber security analysis using attack countermeasure trees", in "Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research", p. 28 (ACM, 2010).

Roy, A., D. S. Kim and K. S. Trivedi, "Scalable optimal countermeasure selection using implicit enumeration on attack countermeasure trees", in "Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on", pp. 1–12 (IEEE, 2012).

Saini, V., Q. Duan and V. Paruchuri, "Threat modeling using attack trees", Journal of Computing Sciences in Colleges **23**, 4, 124–131 (2008).

Satasiya, D., R. Raviya and H. Kumar, "Enhanced sdn security using firewall in a distributed scenario", in "Advanced Communication Control and Computing Technologies (ICACCCT), 2016 International Conference on", pp. 588–592 (IEEE, 2016).

Scott-Hayward, S., G. O'Callaghan and S. Sezer, "Sdn security: A survey", in "Future Networks and Services (SDN4FNS), 2013 IEEE SDN For", pp. 1–7 (IEEE, 2013).

Sheyner, O., J. Haines, S. Jha, R. Lippmann and J. M. Wing, "Automated generation and analysis of attack graphs", in "null", p. 273 (IEEE, 2002).

Sheyner, O. M., "Scenario graphs and attack graphs", Tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE (2004).

Shin, S., V. Yegneswaran, P. Porras and G. Gu, "Avant-guard: Scalable and vigilant switch flow management in software-defined networks", in "Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security", pp. 413–424 (ACM, 2013).

Swiler, L. P., C. Phillips, D. Ellis and S. Chakerian, "Computer-attack graph generation tool", in "discex", p. 1307 (IEEE, 2001).

Tarjan, R., "Depth-first search and linear graph algorithms", SIAM journal on computing **1**, 2, 146–160 (1972).

Team, M., "Mininet: An instant virtual network on your laptop (or other pc)", Google Scholar (2012).

Vulnerabilities, C., "Exposures,the standard for information security vulnerability names", Common Vulnerabilities and Exposures: The Standard for Information Security Vulnerability Names. url: http://cve. mitre. org (2007).

Wikipedia contributors, "Lxc — Wikipedia, the free encyclopedia", URL `https://en.wikipedia.org/w/index.php?title=LXC&oldid=859715065`, [Online; accessed 9-October-2018] (2018).

Zerkane, S., D. Espes, P. Le Parc and F. Cuppens, "Software defined networking reactive stateful firewall", in "IFIP International Information Security and Privacy Conference", pp. 119–132 (Springer, 2016).

APPENDIX A

SOURCE CODE

For Stinger vulnerability scanner module, please visit:
http://gitlab.thothlab.org/ScienceDMZ/Stinger.git

For the rest of the code and implementation details, please visit my Github repository at:
https://github.com/hekmatbacha/MasterThesis