Software-defined Situation-aware Cloud Security

by

Ankur Chowdhary

A Dissertation Presented in Partial Fulfillment
of the Requirement for the Degree
Doctor of Philosophy

Approved November 2020 by the
Graduate Supervisory Committee:

Dijiang Huang, Chair
Subbarao Kambhampati
Adam Doupé
Youzhi Bao

ARIZONA STATE UNIVERSITY

December 2020

ABSTRACT

The use of reactive security mechanisms in enterprise networks can, at times, provide an asymmetric advantage to the attacker. Similarly, the use of a proactive security mechanism like *Moving Target Defense* (MTD), if performed without analyzing the effects of security countermeasures, can lead to security policy and service level agreement violations. In this thesis, I explore the research questions 1) how to model attacker-defender interactions for multi-stage attacks? 2) how to efficiently deploy proactive (MTD) security countermeasures in a software-defined environment for single and multi-stage attacks? 3) how to verify the effects of security and management policies on the network and take corrective actions?

I propose a *Software-defined Situation-aware Cloud Security* framework, that, 1) analyzes the attacker-defender interactions using an Software-defined Networking (SDN) based scalable attack graph. This research investigates Advanced Persistent Threat (APT) attacks using a scalable attack graph. The framework utilizes a parallel graph partitioning algorithm to generate an attack graph quickly and efficiently. 2) models single-stage and multi-stage attacks (APTs) using the game-theoretic model and provides SDN-based MTD countermeasures. I propose a Markov Game for modeling multi-stage attacks. 3) introduces a multi-stage policy conflict checking framework at the SDN network's application plane. I present INTPOL, a new intent-driven security policy enforcement solution. INTPOL provides a unified language and INTPOL grammar that abstracts the network administrator from the underlying network controller's lexical rules. INTPOL develops a bounded formal model for network service compliance checking, which significantly reduces the number of countermeasures that needs to be deployed. Once the application-layer policy conflicts are resolved, I utilize an Object-Oriented Policy Conflict checking (OOPC) framework that identifies and resolves rule-order dependencies and conflicts between security policies.

ACKNOWLEDGEMENTS

After completing my Master's degree at ASU, I developed a keen interest in pursuing research in the field of cybersecurity. While going for a Ph.D. was a long term commitment, I am happy that I decided to go for a doctorate degree.

Dr. Dijiang Huang was always encouraged me to pursue not only academic research but also explore many other possibilities around the field of cybersecurity that enriched my Ph.D. experience. I was able to compete in the Western Region Cybersecurity Defense Competition (WRCCDC), which helped me understand what it is like to deal with cyberattacks in a real-world scenario and helped me make my research work practical. After a fruitful research outcome, we co-founded CyNET LLC, a company to commercialize our research work on attack analysis and policy conflict checking. Dr. Huang also supported my efforts in teaching and mentoring students interested in security training and education. We co-founded the hacking club DevilSec last year to further this goal. It has since grown in popularity and helped many students achieve their academic and professional goals. I was able to gain the best out of my research and engineering skills under his mentorship. I would first and foremost like to thank him for mentoring me during my Ph.D.

I took Information Retrieval and Mining and Integration (IR) under Dr. Subbarao Kambhampati in Spring 2015. It was one of the best course work I attended at ASU. I was able to develop many research ideas based on knowledge gained from that class. I would like to thank Dr. Rao for being such a great instructor, helping improve my research work on Moving Target Defense by his valuable feedback, and most recently supporting my applications for positions in academia. I was fortunate to take up Software Security with Dr. Adam Doupé early in my Ph.D., which helped me build a knowledge base in the field of software security. I was also TA for one of the classes taught by him. It was great to work with him and learn from his teaching methodology. I am also grateful for his time in providing references for my academic job applications. Dr. Tiffany Bao is one of the few researchers I know who have an interest in the application of game-theoretical frameworks in the field of cybersecurity. After reading her work, I decided to take up her class in Spring 2019, and I was able to expand my research work from the domain of MTD to cyberdeception scenarios under her guidance. I was also able to work alongside her in mentoring cyberdeception capstone the same year, which provided me valuable mentorship experience. Unfortunately, with my research commitments, I was not able to keep up with the project, but I hope I can pick up the project again in the near future and help in advancing the research work on this project. I am also thankful to Dr. Gail-Joon Ahn, under whom I worked as a Research Assistant and helped in designing Science DMZ: an SDN-based secure cloud testbed. Dr. Ahn was also generous to support our teams that took part in WRCCDC competitions over several years. His dedication to the success of students involved in cybersecurity programs at ASU is truly admirable.

Research collaborations are very important in Ph.D. research. I have been fortunate to collaborate with most of the students from our lab but also from different research groups at ASU and other universities. I would like to thank Sandeep for helping me in research at the start of my Ph.D. James and Yuli have provided me valuable insights and helped me with system design and deployment, which helped in making my research work practical. I was able to collaborate with Adel and Sowmya on many research topics, and our joint work always progressed smoothly because of mutual respect and understanding of each other's thought process well. Hakim has helped me a lot in experimental design, and his hard work allowed us to produce high-quality results when there was a time crunch. Sailik was one of the most important collaborators I had, and his expertise in theoretical modeling, optimization algorithms, and AI helped take my MTD research to a new level. I would also like to acknowledge Vaibhav and Sukhwa from SEFCOM, who helped me in designing and deploying Science DMZ. More recently, I have been able to collaborate with Neha and Garima. They are both hard-working and always up to the task. I also had a chance to discuss research ideas with Duo and Zhen on some research projects, which helped in enhancing my knowledge. I mentored two Capstone projects and two honor's thesis project during the course of my Ph.D. I am quite lucky to have so many students show interest in my projects and glad that I was able to help them build up a successful career in cybersecurity. My teammates at DevilSec always inspired me with their enthusiasm for the cybersecurity as not just an academic field but as a lifestyle. My colleagues at Bishop Fox, CyNET LLC, Blackberry, Republic Services, and CSC helped me in times of need, be it a technical issue, discussion along with a research task, or business development (special thanks to Ken and Lance). My friends in both US and India, who always helped me during tough times, have been a backbone throughout my journey.

Lastly, I would like to thank my father for making me a disciplined person. I always looked up to him and learned the principle of simple living and high thinking. I thank my mother for teaching me both hard work and compassion, my sister, for her love and support, and my extended family for their encouragement. Communication with my family during testing times always helped me remain calm and composed. They made me the person I am, and to them, I owe a debt that I can never repay.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Chapter 1

INTRODUCTION

Network and cloud infrastructure has become both ubiquitous and more complex in the past few years. Multi-stage attacks, on the other hand, involve attackers compromising some services hosted at the network edge and progressing towards a high-value target located somewhere in the internal network, e.g., an SQL Server consisting of user information or an FTP Server which stores mission-critical information. A special category of multi-stage attacks known as Advanced Persistent Threat (APT) Alshamrani *et al.* (2019). These attacks involve stealthy attackers. The stealthy attackers use attack vectors that evade detection tools.

Modeling Multi-stage Attacks: In this thesis, I first model the multi-stage attacks using an attack modeling tool known as an Attack Graph (AG). The identification of information regarding vulnerability dependencies becomes increasingly difficult as the number of services and vulnerabilities increase in the network system. Ammann *et al.* (2002) proposed an AG generation approach with the scalability of the order $O(N^6)$. MulVAL Ou *et al.* (2005) reduces the AG generation and analysis complexity from $O(N^6)$ to $O(N^2)$ - $O(N^3)$, where N is the number of network hosts. The attack modeling framework that I discuss in this research thesis utilizes a parallel hypergraph partitioning algorithm Devine *et al.* (2006) to create a scalable attack graph that can be used for security analysis. Next, I consider one particular class of multi-stage attacks APT. While there are many proposed machine learning-based solutions that claim the detection of APT attacks, there is no good dataset that can be used to benchmark the ML models. Generic intrusion datasets used in APT research have several key limitations (1) they capture attack traffic at the external endpoints, limiting their usefulness (2) difference between normal and anomalous behavior are quite distinguishable in these datasets (3) data imbalance in existing datasets do not reflect the real-world settings. In order to address these limitations, I contributed a dataset *DAPT 2020* which consists of attacks that are part of Advanced Persistent Threats (APT) Myneni *et al.* (2020). As a part of this research, I benchmarked the DAPT 2020 dataset on semi-supervised models and show that they perform poorly trying to detect attack traffic in the various stages of an APT.

Attack Analysis and SDN-based Moving Target Defense (MTD): Moving Target Defense (MTD) House (2011); Schmerl *et al.* (2014) is a transformative approach to security of a multi-tenant cloud environment that leverages the dynamism in computer systems to create an environment that has a changing attack surface. In the second part of this thesis, I consider MTD as a solution to analyze the attack graph generated during the attack modeling phase. Traditional networks are composed of heterogeneous elements such as routers, firewalls, and switches. Each of these devices has its own proprietary software and protocols. This kind of network setup leaves very little scope for innovation in the network. Software-Defined Networking (SDN) Kreutz *et al.* (2015) has emerged as a solution for addressing this challenge. SDN environment generally consists of OpenFlow switches and controllers, communicating over a secure channel. SDN provides a service-oriented architecture to deploy modular solutions for different requirements. The SDN controller can work as a centralized security policy enforcer. The research work Ethane Casado *et al.* (2007) provides a model for user authentication and stronger binding between packets and origin. The programmable interfaces afforded by SDN can be conformed to achieve a dynamic defensive strategy based MTD Saha and Agarwal (2012); thereby providing a systematic solution by selecting countermeasures to prevent or mitigate attacks in an SDN enabled data center networking environment Chung *et al.* (2015);

Jafarian *et al.* (2012). Thus I used an SDN-based MTD solution for the deployment of optimal defense countermeasures for in the framework. To establish the usefulness and generalizability of the MTD countermeasures, I discuss both single-stage attacks - Distributed Denial of Service (DDoS) and multi-stage attacks - APTs.

Game Theory has proved to be very effective in economics, biology, and other areas for making important decisions. I formulated a Markov Game based approach to analyze multi-stage attack-defense interactions as two-player *zero-sum* game Chowdhary *et al.* (2018d). Sub-networks in the cloud network, determined using the system's Attack Graph (AG), represent the states of a multi-stage game. In particular, I analyze the problem of deploying a monitoring mechanism in a cloud network. The defense strategies take into account the long-term impacts of multi-stage attacks while ensuring that the defender picks a limited number of monitoring actions in each state of the game. The MTD countermeasure strategy in this framework ensures that the placement of detection mechanisms do not affect the performance asymmetrically in the different parts of the network. I worked on extending the proposed zero-sum game to a general-sum game, and demonstrate that for threat models where the adversary has knowledge of defender's strategy, the Stackelberg Equilibrium Basu (1995) provides an optimal strategy for deployment of defensive countermeasures. I modeled the APT attacks in both zero-sum and general-sum Markov games settings and showcased the optimal defense strategy obtained using optimal mixed strategy for zero-sum game Chowdhary *et al.* (2018c), and Strong Stackelberg Equilibrium (SSE) strategy in case of a general-sum game Chowdhary *et al.* (2018d); Sengupta *et al.* (2019) outperforms several other defense mechanisms based on pure strategy and uniform random strategies.

Security Policy Conflict Detection and Resolution: The dynamism provided by MTD gives rise to the need for a framework to accurately and in a timely fashion examine the complex relationships between various hosts and security vulnerabilities in an ever-changing cloud networking environment and ensure that any changes made to the environment do not conflict with security policies. Countermeasures selected as a part of MTD strategy could be based on link and network layer operations and reconfiguration, such as *a*) traffic redirection; *b*) traffic inspection; *c*) quarantining; *d*) MAC/IP addresses reconfiguration; etc. The countermeasures, if applied without proper analysis and validation in the network, can prove to be more catastrophic than useful.

Software-defined network (SDN) consists of a three-tiered architecture Kreutz *et al.* (2014) (a) application plane which is used by network administrator's, and the user's for defining high-level policies, (b) the control plane, which manages underlying network infrastructure and performs operations such as packet routing, and access control using well defined OpenFlow APIs, and (c) the infrastructure plane (data-plane) that consists of network switches and end-hosts. The security policies of the application-tier are intertwined with the OpenFlow rules when the underlying cloud network is managed by SDN. The dependencies between security policies can lead to security violations and network outages caused by misconfiguration of security policies. The results from a survey of enterprise middlebox deployments in the cloud indicate that 1) security violations and network outages caused by misconfiguration of security policies account for 60-70% failures in cloud enterprise middlebox deployments Sherry *et al.* (2012). 2) Mission requirement violations (SLA violation), i.e., service disruption/downtime for normal users because of forwarding loops and black-holes in the network as discussed by Khurshid *et al.* (2013). Thus, there is a need for verification of security and safety properties in a network before the deployment of a countermeasure in the form of a normal network security policy. It is also important to consider the network management policies within the scope of network policies. The safety and security properties of networks called *Network Invariants* need to be verified in a scalable fashion to ensure the smooth functioning of a network.

The realization of network policies at the data-plane level introduces a lot more flow rules. Detection and resolution of conflicts amongst flow rules at the data-plane level can impact the performance of network services. This problem is dramatically amplified when expanding SDN systems into multiple SDN domains, where inter-domain network and service policies introduce an additional level of complexity. In the third part of this thesis, I designed a multi-tiered policy conflict management framework. The Intent-driven policy language *INTPOL* allows the network administrator to express security policies at the application plane level. The policy designers can create network management and operation policies while remaining abstracted from the underlying SDN controller. The user-defined policies, as described above, are analyzed early in the security policy lifecycle to detect potential conflicts using a lightweight formal model checking framework. As a part of this policy checking model, the framework provides early detection of network policy conflicts at the application and control-plane. This approach will help reduce the scope of flow rule conflict checking at the data-plane level. Once the application plane policies are translated into network flow-rules, I perform a second stage flow-rule conflict detection and resolution at the data plane. The flow-rule conflict detection and resolution mechanism utilizes an object-oriented paradigm, referred to as Object-Oriented Policy Conflict (OOPC) checking. OOPC achieves 20% faster detection rate compared to the existing state of the art research work Brew Pisharody *et al.* (2017), and Flowguard Hu *et al.* (2014) on the Stanford dataset of 60k OpenFlow rules Kazemian *et al.* (2013).

1.1 Contributions

- This research provides a scalable solution for multi-stage attacks using attack graphs. I employ a parallel hypergraph partitioning mechanism for faster generation of large scale attack graphs Chowdhary *et al.* (2016); Sabur *et al.* (2019). I analyzed a special class of multi-stage attacks, Advanced Persistent Threats (APTs) and contributed a dataset DAPT 2020 Myneni *et al.* (2020) to benchmark machine learning models used for APT attack analysis.

- I utilized SDN-based moving target defense (MTD) solutions for single and multi-stage attacks. I devised a Markov game model to analyze attack defense scenarios in a cloud network. The defender's MTD countermeasure obtained by solving the Markov game optimizes the cost-incurred and security provided by detection mechanisms in a multi-tenant cloud network. The proposed framework is generalizable to both two-player zero-sum Chowdhary *et al.* (2018d,c) and general-sum Markov games Sengupta *et al.* (2019).

- I introduce INTPOL, a new intent-based language for implementing SDN countermeasures in a safe and efficient manner. INTPOL framework utilizes bounded model checking (BMC) a the application plane, thus reducing the scope of policy conflict checking at the data plane. My framework, OOPC performs the second level of flow-rule conflict detection and resolution at the data-plane Chowdhary *et al.* (2019b) to address the security and performance issues induced by conflicting security policies.

1.2 Organization

- Part I covers a scalable attack graph as a modeling tool for multi-stage attacks in a large network - Chapter 2. Advanced Persistent Threats (APTs), a special case of multi-stage attacks, and the dataset I generated for benchmarking APT attacks have been discussed in Chapter 3.

- Part II of the thesis comprises Chapter 4, which provides a survey of existing moving target defense mechanisms. This part emphasizes the solutions that utilize programmable networks (SDN) for the deployment of MTD. I model the multi-stage attacks using Markov Games in Chapter 5 and also discuss APT as a special case, where MTD works well compared to other proactive means of cyberdefense.

- Part III explores the effects of MTD countermeasures. I introduced an intent-driven formal model to analyze the effect of SDN policies at the application plane in Chapter 6, and extend the discussion on policy conflicts by considering flow-rule conflicts at the data plane in Chapter 7. I conclude the thesis in Chapter 8. The appendices provide details on APT dataset construction, proofs of formal modeling used for object-oriented policy conflict checking, and the other research works that I was not able to incorporate as part of this thesis.

Chapter 2

MODELING MULTI-STAGE ATTACKS USING SCALABLE ATTACK GRAPHS

In this chapter, I discuss the attack graph as a multi-stage attack modeling tool. I formalize the attack graph and hypergraph, introduce parallel hypergraph partitioning as a means for generating scalable attack graphs in a large cloud network. I consider existing research in the field of attack graph generation and their inherent limitations and use empirical evaluation to establish the scalability of the graph generation algorithm. I also discuss the use of micro-segmentation and distributed firewalls as a means of managing the complexity of the attack graph generation process.

2.1 Attack Analysis Model

In the attack model for this work, I assume that an attacker can be located either outside or inside of the virtual networking system. The attacker's primary goal is to exploit vulnerable Virtual Machines (VMs) and compromise them as zombies. The protection model in this research focuses on virtual network-based attack detection and reconfiguration solutions to improve the resiliency to zombie explorations. The proposed solution can be deployed in an *Infrastructure-as-a-Service* (IaaS) cloud networking system, where the *Cloud Service Provider* (CSP) is benign. I also assume that cloud service users are free to install any operating systems or applications, even if they are known to be from adversarial sources.

Let's consider an OpenStack Sefraoui *et al.* (2012) based cloud networking environment Figure 2.1. The components in the figure are OpenStack modules responsible for various functions. Nova is responsible for VM provisioning and management. Neutron provides network control. The Firewall-as-a-Service (FaaS) has been installed

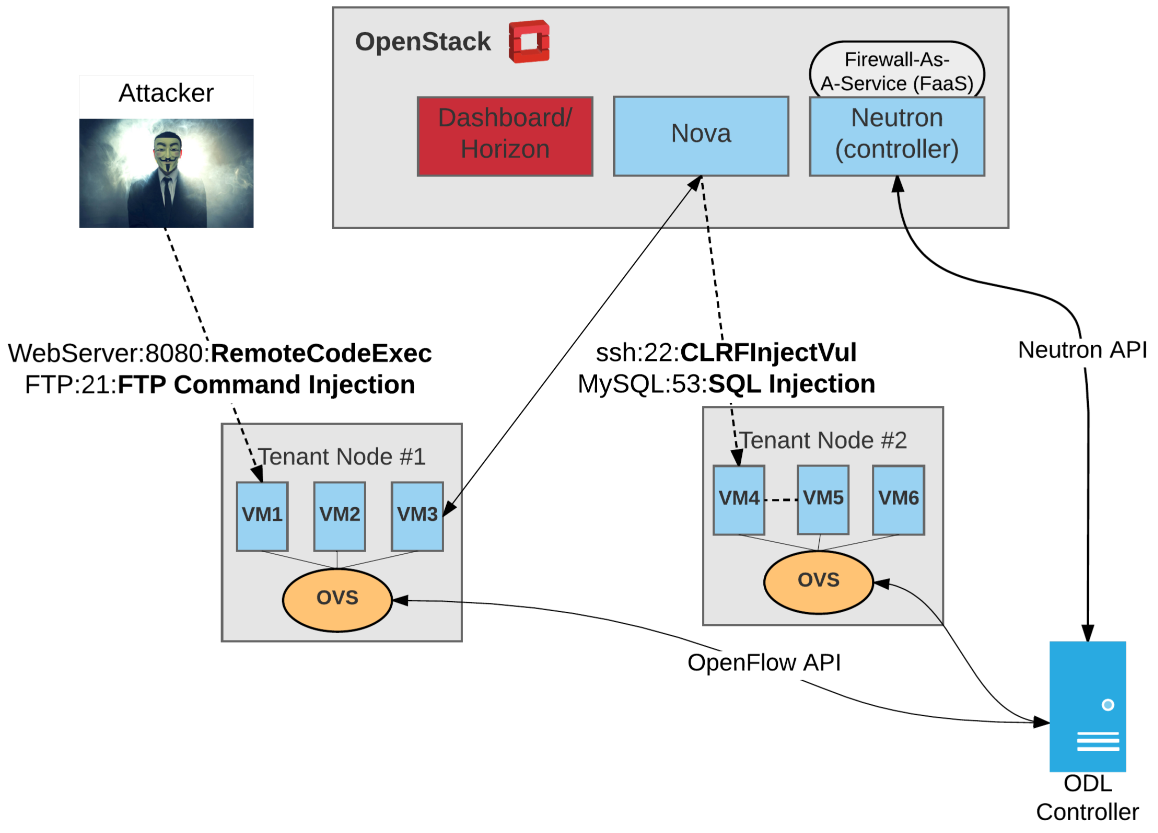


Figure 2.1: Threat Model in a SDN-managed Openstack Cloud Network

on top of Neutron. Neutron interacts with OpenDaylight (ODL) controller through a REST API. Various firewall operations at layer-2 or layer-3 in this overlay network can be deployed to VM's ODL policies via Neutron (OpenStack network manager).

Based on the assumption that one VM of tenant 1 has a web server running and another VM of a tenant, 2 has a database server running. If the goal of the attack is to compromise the database server, the attacker can first compromise the web server of tenant 1 via remote code execution or FTP based vulnerability. Once he/she has access to the web server, he/she can use a web server as a zombie VM and compromise the database server of tenant 2 using SQL injection or SSH CLRF injection vulnerabilities.

2.1.1 Attack Graph Formalism

Attack graphs (AG) are a good tool to represent the security state of the entire network. AGs have proved to be a very useful tool to detect multi-stage and multi-hop attacks, which may not be obvious to the network administrator by plain analysis. Some of the earlier works in this field have used model checking Sheyner and Wing (2003); Sheyner (2004) and formal language-based methods Sheyner (2004) to enumerate all possible attack scenarios in the cloud system. I utilized an AG based analysis in the threat model for this research work. A node N in an AG is a combination of hosts in the environment and the possible vulnerabilities that exist on that particular host. The attack graphs (AG) can be formally defined as follows:

Definition 2.1.1 *An Attack Graph (AG) is a tuple* $G = \{S,\ \tau,\ s_0, s_t\}$ *where* $S = N$ *is number of states possible,* $\tau \subseteq S \times S$. $s_0 \subseteq S$ *is set of initial states and* $s_t \subseteq S$ *is set of success states.*

- $V = N_C \cup N_D \cup N_R$ *denotes a set of vertices that includes conjunction nodes* (N_C) *to represent exploits, disjunction node* (N_D) *denoting results of exploit, and root node* N_R *denoting the initial step of an attack scenario.*

- $E = E_{pre} \cup E_{post}$ *denotes the set of directed edges. An edge* $e \in E_{pre} \subseteq N_D \times N_C$ *denotes that* N_C *can be attained only if* N_D *is satisfied. An edge* $e \in E_{post} \subseteq N_C \times N_D$ *means that* N_D *can be obtained if* N_C *is satisfied.*

Second, in order to perform scalable attack graph generation, I used the hypergraph partitioning algorithm - ParMETIS Devine *et al.* (2006), which converts the vulnerability and reachability information from the network environment into parallel *sub attack graphs* (SubAGs).

Definition 2.1.2 *A hypergraph is a generalization of a normal graph $H = (V, E)$. V represents vertices $V = \{v_i | 1 \leq i \leq n\}$ and $E = \{e_j | 1 \leq j \leq m\}$ represent the hyperedges. A hyperedge is a subset of vertices. The degree of vertex is the number of hyperedges it is part of. The neighborhood of a vertex $N(v)$ refers to vertices directly incident on same edge as vertex* v.

The performance analysis of existing graph partitioning algorithm shows that hMETIS (METIS algorithm without parallelization) can compute a partition of large hypergraph (3.5 million vertices) in *20 mins* on a *32-bit*, *4GB* RAM machine. This motivated the use of ParMETIS (a parallelized version of hMETIS) for creating of large attack graphs by first partitioning attack graphs based on reachability information, and then utilizing Spark-based data structures Resilient Distributed Datasets (RDD) Zaharia *et al.* (2012) for keeping track of post-conditions across *SubAGs*. The RDD information is utilized to merge SubAGs and generate a full attack graph. The experimental results on Mininet (2015) using an ODL controller shows that it takes over twenty minutes to generate an AG for a network of ten thousand nodes for the current attack graph generation framework MulVAL Ou *et al.* (2005). Our graph generation procedure in this framework scales well on a large network - $\sim$ 7s for an attack graph with 11k vulnerabilities. This shows significant performance gain when utilizing a hypergraph partition and merge approach, as described in Chowdhary *et al.* (2016). The hypergraph partitioning can be defined as follows:

Definition 2.1.3 *Hypergraph partitioning is the process of partitioning a hypergraph into k-way ($k \geq 2$) disjoint set of node blocks B_1, B_2, B_k such that $B_i \cap B_j = \emptyset\ \forall i, ji \neq j$. Thus, the goal is to find a k-way mincut in a hypergraph.*

I utilized the SDN based ODL controller in cloud infrastructure to manage the complexity of assessing the security state of the entire network. An MTD solution for such a large network will face scalability challenges like state space explosion, something the proposed hypergraph partitioning scheme effectively addresses.

The framework partitions the large AG into Sub Attack Graphs (SubAGs). The partitioning scheme uses the SDN controller as a driver program. The driver program coordinates with the sub attack graph creation modules known as SubAG Agents to construct a full AG. This helps in fast real-time attack scenario analysis. The process of the hypergraph partitioning based AG creation reduces the time to construct full AG. The AG constructed helps in MTD countermeasure selection, which will be discussed in Chapter 6 in real-time. For instance, in Figure 2.2, the hosts are partitioned into two SubAGs, each shown with a different color. Services SSH and FTP on hosts VM1, VM2, VM3, and ISCSI, SSH, and MySQL on VM4, VM5, VM6 represent two partitions for the AG. The key idea is to distribute a load of SubAG creation over several processors for each tenant and then check reachability links across the tenants. The attacker in Figure 2.2 has root access on VM1, so he is able to exploit service MySQL based vulnerability on VM4.

The proposed algorithm merges reachability information across SubAG's to get the final AG. The number of processors depends upon the Cluster on which OpenStack is deployed. For instance, the HP Blade Server Cluster with 10 CPUs has been allocated to the task of partitioning, with 2 processors per CPU, the value of the number of processors used in partitioning procedure is 20.

Attack graphs are also useful in identifying the critical assets in the network. For instance, Asset Rank Sawilla and Ou (2008) based approach can be used to rank critical assets in a network, which are more likely to be affected as a result of network vulnerabilities. The framework prioritizes these assets for selecting countermeasures. The attack graph-based approach is very effective in handling dynamic attacks such as DDoS and multi-stage attacks such as Advanced Persistent Threat (APT). If a

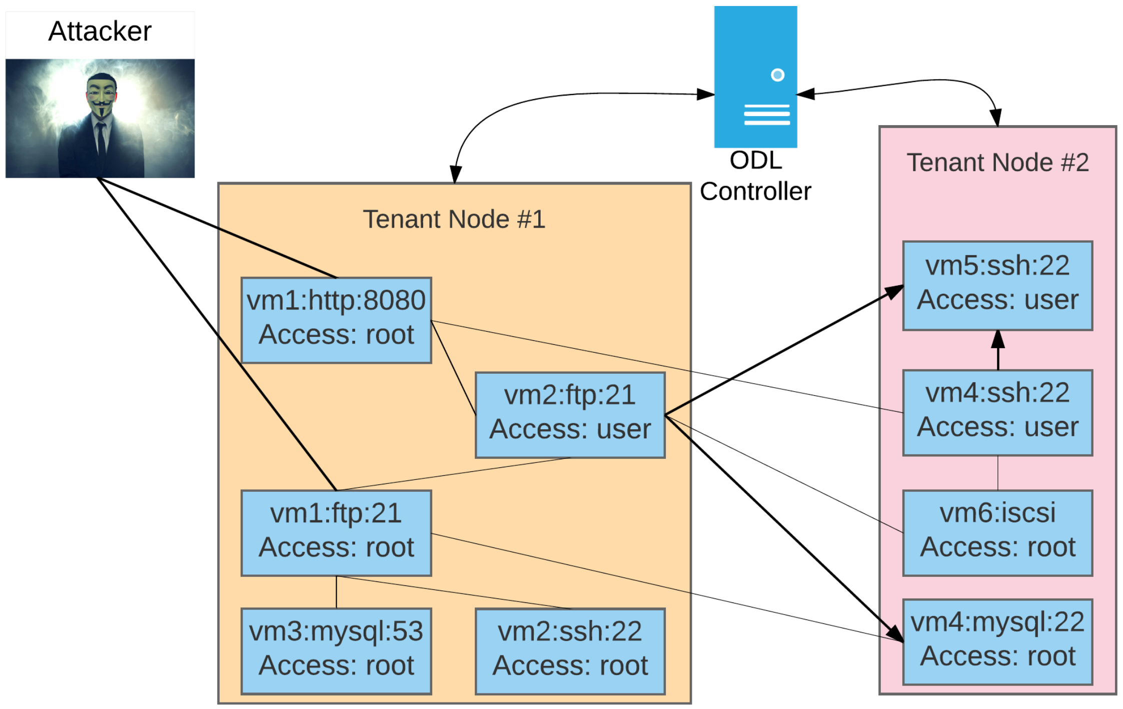


Figure 2.2: A Network Reachability-based Partitioned Attack Graph

botnet server communicates with clients to target a system resource, this information can be modeled using Attack Graph.

Most modern vulnerability scanners report only the known vulnerabilities in the network, such as *remoteCodeExecution, localBufferOverflow* etc. An attacker, however, on gaining privileges on a machine can install malicious applications that can trigger zero-day vulnerabilities. For instance, in the Figure 2.2 VM1, which is Web-Server, maybe providing some web service to VM2. Once an attacker has gained root access on VM1, he can downgrade a patched software, e.g., flash player, to a vulnerable version and try to compromise VM1. These unknown security risks can also be modeled using Attack Graphs.

The corresponding attack graph for the threat model Figure 2.1 has been shown in Figure 2.3. As can be seen from Figure 2.1, the threat model under consideration has four nodes, i.e., the web server running on VM1, the database server on VM4,

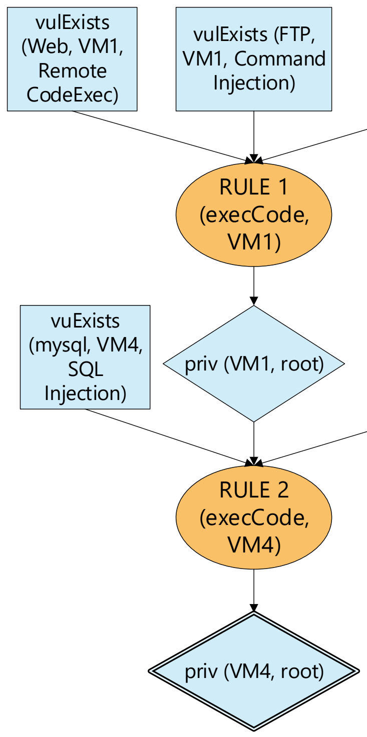

vulExists (Web, VM1, Remote CodeExec)
vulExists (FTP, VM1, Command Injection)
priv (VM1, User)
RULE 1 (execCode, VM1)
vuExists (mysql, VM4, SQL Injection)
priv (VM1, root)
vuExists (ssh, VM4, CLRF Injection)
RULE 2 (execCode, VM4)
priv (VM4, root)

Figure 2.3: Attack Graph Generated for the Cloud Network

and the attacker as `A`. An attacker could chain the exploits as follows:

- `execCode(A,VM1,8080)` - Remote Code exploit.
- `execCode(A,VM1,21)` - FTP vulnerability exploit.
- `execCode(VM1,VM4,22)` - CLRF inject vulnerability exploit.
- `execCode(VM1,VM4,53)` - SQL injecttion exploit.

The code execution on FTP or Web Server can allow an attacker to obtain a post-condition, i.e., *root* level privilege on VM1. Similarly, the implicit assumption is that VM1 and VM4 can communicate over the internal network. Moreover, VM4 has an SSH server and SQL server with vulnerabilities. Thus an attacker can exploit these vulnerabilities in stage-2 of the exploit and obtain his goal, i.e., *root* privilege on VM4 as shown above.

2.2 Scalable Attack Graph Generation

In this section, I discuss the implementation details for modules that comprise a scalable attack graph generation framework. The overall architecture of AG generation involves the creation of reachability and vulnerability-based scalable attack on each tenant. The decision on the creation of the attack graph is based on the size of the number of nodes vulnerabilities in the network.

For example, if three processors are allocated by the admin, then each processor will take care of the creation of a `Sub-Attack Graphs` (SubAG), and we will have three SubAGs. The framework uses parallelism based on PySpark pys (2016) framework to merge the SubAG for each tenant. `Resilient Distributed Datasets` (RDDs) Zaharia *et al.* (2012) are parallel data-structures that are used for information exchange in distributed networks. RDDs exchanges post-conditions (result of a successful exploit) generated by each SubAG and attack tenant AG till no new post-conditions are left.

In Algorithm 1, procedure `Generate-SubAG` checks all vulnerability and reachability edges for each `SubAG`. The pre-conditions of the AG are the requirements for an exploit to be successful. The post-conditions, on the other hand, mean privilege gained on a successful exploit, e.g., root access. Since post-conditions act as triggers or pre-condition for another vulnerability, all new post-conditions are written to the RDD. The agents or processors can check post-conditions specific to their SubAG and update the SubAG if necessary.

The procedure `Part-Hypergraph` of the algorithm 1 is responsible for creating a hypergraph and partitioning the hypergraph-based on parallel hypergraph partition-

Algorithm 1 Scalable AG Generation Algorithm

```
1: $AG \leftarrow \emptyset$
2: procedure PART-HYPERGRAPH($\mathrm{G}, \mathrm{k} \geq 2, \mathrm{p}, \mathrm{H} \leftarrow \emptyset$)
3:     $\mathrm{V} \leftarrow v_1, v_2, ...., v_n$
4:     for $i = 1 \rightarrow$ (n) do
5:         $\mathrm{H} \leftarrow H \cup \{v_i, N(v_i)\}$
6:     end for
7:     $\mathrm{edgecuts, parts} \leftarrow$ ParMETIS$(H, k)$
8:     return SubAG
9: end procedure
10: procedure GENERATE-SUBAG($SubAG$)                ▷ SubAG Generation Phase
11:     $EN$ = Find_Edge_Nodes($SubAG$);
12:     for each $e \in EN$ do
13:         $Attackers$ = Find_External_Nodes_with_priv($e$)
14:     end for
15:     $Vulns$ = Find_Vulns_Info($SubAG$)
16:     $Reaches$ = Find_Reach_Info($SubAG$)
17:     $SubAG$ = Create_SubAG($Attackers, Vulns, Reach$)
18:     while $true$ do
19:         $PostConds$ = Find_New_PostConds($SubAG$)
20:         for each $p \in PostConds$ do
21:             Write_To_RDD(Agent, $p$)
22:         end for
23:         $PostConds$ =Read_NewPostCond(Agent)
24:         if $PostConds.size() == 0$ then
25:             $break$
26:         end if
27:     end while
28: end procedure
29: procedure GENERATE-FULL-AG(SubAG[])             ▷ Attack Graph Merge Phase
30:     $AG \leftarrow SubAG[]$
31:     for each $e_i \in Read\_From\_RDD$ do
32:         if $e_i = \{SubAG_i, SubAG_j\}$ i $\neq$ j then
33:             $AG \leftarrow AG \cup e_i$
34:         end if
35:     end for
36: end procedure
```

ing algorithm ParMETIS Karypis and Schloegel (2013). The algorithm takes graph connectivity information represented by G as input, along with a number of processors p, a number of partitions required k, and an empty data structure for hypergraph H. The vertices in the neighborhood of the vertex $N(v)$ are part of the hyperedge.

The hypergraph H is then partitioned into regions based on the number of tenants, as shown in Figure 2.2 using the partitioning algorithm. $SubAG[n_i]$ will return partition to which node n_i belongs to after partitioning. For example, in Figure 2.2, `SubAG[vm1:http:8080]` will return `Tenant Node 1`. Each processor p is responsible for the construction of a SubAG or cluster for a given tenant. This distribution of load for AG construction is done using a PySpark pys (2016) based framework which maps SubAG construction phase over the p processors. Hypergraph partitioning process aims to find k-way min-cut in a Hypergraph, with $k > 1$. Consider a variable ϵ such that $0 < \epsilon < 1$. The objective of partitioning algorithm is to construct a partition set $\pi = \{B_1, B_2, .., B_k\}$ from the hypergraph. The cost function for partitioning algorithm is $f_o(\pi, E)$ where E represents the hyperedges. In the case of my proposed solution, the cost is the run-time for the algorithm. The weight function for the partitioning algorithm is $f_w(\pi)$, which is the cumulative vulnerability score for each partition in the design. The weight of each partition can be fetched from a weight function by relation $W_i = f_w(B_i)$. The average vulnerability score of all partitions is W_{avg}. The goal of the algorithm is to optimize f_w and f_o, such that $W_i < (1+\epsilon)W_{avg}$ Trifunovic (2006).

The partitions generated from each tenant are taken as input $SubAG$ for Algorithm 1. Since the algorithm checks the RDDs for updated information, the AG generation process makes sure new information such as nodes leaving the system or VMs migrating from one physical server to another are reflected in real-time.

Next, in Algorithm 1 procedure `Generate-Full-AG`, the algorithm merges the individual SubAGs. The input for the algorithm is an array of individual SubAG's. The edges that are part of bipartite graph G' between two SubAG contain edges such that $e_i \in \{SubAG[v_i], SubAG[v_j]\}$ and $SubAG[v_i] \neq SubAG[v_j]$. This means bipartite Graph $G' = SubAG_i \cap SubAG_j$. The proposed solution makes use of RDD to update edges that belong to such bipartite matching. Since there will be only a finite number of edges that will be part of such matching, the framework will be able to do the updates in linear time.

2.3 Experimental Analysis

The experiments were run on an Intel i7 based cloud system. The host Operating System was Ubuntu 14.04. First, I evaluated the dependency of time to generate AG to the number of hosts and their connectivity. I conducted the test on a MulVAL file with $1,700$, densely connected hosts. The implementation of AG clusters/partitions generation involves the use of the PySpark based parallelization framework.

Each SubAG is generated by one processor in an Apache Spark framework. I computed the time to construct a scalable AG for a fixed number of partitions while increasing the number of nodes from $1,700$ to $11,500$. The time to construct scalable AG increases as expected, going from around 9 seconds for $5,600$ nodes to about 28 seconds to generate AG for $11,500$ nodes. This drastic increase in time is because the graph is densely connected, and the number of interconnections increased as the number of nodes were scaled up. The time to construct an AG is still less than half a minute, which is acceptable considering the number of reads/write operations involved and the merging phase of the different AG partitions.

Next, I conducted an experiment to measure the time required by the various parts of the algorithm, such as time to partition graph (`PartGraph`), time for the SubAG

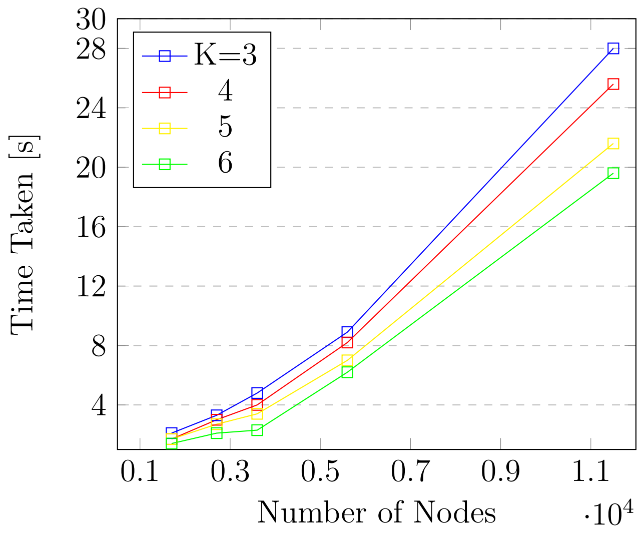


Figure 2.4: AG Generation Time vs Number of Nodes

generation (`Generate-SubAG`) and time for merging the smaller subgraphs into a large connected AG (`Generate-Full-AG`). The results are shown in Table 2.1. The size of

Table 2.1: Time Based Complexity Analysis of Algorithm

Parts K	**AG Part**	**SubAG Gen**	**AG Merge**	**Total Time**
3	0.30	6.28	0.24	6.82
4	0.35	7.76	0.29	7.914
5	0.36	7.84	0.43	8.27
6	0.37	10.16	0.48	10.64
7	0.38	7.8	0.44	8.25
8	0.36	10.09	0.57	10.56
9	0.36	10.32	0.68	10.8

AG is $N = 10,000$. The network is sparsely connected. The time taken by various phases of the algorithm doesn't include the time taken to create the input SubAG, which is produced by MulVAL. I started by generating a graph, which is dependent on the number of edges. The time taken to generate the graph will be fixed if the number of nodes are also fixed. The SubAG generation takes the most amount of time, and this time increases with the increase in the number of partitions K, from $K = 3$ to $K = 6$, which can be attributed to the time involved in spawning separate agents for each SubAG. For $K = 7$, the time decreases for SubAG generation, which is due to the performance gain obtained due to parallelization. For $K = 8$ and $K = 9$, the SubAG generation time increases again. K can thus be selected based on numerous iterations of the algorithm. The merge time for the algorithm shows a constant increase, which is expected except for an outlier at $K = 7$. The overall performance gain in this network configuration due to the use of spark based distributed structure is significant. Thus, the solution can serve as the best model for large, densely, or sparsely connected networks.

2.4 ATTACK GRAPH GENERATION RESEARCH

The attack representation methods have suffered from scalability issues. The model checking approach proposed by Amman *et. al.* uses counterexamples as a means to check the security policies Ammann *et al.* (2005). This approach will suffer from scalability issues, as path explosion is often an issue with model checking. Ingols *et al* Ingols *et al.* (2006) proposed multi-prerequisite graph-based security assessment. The attack graph scales linearly with the network size. The computational complexity, as described in the paper, is $O(E + NlgN)$. The balance between completeness of attack representation and scalability of approach is a widely researched area. *Attack Graph distillation* Huang *et al.* (2011) uses severity metrics to choose most critical attack paths. This helps the administrator control the information presented. Their paper utilizes a Minimum-Cost SAT solving approach to identify the most critical attack paths for the attacker to launch multi-step attacks.

Jha *et al.* (2002) presents a formal analysis of attacks on a network along with cost-benefit analysis and security measures to defend against the network attacks. Ammann *et al.* (2002) utilized the assumption of monotonicity to present an attack graph generation solution which achieves a complexity of $O(N^6)$. This was better compared to earlier research, however not a scalable solution for a large network. Logical representation of attack information such as host connectivity and network vulnerability is a popular method used by many research works to address the state-space explosion problem associated with Attack Graphs Ou *et al.* (2005). A hierarchical attack representation model that utilizes the network structure to group the nodes of the attack graph has been proposed by Hong and Kim (2013). A shared memory based optimization approach for the generation of a distributed attack graph has been proposed by Kaynar and Sivrikaya (2015). The time required for the generation of an attack graph when then the number of hosts is 450 is 2-3 minutes. This limits the practical application of the attack graph for real-time analytics. A parallel attack graph decomposition approach for the fast-generation of scalable attack graph has been proposed by Mjihil *et al.* (2017). The network was tested for 50 vulnerabilities and does not consider the dependency between different services when generating attack graphs, which is considered in this research.

2.5 SUMMARY AND DISCUSSION

The parallel hypergraph partitioning approach considered in this research identifies the dependency between different services, utilizes the network structure and service dependencies to abstract the graph edges as hyper-edges, and the parallel graph partitioning approach to partition the task of generating a large attack graph into manageable sub attack graphs. The order of generating an attack graph with $\sim$10k network nodes is $\sim$20-30(s). This is significantly better than prior research works in the field of scalable attack graph generation. I expanded the attack graph generation research described in this chapter to further optimize the attack graph generation process in my research, S3 Sabur *et al.* (2019). S3 utilized an SDN-based distributed firewall (DFW) Pena and Yu (2014) for managing the service reachability between different segments of a cloud network. The pre-computation of segment-specific sub attack graphs and tracking of changes in network topology helps avoid the cost of re-computing sub attack graphs. Additionally, the S3 framework helped in managing the security policies for cloud hosts, services, and network segments at a granular level, and in effect, limiting the lateral movement (one of the key phases of APT attacks).

Chapter 3

ADVANCED PERSISTENT THREAT (APT)

In this chapter, I describe APT attacks in detail, identify their key characteristics. Next, I detail models prevalent in APT detection, with emphasis on semi-supervised machine learning models. I elaborate on limitations of both existing APT detection models and benchmark datasets and present an APT dataset DAPT2020 Myneni *et al.* (2020), which covers different APT phases, and attack vectors missing in current research. This chapter's empirical evaluation serves as a motivation for proactive defense mechanisms such as Moving Target Defense (MTD) to deal with APT attacks, discussed in the next chapter.

Advanced Persistent Threats (APT's) are stealthy attacks mounted by a sophisticated group of attackers often sponsored by large organizations or governments to gain useful information about the target organizations. APT is a combination of three words Ross (2011) namely,

Advanced: The APT attacks are well-funded and use *advanced* mode of operations, sophisticated tools, as opposed to regular information discovery tools used by individual attackers. The advanced tools employ multiple attack vectors, and the target organization in the case of APT is often a highly valued target.

Persistent: The group of attackers in the case of APT are highly motivated and *persistent.* Once the attackers gain access to the system, they try to access connected systems without raising security tool alarms. The attackers employ several evasive techniques and follow *slow and low* approach to increase the chances of success.

Threats: The *threat* in case of an APT attack is a loss of data or critical information that can disrupt an organization's everyday operations, loss of reputation, and

Table 3.1: Comparison of Traditional and APT Attacks.

	Traditional Attacks	**APT Attacks**
Attacker	Mostly single person	Highly organized, sophisticated, determined, and well-resourced group
Target	Unspecified, mostly individual systems	Specific organizations, governmental institutions, commercial enterprises
Purpose	Financial benefits, demonstrating abilities	Competitive advantages, strategic benefits
Approach	Single-run, “smash and grab”, short period	Repeated attempts, stays slow and low, adapts to resist defenses, long term

mission-critical data. These threats are difficult to detect and require sophisticated defense mechanisms to detect and prevent.

The APT lifecycle, according to the well-established APT models, can be split into five core phases. (1) *Reconnaisance* - scanning the network for identification of vulnerabilities, gathering information about organization and employees, which helps the attacker in understanding the attack surface (2) *Establishing Foothold* - exploiting known or unknown vulnerabilities to gain initial access on one of the network. The attacker can also use targeted phishing campaigns to infect a device on the network organization. (3) *Lateral Movement* - targeting network resources which are mission-critical such as internal Active Directory or FTP server (4) *Maintain access* - using tools and techniques to ensure that even if connection breaks, it can be established again quickly. A sophisticated attacker can deploy backdoors on the compromised system to establish communication with the command and control center (C&C) when required. (5) *Data Exfiltration* - sending important data and files to a remote server under attacker’s control, e.g., a remote FTP server, remote shared drive, etc. The number of phases has been further sub-categorized by Mandiant McWhorter (2013) (foothold establishment, and privilege escalation - phase 2). Similarly, Ussath *et al.* (2016) abstract the APT life cycle into three phases.

The data-collection and feature selection described next consider host-level logs (establishing foothold and privilege escalation) as one. The key reason is that host-level records for both stages use log generation sources (audit logs in Linux, Windows Logon events in Windows, access logs for web and database services).

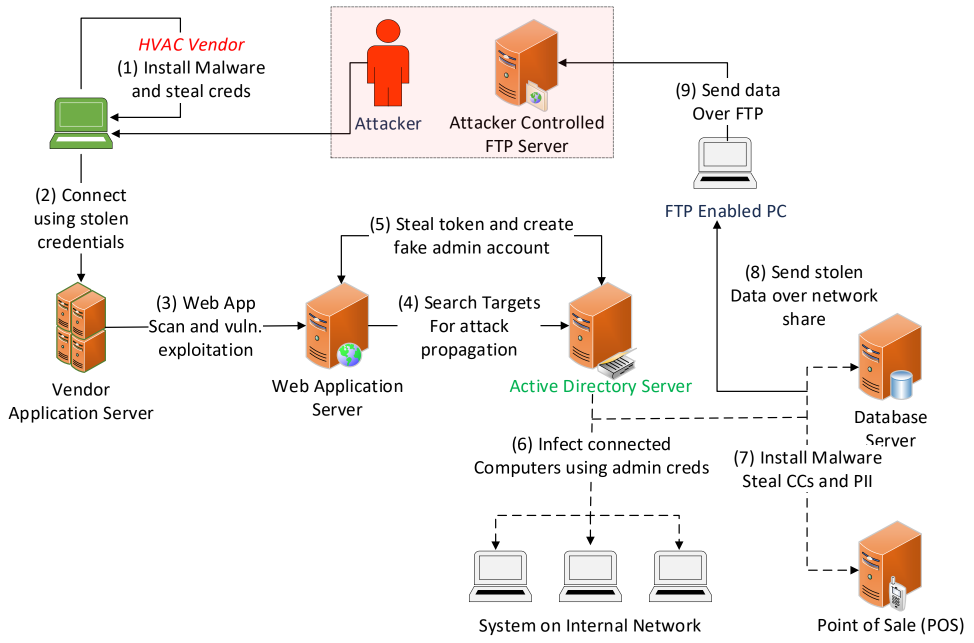

HVAC Vendor
(1) Install Malware and steal creds
Attacker
Attacker Controlled FTP Server
(9) Send data Over FTP
FTP Enabled PC
(2) Connect using stolen credentials
(5) Steal token and create fake admin account
(8) Send stolen Data over network share
(3) Web App Scan and vuln. exploitation
(4) Search Targets For attack propagation
Vendor Application Server
Web Application Server
Active Directory Server
Database Server
(6) Infect connected Computers using admin creds
(7) Install Malware Steal CCs and PII
System on Internal Network
Point of Sale (POS)

Figure 3.1: Different Phases of APT Attacks for Target Network. The Attacker utilized Vendor PC to Infiltrate vulnerable Web Server, and Weak Authorization and Monitoring Capabilities to Exfiltrate Personally Identifiable Information (PII) and Credit Card (CC) Information over FTP.

For illustrating the real-world APT attack, the example above provides details of the Target APT attack as a use case in Figure 3.1. The initial penetration point - step (1) for the attacker located outside the network was Heat, Ventilation, and Air Conditioning (HVAC) vendor PC. The attacker could steal HVAC access credentials and gained access to the Target network's target web services - steps (2), (3). The attacker performed internal `reconnaissance` to identify the target machines which were connected to Active Directory (AD) Server, and steal their credentials - steps (4), (5) - `footlhold establishment`. The attacker utilized `lateral movement` on the internal network to target Point of Sale (POS) systems. The attacker also created fake admin accounts on the AD server to `maintain access`. The attacker deployed `Kaptoxa` malware on the POS systems using compromised internal machines - steps (6), (7). The attacker was able to obtain database access using elevated privileges on the POS system. Next, the attacker used the stolen Personally Identifiable Information (PII) and Credit Card (CC) information from POS and database server and sent them to a centralized system in the Target's network using standard Windows SMB protocol - step (8). In the last phase of the attack, the data was `exfiltrated` by the attacker from a centralized repository to the attacker's controlled FTP server - step (9).

3.1 Models Used for APT Detection

One poring APTs attack patterns as outliers (anomalies)pular approach used for APT detection is conside and utilizing the outlier detection methods for detecting APT attacks. Some popular outlier detection methods, as discussed by Hodge and Austin (2004), include unsupervised clustering. This approach utilizes the static distribution and identifies the data points outside the normal data distribution range as anomalous. The second method involves the supervised classification of new data based on pre-labeled data as normal or abnormal. The third category includes the use of semi-supervised models that considers pre-classified data to model standard data. As the new data is received, the model fine-tunes and refines the normal vs. abnormal data boundary. Authors also utilized machine learning methods to categorize anomaly detection methods, particularly the use of neural networks and statistical anomaly detection techniques. Chandola *et al.* (2009) considered a problem associated with anomaly detection when discrete sequences are in consideration. They utilized sequence-based anomaly detection and pattern-frequency-based anomaly detection methods for addressing issues related to discovering anomalies in a discrete domain. Use of machine learning methods for anomaly detection has been detailed by Mehmood *et al.* (2013). The authors considered widely used methods such as Support Vector Machines (SVM) Noble (2006), Fuzzy Logic (FL) Klir and Yuan (1995), Genetic Algorithm (GA), K-means approach Ding and He (2004), Artificial Neural Networks (ANNs) Yegnanarayana (2009), and association rule matching for detection of anomalies. The use of anomaly detection methods in APT detection research can be quite useful. The use of machine learning methods for anomaly detection has been detailed by Mehmood *et al.* (2013). The use of machine learning models for detecting normal and abnormal behavior of the system over time to identify APT attack patterns has been discussed by Friedberg *et al.* (2015). The authors used the system logs produces by different components of the Instrumentation Control Technology (ICT) network to identify anomalous events and event classes. However, this approach does not scale when considering attributes with different values for the given data. The machine learning methods used for the detection of APT attacks can help in the identification of contextual Hayes and Capretz (2015) and collective anomalies Zheng *et al.* (2015), that are characteristic of many APT attacks. The event classes identify implications between different events and detect emerging APT attack patterns. Cappers and van Wijk (2016) utilized a machine learning approach for contextual analysis of network traffic alerts and splitting them into messages and attributes. Anomaly detection for web-based log data has been considered by Qu *et al.* (2018). The research work gated-recurrent unit (GRU) as a basic unit trained using unsupervised machine learning models for detecting anomalies. The authors compared their model with Support Vector Machines (SVM) and Long Short-Term Memory (LSTM) Hochreiter and Schmidhuber (1997) methods. Du *et al.* (2017) utilized LSTM approach for detecting DDoS attacks. The authors used a model trained on the normal pattern of system log events to detect abnormal traffic patterns present in the case of DDoS attacks.

3.1.1 Semi-Supervised Models for Detection of APT Attacks

Semi-supervised machine learning models consider regular traffic as a baseline, and any deviation from the baseline is anomalous Borghesi *et al.* (2019). Network traffic data depicts a class imbalance pattern, i.e., the percentage of regular traffic is very high compared to abnormal traffic. Semi-supervised machine learning models are robust to such instances of class imbalance. I consider three semi-supervised models that have been used by research works on APT attack detection.

One-Class Support Vector Machines (1-SVM)

1-SVM presents a useful model in the instances of high-class imbalance, i.e., when the percentage of regular traffic is large compared to abnormal traffic Emmott *et al.* (2013); Microsoft (2019). Thus, I utilized one-class SVM trained on regular network traffic. In this work, I used reconstruction error Machiraju and Yagel (1996) to classify network events as anomalous.

Stacked Auto Encoder (SAE)

The attack patterns in the case of APT depict some latent activities which are difficult to discover by traditional machine learning models. SAE, a special kind of feed-forward neural network is used to find these compact latent space representations of input, which can, in turn, allow reconstruction of attack patterns. A deep neural network is used by SAE for compression, followed by reconstruction, instead of single-layer neural networks. The loss function in SAE seeks to minimize the distance between original input and reconstruction output. The SAE model can be trained on standard data and tested on both normal and abnormal data. SAE mimics the input data, it effectively reconstructs the normal data, but in anomalous data, the reconstruction error is high. It allows the SAE model to detect the abnormal data by comparing reconstruction error to a predefined threshold.

Stacked Auto Encoder with Long Short-Term Memory (LSTM-SAE)

LSTM-SAE is a popular model in time-series forecasting problems. SAE by itself is not able to detect contextual anomalies Jiang *et al.* (2014). These anomalies are significant in APT attacks since different activities such as reconnaissance, foothold establishment, etc. are dependent on each other for attack progression. LSTM-SAE addresses this issue using LSTM cells instead of hidden layers cells present in SAE-based neural networks. In effect, LSTM compresses the network traffic packets in multiple time-steps and then reconstructs them. The multi-step attacks, such as APT, which showcase a pattern of attack activities dependent on each other over time, can be detected using the LSTM-SAE model.

3.2 Existing Datasets for APT Detection Research

Table 3.2: Analysis of Phases of APT Attack Covered by Attack Vectors of Existing Works Involving APT, Network Intrusion, and Anomaly Detection in Cybersecurity. The Table Compares Attack Phases Covered by Datasets UNB-15 Moustafa and Slay (2015), CICIDS 2017 Sharafaldin *et al.* (2018), NSL-KDD Dhanabal and Shantharajah (2015), Mawi Fontugne *et al.* (2010), ISCX Shiravi *et al.* (2012), DARPA Cunningham *et al.* (1999), HERITRIX Wang *et al.* (2016b), and DAPT 2020.

Dataset / **APT Phase**	**UNB-15**	**CICIDS**	**NSL-KDD**	**Mawi**	**ISCX**	**DARPA**	**HERITRIX**	**DAPT 2020**
Normal Traffic	✓	✓	✓	✓	✓	✓	✓	✓
Reconnaissance	✓	✓	✓		✓	✓		✓
Foothold Establishment		✓	✓	✓	✓	✓	✓	✓
Lateral Movement								✓
Data Exfiltration								✓

An analysis of existing datasets Table 3.2 used in APT detection research shows that most datasets cover only reconnaissance and foothold establishment phases based on analysis of network traffic features attack vectors employed in these datasets. As part of APT research work, the dataset I proposed considers lateral movement and data-exfiltration, critical phases of an APT attack. Thus, DAPT 2020 is comprehensive enough to cover all stages of an APT attack.

Table 3.3: Comparison Between Attack Vectors of Each Dataset in Terms of Different Attack Vectors That is Involved in APT Attack. The Table Compares Existence of Every Attack Vectors by Datasets UNB-15 Moustafa and Slay (2015), CICIDS 2018 CSE-CIC-IDS2018 (2018) CICIDS 2017 Sharafaldin *et al.* (2018), NSL-KDD Dhanabal and Shantharajah (2015), MAWI Fontugne *et al.* (2010), ISCX Shiravi *et al.* (2012), DARPA Cunningham *et al.* (1999), HERITRIX Wang *et al.* (2016b), and DAPT 2020.

Dataset / **Attack**	**UNB-15**	**CICIDS 2018**	**CICIDS 2017**	**NSL-KDD**	**MAWI**	**ISCX**	**DARPA**	**HERITRIX**	**DAPT 2020**
Network Scan	✓	✓	✓	✓	✓	✓	✓		✓
Web Vulnerability Scan	✓	✓							✓
Account bruteforce		✓	✓	✓	✓	✓	✓		✓
SQL injection		✓	✓	✓		✓			✓
Malware Download	✓		✓						✓
Backdoor	✓	✓				✓			✓
Command Injection	✓	✓	✓				✓	✓	✓
DoS	✓	✓	✓	✓	✓	✓	✓		✓
CSRF		✓	✓			✓			✓
Privilege escalation			✓		✓	✓		✓	✓

Next, I consider the attack vectors employed in the construction of datasets used in APT research. It can be observed from Table 3.3 that existing datasets consider only a few attack vectors, whereas, in the case of APT attacks, an attacker is likely to employ a wide range of attack tools and techniques.

3.3 DAPT 2020 Design and Implementation

The DAPT 2020 uses VMWare ESXi physical servers used for hosting the enterprise cloud network. To mimic the attack pattern typical of an APT attack, I created separate public and private networks, hosted using Docker Merkel (2014) based virtual environment. I deployed Mutillidae OWASP (2020), DVWA (2020), Badstore Vulnhub (2020), and Metasploitable Security (2020) on the VM public network. The private network comprises some core business services such as DNS, Nexus

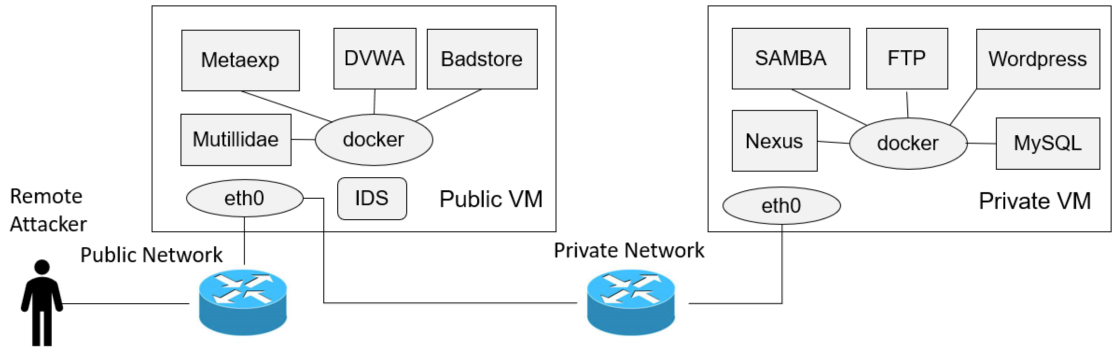


Figure 3.2: DAPT 2020 System Overview

storage service, and WordPress website. The public network can be configured with Snort based IDS Roesch *et al.* (1999). Each virtual machine was also configured to store network traffic capture and service logs (access logs, authentication logs, DNS logs, etc.). The logs were collected at a centralized log server.

To mimic the APT attacks, a team of normal users was instructed to perform business tasks such as creating an account on a WordPress website, accessing Nexus storage, browsing the web APIs on the public and private network. The detailed activity of tasks that were performed in order to mimic APT attacks has been provided below:

- **Monday:** Normal traffic baseline was performed. The users browsed the public services, used tools such as ping, dig, GET/POST request methods, performed standard upload, download operations.

- **Tuesday:** An internal Red Team utilized activities such as fingerprinting, network service, and vulnerability scanning. The tools such as burpsuite, webscarab, dirbuster, and account discovery tools were employed. The goal was to identify vulnerabilities present on the public network, authentication and authorization weakness on the system, etc.

- **Wednesday:** Initial foothold establishment was performed by the Red Team on Wednesday. Team employed PHP reverse shell, sqlmap scripts, metasploitable payloads, authentication, and authorization bypass methods to get access to network services exposed on the public network. The team added additional accounts to maintain persistence.

- **Thursday:** The goal was to perform lateral movement and target some core services of the private network. The team scanned the internal system, identified vulnerabilities present on the core network services such as *vsftpd 2.3.4*, weak SSH authentication, CVE-2012-2122. The team obtained user and root-level privileges on the containers with core services by using lateral movement.

- **Friday:** The data from core services such as MySQL, FTP was exfiltrated to a Command and Control (C&C) center present on the remote location. The team created Google Drives and Remote FTP server to act as C&C. Commands and tools such as scp, wput, PyExfil, etc. were utilized to exfiltrate the data to a remote location.

An analysis of known APT attacks such as APT41 Dragon (2020), Target APT attack Wagner *et al.* (2017), and RSA Secured ID TheRegister (2020) provided in DAPT 2020 Myneni *et al.* (2020) shows that attack vectors and APT phases present in these well-known attacks are covered in the dataset that I have described in this section.

3.4 Experimental Analysis for Benchmarking APT Models and Datasets

I compared the DAPT 2020 with well-known security datasets CICIDS 2017 Sharafaldin *et al.* (2018), CICIDS 2018 CSE-CIC-IDS2018 (2018), UNB 2015 Moustafa and Slay (2015), used by machine learning models for detection of the APT attack. I compared the performance of anomaly detection models based on semi-supervised machine learning Stacked Autoencoder (SAE), Stacked Autoencoder with Long Short-Term Memory (LSTM-SAE), and 1-class Support Vector Machine (1-SVM). I utilized the Precision-Recall (PR) curve as a measurement metric. Ideally, the anomaly detection algorithm should be sound and complete with respect to anomalies

- **Precision** identifies only anomalies. It measures the percentage or rate. If the algorithm in empirical evaluation identifies set *S(t)* of anomalies for threshold t, precision determines the percentage of real anomalies. The precision is represented using the formula below. The symbol #(.) denotes the cardinality..

$$y = \text{Precison} = \frac{TP}{TP + FP} = \frac{\#(\text{Anomalous Traffic})}{\#(\text{Dataset})}$$

- **Recall** measures how well the algorithm identifies all the anomalies. The algorithm in experimental analysis identifies set *S(t)* of anomalies. This set is a percentage of the full set of anomalies..

The anomaly datasets exhibit high-class imbalance. The representation of attack data is $\sim 2\%$ in the test-set. In such a scenario, a naive classifier will classify all data to the majority class. Next, due to the variance in the algorithms and datasets used in the experimental analysis, it isn't easy to reason about 2% scale. The measurement metrics such a Receiver Operating Characteristics (ROC) Davis and Goadrich (2006) utilizes False Positive Rate (FPR) in comparison to True Positive Rate (TPR) used by PR-curve. The FPR is less sensitive to changes in the number of false positives (regular traffic misclassified as attacks). In contrast, the precision (positive predicted value) looks at only samples of the positive class. Since the positive class in anomaly datasets is underrepresented, the experimental analysis uses PR-curve as a better measurement metric. An analysis of dataset CICIDS 2018 shows brute-force attacks represent $\sim$ 22% of total attack traffic, whereas, in UNB 2015, brute force attack represents $\sim$ 14% of the total traffic. If a naive classifier is utilized in such cases, it will classify everything to the majority class (86% in the case of CICIDS 2018 and 75% for UNB 2015).

I utilized a confidence threshold p, which is exhibited by the model after normalizing the error across test samples between $[0, 1]$. The value p is the indicator of confidence with which the model predicts input as an anomaly. I utilize a threshold τ to plot a point on PR-curve. Suppose the test input value is $p < \tau$ in this model. I classify it as normal traffic (anomaly otherwise). A confusion matrix is obtained by doing this for all test inputs. The Precision and Recall for a particular τ are thus obtained and plotted on PR-curve. The model considers the ideal classifier and No-skill (NS) classifiers to showcase how semi-supervised algorithms perform for different phases of the APT attack. An ideal classifier will predict the input test label with complete confidence, i.e., $p = 1$ for anomalies, and *p=0* for normal traffic. The result will be $y = 1$ on PR-curve for such a classifier. On the other hand, an NS classifier will output $p = 1$ on input data and can be plotted as a line. For the empirical evaluation, I considered each phase of the APT attack in isolation and analyzed how well each algorithm performs on APT attack phases. Note that some datasets in the experiment do not have APT attack phases, lateral movement, and data exfiltration, and hence there is no data reported for these cases. The attack vectors utilized in each APT phase have been described in Table 3.3. The training set for each attack phase comprised regular traffic, whereas the test set comprised both normal and attack traffic. For instance, in data exfiltration, the training set consisted of regular data traffic from Monday-Friday. In contrast, the test set consisted of standard and attack traffic on Friday (the red team performed data exfiltration).

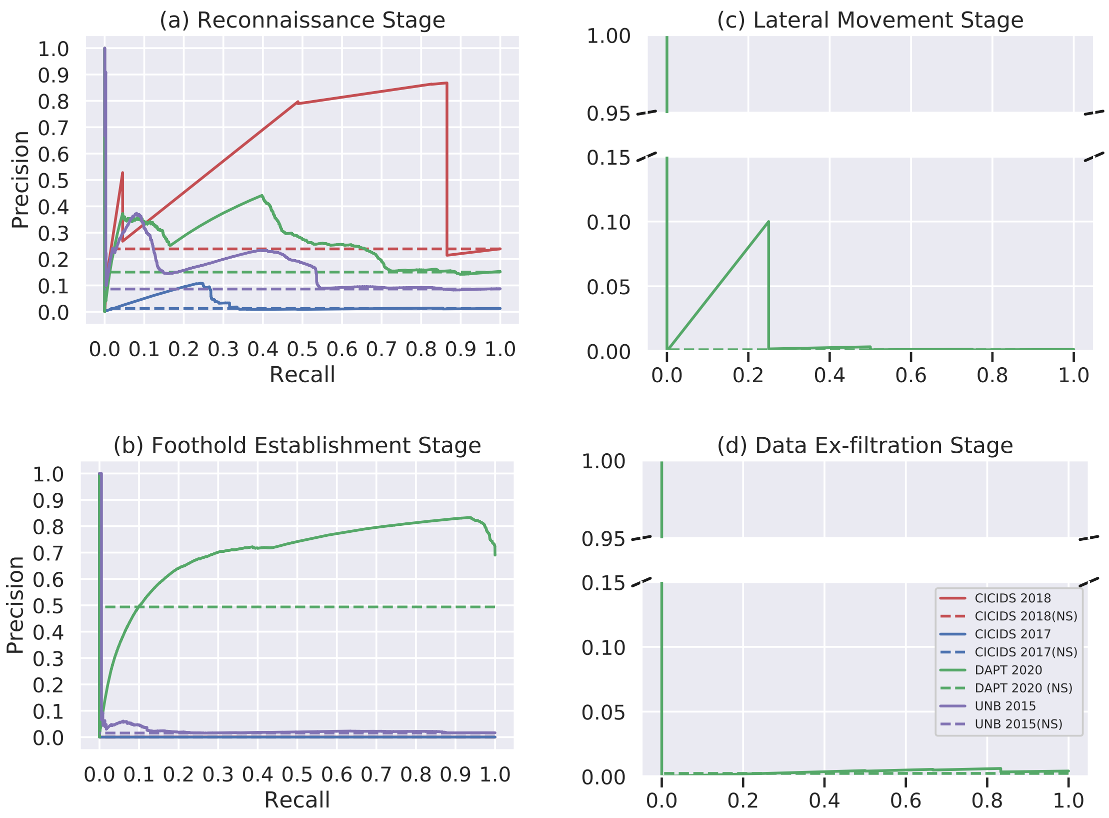


Figure 3.3: Precision-Recall (PR) Curves for Detecting Attacks Across the Various Stages of an APT for the Different Datasets using the Stacked Auto-encoder (SAE).

3.4.1 Performance Analysis for SAE

The results of anomaly detection performance for SAE showcase good performance on a) *reconnaisance* attacks for CICIDS 2018, and foothold establishment for DAPT 2020, Figure 3.3. During the reconnaissance phase, the NS classifier has a precision value of 0.23 for CICIDS 2018. The fraction of anomalies detected is highest with a precision value of 0.87. There is a high value of reconnaissance traffic in CICIDS 2018, which is reflected in the PR-curve. The performance of SAE on the DAPT 2020 dataset and UNB 2015 does not differ significantly. This can be explained by the fact that the NS classifiers' baseline in DAPT 2020 is higher than UNB 2015. The detection of reconnaissance using SAE for CICIDS 2017 is the worst of all existing datasets, with the highest value of ~ 0.1. (b) the datasets analyzed during *foothold establishment* phase consisted of SQL Injection and brute-force as major attack vectors. In DAPT 2020 dataset, nearly 50% of traffic comprised of foothold establishment attacks. The SAE outperforms other datasets for the foothold establishment on DAPT 2020. The low traffic on other datasets is hard to distinguish compared to attack traffic, and hence detection rate for SAE is quite low (as bad as No skills classifier). (c) The y-axis in case of *lateral movement and data exfiltration* show low detection rate [0, 0.15]. The SAE shows a poor detection rate in our dataset < 0.1 for these two phases.

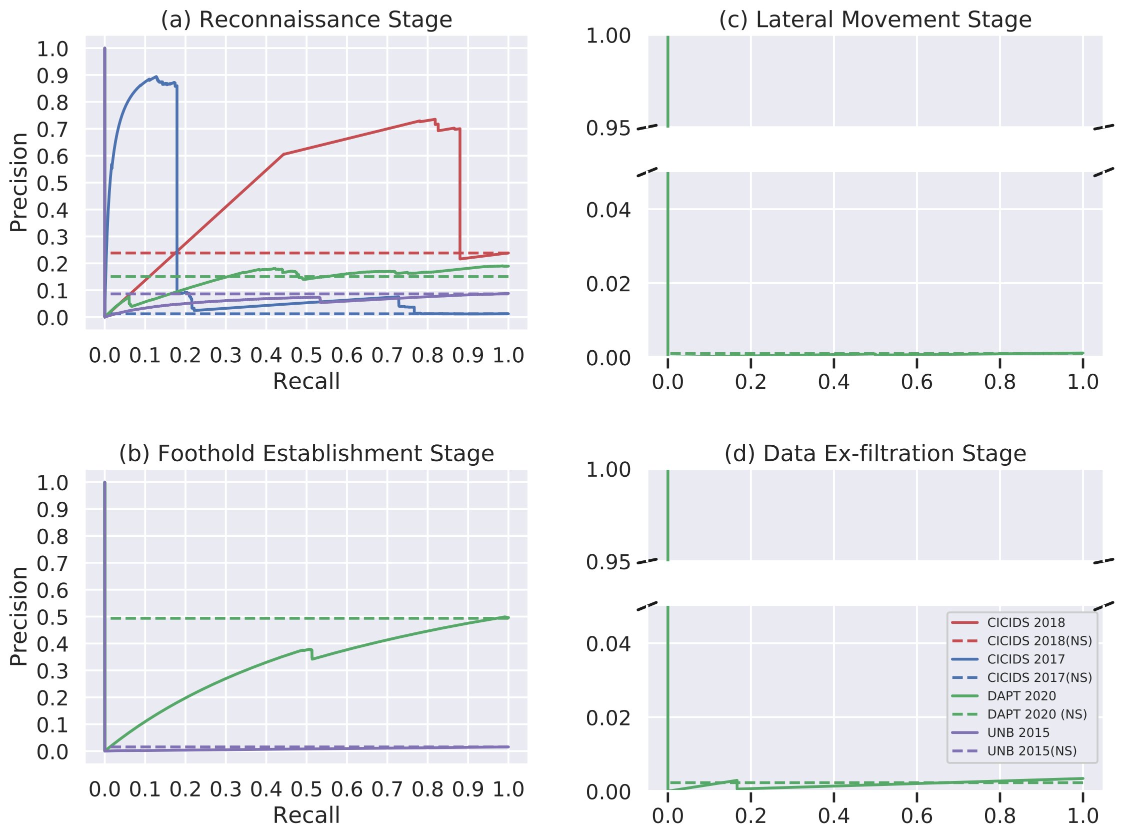


Figure 3.4: Precision-Recall (PR) Curves for Detecting Attacks Across the Various Stages of an APT for the Different Datasets using encoder LSTM-SAE.

3.4.2 Performance Analysis for LSTM-SAE

The LSTM-SAE shows an improvement in performance when detecting a) *reconnaisance* attacks on CICIDS 2017, Figure 3.4, i.e., the performance value improves to ~ 0.9, compared to 0.1 for SAE model. The performance improvement indicates that the contextual information in the case of the reconnaissance phase is identified using the LSTM-SAE model. However, the analysis of the reconnaissance phase on other datasets indicates that the addition of contextual information makes a distinct representation of attack vectors difficult. The PR value obtained is close to normal representation, and in-effect the effectiveness of anomaly detection on these datasets for the reconnaissance phase is limited. (b) the LSTM-SAE performs poorly when *foothold establishment* phase is analyzed. Even though there is a significant amount of attack data present, the PR-value is close to the no-skills classifier for LSTM-SAE in this APT phase. This observation is indicative of the fact that as attack timesteps increase, the effect of contextual information is not as prominent. (c) during *lateral movement and data exfiltration* phases, the LSTM-SAE performs worse than SAE. This shows that the distribution of contextual information in these phases of attacks is almost the same as the normal traffic.

3.4.3 Performance Analysis for 1-SVM

The classifier performs poorly (a) during the *reconnaisance* phase for all datasets except CICIDS 2018. For other datasets, the performance is comparable to the No-skills classifier. Thus, the classifier can only identify reconnaissance events in the case of CICIDS 2018. This is due to the presence of a large portion of clearly evident reconnaissance traffic in the case of CICIDS 2018. The percentage of attack traffic for all other datasets, including DAPT 2020, is relatively low. Hence, the classifier cannot

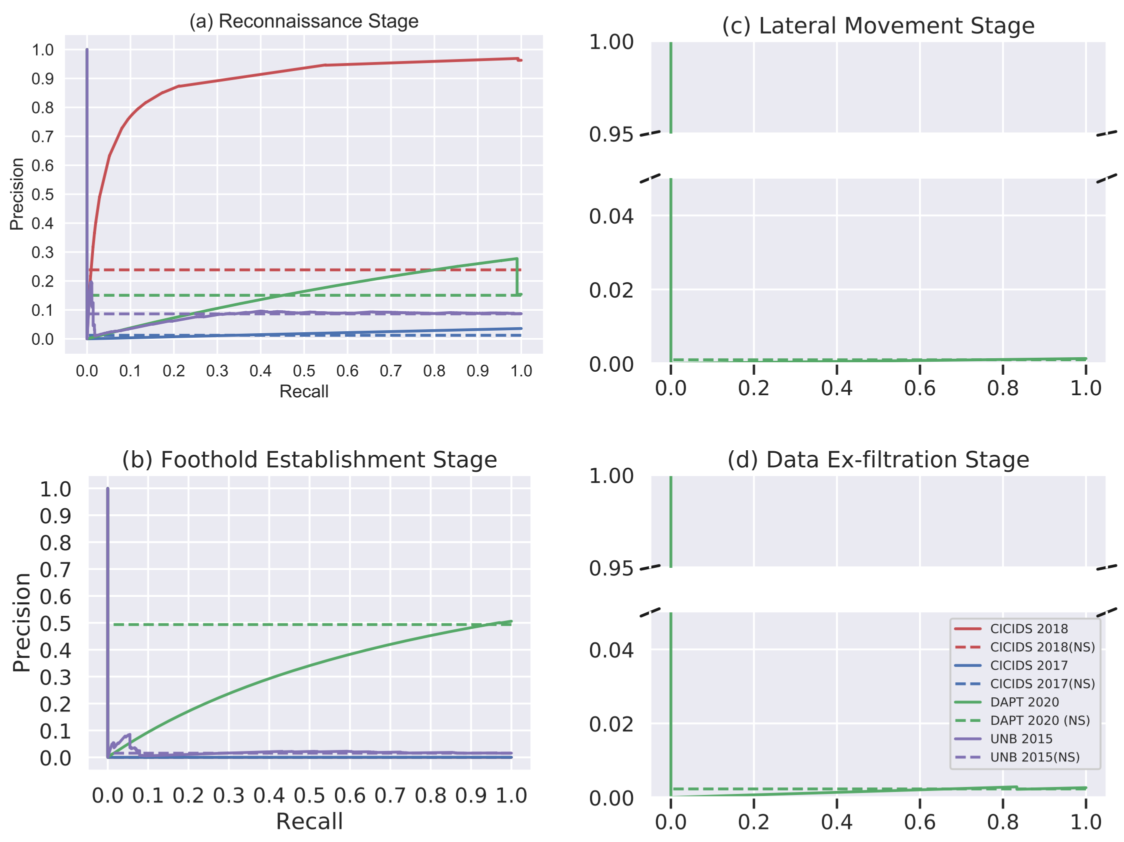


Figure 3.5: Precision-Recall (PR) Curves for Detecting Attacks Across the Various Stages of an APT for the Different Datasets using 1-class Support Vector Machine (1-SVM).

identify reconnaissance events with high precision (b) the *foothold establishment 1-SVM determines* phase for DAPT 2020. In contrast, performance on all other datasets for this phase of the APT attack is relatively low (comparable to a No-skills classifier) (c) the PR-value for *lateral movement and data exfiltration* is extremely low for 1-SVM. This is similar to the PR-value pattern observed for SAE and LSTM-SAE. Thus, existing anomaly detection models exhibit a low detection rate for the APT attack's final two phases.

3.5 Summary and Discussion

The data related to APT attacks are difficult to obtain because of privacy concerns inherent in the data. Thus the creation of a dataset that closely mimics an APT attack - DAPT 2020 is extremely useful. The data augmentation techniques Antoniou *et al.* (2017) prevalent in existing machine learning research pose challenges. (1) It is quite difficult to distinguish regular traffic and attack traffic in existing datasets since APT attacks try to use tools and techniques to evade detection and blend in normal traffic. (2) The regular intrusion detection datasets such as CICIDS 2017, 2018 that have been used for detection of APT attacks do not have dependencies between consecutive attack phases like reconnaissance, foothold establishment. This limits the usability of these datasets in APT attack detection research. Thus, the effectiveness of data augmentation techniques in APT attacks requires further investigation. The analysis of existing machine learning models showcases both the contextual and collective effects of APT attacks. These anomalies are difficult to identify over multiple phases of an APT attack correctly. The use of alternative models such as Generative Adversarial Networks (GAN) Goodfellow (2016) and other models that can correlate attack activities from different phases of APT attack can be present some exciting results for detection of APT attacks. The lack of reliable detection of APT attacks serves as a motivation for exploring proactive defense mechanisms such as Moving Target Defense (MTD) when dealing with APT attacks. I describe the MTD in the next part of this thesis and discuss the use of programmable networks (SDN/NFV) technologies Huang *et al.* (2018) for implementing the MTD defense mechanism.

Chapter 4

SDN-BASED MOVING TARGET DEFENSE FOR SINGLE-STAGE ATTACKS

The static nature of networks is useful from the service provisioning aspect. In a cloud network, the service providers want system configuration to remain unchanged once an application is deployed. This static configuration makes the cloud system a soft target for the attackers since they can spend the time to perform reconnaissance on the system and craft the necessary attacks based on the information gathered by cloud system exploration. In this chapter, I discuss Moving Target Defense (MTD) based proactive security defense to counter single-stage attacks. MTD helps in continuously shifting the underlying system's configuration, thus taking away the asymmetric advantage that attackers have in a static network. In the next chapter, I discuss the multi-stage and APT attacks and propose MTD solutions that can disrupt attack propagation at various stages of a multi-host, multi-stage attack.

4.1 MTD Categorization

The categorization of MTD, as presented in Figure 4.1 is useful for a system administrator to implement situation-aware MTD defenses. First, I will elaborate on each category of MTD.

4.1.1 What to Switch - Movement of Configuration Set

The software system, when considered from the perspective of an attacker, can be classified into (a) Exploration Surface (b) Attack Surface (c) Detection Surface (d) Prevention Surface.

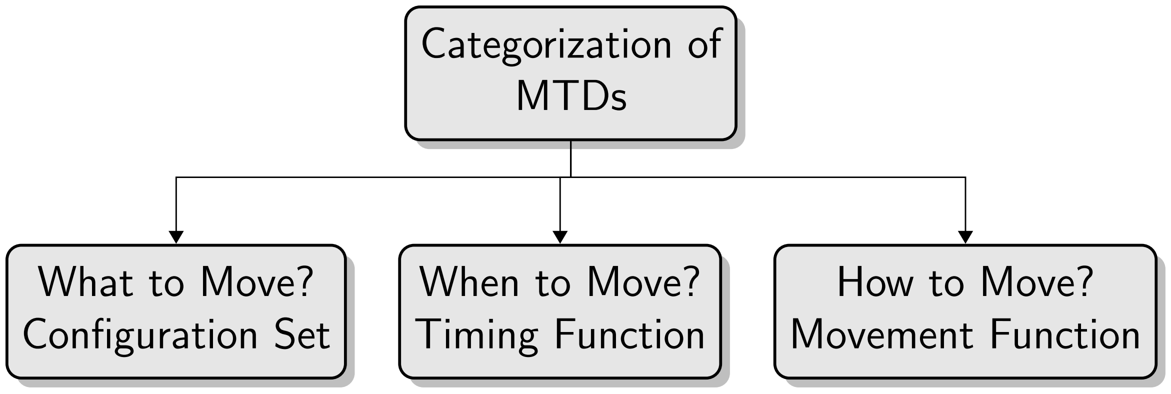

Categorization of MTDs
What to Move? Configuration Set
When to Move? Timing Function
How to Move? Movement Function

Figure 4.1: Moving Target Defense (MTD) Categorization

Exploration Surface Shifting

An attacker may try to explore the network using network scanning techniques. The attack performs reconnaissance to understand network topology, host operating systems, network bandwidth. The knowledge obtained from the exploration surface can help an attacker perform targeted attacks and establish a foothold in the network. The goal of using MTD during this phase is to ensure the attacker's reconnaissance activities are rendered inefficient. The adversary is presented a false view of the network information. Thus, subsequent steps of the attack operate on this faulty view of the information. Some examples include Al-Shaer *et al.* (2012); Albanese *et al.* (2013) using Random Host Mutation (RHM) for moving the target hosts to unassigned random virtual IP addresses (vIP), Jafarian *et al.* (2012) utilized SDN based framework to implement MTD on similar configuration set. Some exploration techniques utilize the timing information to fingerprint the target software. Algin *et al.* (2017) modify the schedule, guiding host transmission information, and reduce the effectiveness of timing attacks. Similarly, Schlenker *et al.* (2018) responds to the attacker's query with incorrect information, and in-turn deceives the adversary.

Attack Surface Shifting

Once the attacker has sufficient knowledge of the target network, the attacker selects exploits (e.g., metasploitable payload) Singh *et al.* (2020) to perform a targeted attack. For instance, if the webserver identified during the exploration phase is an apache web server, the attacker checks if there is a known vulnerability for the particular version of the software and selects the available attack scripts to exploit the vulnerability. Carter *et al.* (2014) utilizes an Operating System (OS) switching strategy to shift to an alternate version of OS using a centralized control center. Chowdhary *et al.* (2016), describe an MTD that leverages port hopping to thwart known attacks on a cloud-based system. El Mir *et al.* (2016) utilize Virtual Machine Migration (VMM) as a means of dealing with known vulnerabilities. The capacity constraints of physical server hosting VM are considered while shifting the attack surface. A multi-layer attack surface shifting has been discussed in research works multi-layer MTD, as discussed in Sengupta *et al.* (2017); Zhu and Başar (2013).

Detection Surface Shifting

The detection surface refers to network monitoring tools - centralized log server Carasso (2012), intrusion detection system (IDS) Roesch *et al.* (1999), which are placed strategically in the network to detect an attacker. The placement of detection systems in a static manner can lead to attackers discovering the detection mechanism and using means to evade detection. On the other hand, over-provisioning the detection system can lead to degradation of network performance, as noted by Sengupta *et al.* (2018). The movement of detection surface in a strategic manner can ensure that attackers can be detected with a high probability while ensuring that the network performance is not severely impacted Venkatesan *et al.* (2016); Sengupta *et al.* (2018).

Prevention Surface Shifting

The prevention surface refers to the countermeasures present to mitigate the security threat. This includes network-based Firewall Russell and Welte (2002), web application firewall, intrusion prevention system. Movement of prevention surface also adds a layer of reasoning on the adversaries part; for example, distinguishing between whether their attack was undetected or the attacker's behavior is being currently monitored. There has been limited research in the field of MTD-based prevention surface shifting. In some cases, authors make this assumption in Chowdhary *et al.* (2017b), an MTD mechanism that modifies the bandwidth of the network in response to malicious activity is proposed.

4.1.2 When to Switch? - Timing Function

An essential question in MTD is when and how-often the change in network or software configuration is required. The defender needs to devise a strategy that changes the network at a constant or variable rate. The MTD based on timing function can be classified into (a) Constant Period Switching (b) Variable Period Switching.

Constant Period Switching (CPS)

Research works Chowdhary *et al.* (2018a); Sengupta *et al.* (2017, 2018); Manadhata (2013); Jajodia *et al.* (2018) do not discuss the time-period for MTD. These works, however, have an implicit assumption that the period for movement is fixed. The research works Zhu and Başar (2013); Carter *et al.* (2014); Algin *et al.* (2017) consider that time for equilibrium MTD strategy is at the beginning of each stage. Algin *et al.* (2017); Albanese *et al.* (2013) consider the effectiveness of MTD for different values of timing functions, i.e., impact on MTD effectiveness when different values of a constant period are considered. Debroy *et al.* (2016) study the historical attack data and obtain a value cyber-attack inter-arrival (CAIA) rate as a constant period for MTD. *traceroute* Padmanabhan and Simon (2003) considers data between possible source-destination pairs to decide on a reasonable time-period for obfuscating links or mutating routes of ICMP packets on the network.

Variable Period Switching (VPS)

The value of the time period can be obtained using system and network attack parameters. The variable period switching is a two-layer MTD strategy. In the first layer, the MTD mechanism for shifting the timing surface is deployed, whereas, in the second layer, the actual cyber-surface shifting is implemented. The variable period switching strategy can be further sub-categorized into:

- *On-event switching* - the shifting strategy depends on an event such as attack detection, link unavailability. Based on the event, the timing function determines the time-period to switch. Van Dijk *et al.* (2013) considers the boolean variable [0/1] as a parameter to indicate who has control over the system - attacker or defender. The belief that the real value of the variable is 0, which means the attacker has control over the course, is used by the defender to decide MTD's time period. Chowdhary *et al.* (2017b) utilizes the network bandwidth value as a parameter for rate-limiting the bandwidth for the attacker. Shi *et al.* (2017) scan for unexpected connections on the network and decide on an MTD based on the presence of unforeseen links on the system.

- *Strategic Switching* - Lei *et al.* (2017) utilize game-theoretic strategy based on current configuration (c) and discrete value of time (t) to represent game state. Each state is represented as a tuple (c,t). The optimal strategy for defense in each tuple value can be identified using the game's solution formulated in this work. However, the state-space representation of this nature can lead to an explosion in the number of states, thus limiting the model's scalability. El Mir *et al.* (2016) utilizes state-space modeling, which involves timing function, but authors resort to a simple strategy for defining the timing function. The timing function considers the impact of (known) vulnerabilities in the currently deployed configuration and, based on it, recommends a switching time.

4.1.3 How to switch? - Movement Function

The movement policy can be obtained based on intelligent game-theoretic modeling. The modeling can be classified into games that consider (a) Single-stage (b) Multi-stage interaction between the attacker and defender.

Single-stage Modeling

The goal is to identify a mixed strategy that maximizes the defender's reward. Manadhata (2013) tuple of actions available to attacker and defender, and rewards, $R^{\mathcal{A}}$ and $R^{\mathcal{D}}$ for the attacker and the defender to calculate Subgame Perfect Equilibrium (SPE), which maximizes the reward for the defender. Chowdhary *et al.* (2018a) utilize game modeling based on a real-world example and compare the effectiveness of a reactive switching strategy to the Uniform Random Strategy (URS). A single-stage normal form game is considered by Carter *et al.* (2014); Sengupta *et al.* (2018); Vadlamudi *et al.* (2016); Venkatesan *et al.* (2016); Zhu and Başar (2013). Vadlamudi *et al.* (2016); Sengupta *et al.* (2018) identify strategy based on Strong Stackelberg Equilibrium (SSE) for identification of optimal MTD strategy. Stackelberg Equilibrium (SE) as a means of thwarting DDoS attacks has been proposed by Venkatesan *et al.* (2016).

Multi-stage Modeling

The research works consider the history of actions taken by attackers and defender future states' reward values based on present state and actions to create a multi-stage game-theoretic model. Chowdhary *et al.* (2017b) discretizes the continuous action space of the defender and utilizes the bandwidth values over time to ensure that DDoS attacks mounted by the attacker are rendered ineffective. Colbaugh and Glass (2012) map attacker-defender interactions as a repeated game, and identify defender's policy against self-learning attacker. Maleki *et al.* (2016) considers policies over the defender's action set comprising of single or multiple IP hops. Zhao *et al.* (2017) observe attacker-defender interactions over multiple gameplays and update their belief states in each stage to identify the optimal strategy for the defender.

4.2 SDN for Implementing MTD

SDN has emerged as a state-of-the-art network architecture for data centers and backbone networks. Google's *B4 project* Jain *et al.* (2013) shows the feasibility of SDN for handling real-world network challenges such as traffic engineering and Quality of Service (QoS). SDN allows centralized command and control of the network. The flexible programmable network solution offered by SDN makes it an ideal candidate for provisioning MTD solutions. For instance, in a cloud network managed by SDN, the SDN controller can be notified by security analysis tools about an active threat in the system. The controller can take preventive methods to deal with the situation, such as IP address hopping, reconfiguration of routes for network traffic, or changing the host endpoint to delay the attack propagation.

Network Mapping and Reconnaissance Protection The first step of the Cyber Kill Chain is identifying vulnerable software and OS versions. Most scanning tools use ICMP, TCP, or UDP scans to determine the potential targets' connectivity and reachability. The scans' replies can also reveal the firewall configuration details, i.e., what traffic is allowed or denied. The Time to Live (TTL) information can also help identify several hops to the attack target Kampanakis *et al.* (2014). An SDN-based solution can modify the TTL field associated with the reply for the attacker's scan query.

Service Version and OS Hiding The attacker needs to identify the OS or vulnerable service version to mount an attack. For instance, the attacker can send *HTTP GET* request to Apache Web Server, and the response can help identify vulnerability associated with a particular version of the Apache software. If the attacker gets a reply *404 Not Found*, he can locate some obfuscation happening at the target software. A careful attacker can thus change the attack vector to exploit the vulnerability at the target. An SDN-enabled solution can override the basic service version with a bogus version of the Apache Server. Some application proxies leverage this technique to prevent service discovery attempts by a scanning tool. Another attack method is known as OS Fingerprinting , where the attacker tries to discover the version of the operating system which is vulnerable. Although modern OS can generate a random response to TCP and UDP requests, how TCP sequence numbers are generated can help an attacker in the identification of the OS version.

Distributed Denial of Service (DDoS) is a significant security problem affecting networks. Some recent cases include a massive DDoS attack on DNS provider Dyn in October 2016, Mirai Botnet 2018, and an attack on the well-known security blog `krebsonsecurity.com` which was of magnitude 650 Gbps. The attackers utilize sophisticated botnets to send a large volume of traffic to the victim, thus overwhelming their capacity. The victim cannot reply to any new requests, therefore, causing service disruption for legitimate users. The defense mechanisms such as Firewall, Intrusion

Prevention System (IPS) alone are insufficient to deal with such attacks. Next, I will present the use of an SDN-based MTD strategy to deal with targeted attacks such as Distributed Denial of Service (DDoS). There are multiple entities (such as routers, firewalls, switches, etc.) to enforce the security mechanism. The cooperative sharing of attack information between devices with data and control planes embedded in a single machine is hard; it becomes challenging to detect and counter such attacks.

Software-defined networking (SDN) provides a clean separation between data and control plane Kreutz *et al.* (2015). A controller such as OpenDaylight (ODL) is logically centralized to take critical decisions such as routing, load-balancing, and implementing Firewall policies Feamster *et al.* (2014). The data-plane takes care of traffic forwarding, whereas devices enforce the control plane's control decisions. I model the attacker-defender interaction in case of a DDoS attack as a two-player dynamic game. The attacker targets critical infrastructure by sending a huge volume of traffic through bots distributed inside or outside the target environment. To this end, I introduce a game-theoretic model that will help uncover the entire botnet and rate-limit traffic from these malicious users/bots. The concept of reward and punishment, which is used in game-theoretic models to enforce cooperation between firms, has been employed in this research work. To sustain mutually desirable outcomes, the agents/users with undesired behavior receive a penalty.

The system is defined using a dynamic two-player game. An administrator controls network infrastructure and deploys OpenFlow rule-based countermeasures to deal with targeted attacks like DDoS. Some countermeasures deployed as part of defense strategy can conflict with some existing rules. Part III of this thesis is dedicated to handling such security policy conflict scenarios. The network admin observes the normal baseline behavior for all hosts managed by the SDN controller initially. The attacker has an incentive for deviating from normal behavior.

		Player 2	
		a_2^1	a_2^2
Player 1	a_1^1	$(\frac{B}{2}, \frac{B}{2})$	$(\frac{3B}{4}, \frac{B}{4})$
	a_1^2	$(\frac{B}{4}, \frac{3B}{4})$	$(\frac{B}{5}, \frac{4B}{5})$

Table 4.1: Normal Form Representation of Attacker and Administrator Payoff's

4.2.1 Game Theoretic Modeling

Definition 4.2.1 *An N player extensive form repeated game G with perfect information between two players can be represented as* $G = \{N, A_i, u_i\}$ *where* $N = \{1, 2, \dots n\}$ *denotes number of players,* $a_i \in A_i$ *is the action set available to player i.* $u_i : a_i \mapsto R_i$ *is the payoff function that maps actions to reward value R.*

Definition 4.2.2 *Consider that the game has been played to t periods of time, and define game history in an instance t as* $h_t = \{a_1, a_2, .., a_{t-1}\} = A^{t-1}$. *This denotes actions taken by a player until now.* $H_1 = \{\emptyset\}$, *and* $H = \sum_{t=1}^{\infty} H_t$.

A player i prefers an action a^t over b^t if $u_i(a^t) \geq_i^* u_i(b^t)$. The payoff profile for a player is considered feasible in this game, i.e. convex combination of payoff profiles. A, $s = \sum_{a \in A} \alpha_a u(a)$ such that $\sum_{a \in A} \alpha_a = 1$. Here s represents the strategy vector for a player. Consider an example of two players, who take turns to decide on an action. Assume the players P_1 and P_2 correspond to an attacker and administrator.

Definition 4.2.3 *The strategy vector of a player i,* s_i^* *is best response to strategy vector of all other players* s_{-i}^* *if* $u_i(s_i^*, s_{-i}^*) \geq u_i(s_i, s_{-i}^*)$ *for all* s_i. *This vector is a Nash Equilibrium if the relation holds for all* s_i *and all i.*

In the game-setting of two players, I consider both players to try to play the Nash Equilibrium strategy against each other. The game-theoretic formulation considers utility in terms of network bandwidth for this game. P_1 has actions $a_1 = \{$*Cooperate,*

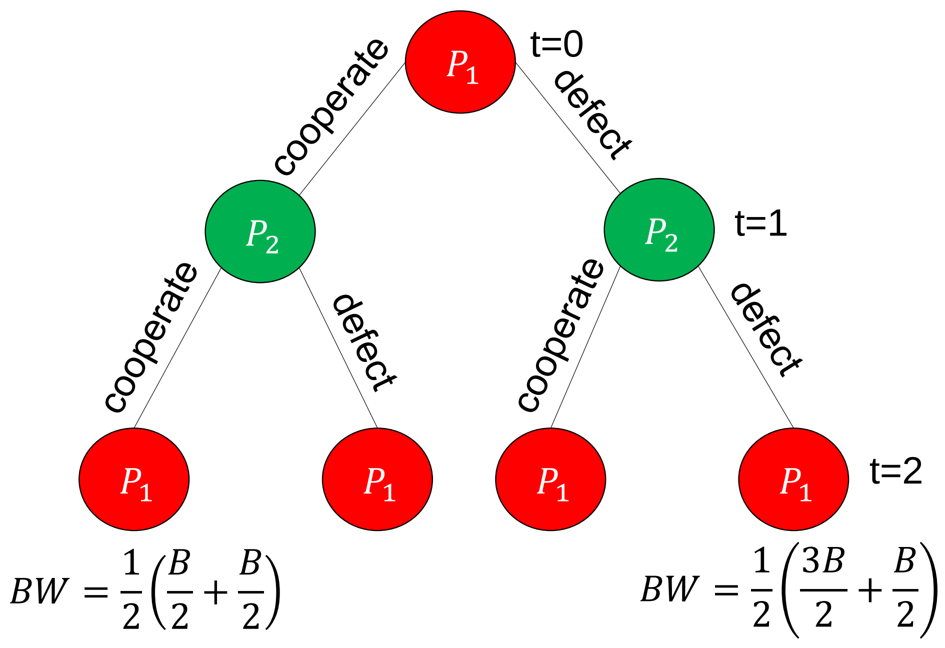

P_1
t=0
cooperate
defect
P_2
P_2
t=1
cooperate
defect
cooperate
defect
P_1
P_1
P_1
P_1
t=2
$BW = \frac{1}{2}\left(\frac{B}{2} + \frac{B}{2}\right)$
$BW = \frac{1}{2}\left(\frac{3B}{2} + \frac{B}{2}\right)$

Figure 4.2: An Extensive Form Dynamic Game Between Attacker and the Admin.

Defect} and P_2 has action set $a_2 = \{$*Cooperate, Defect*$\}$. As long as any player behaves benignly, the administrator will allow normal bandwidth. If the administrator detects an attack from some malicious node in a network, he/she will play a strategy of Rate-Limiting the attacker's bandwidth available. The sample payoff matrix in normal form has been shown in Figure 4.2, where B denotes the total network bandwidth.

4.2.2 Implementing Rate Limiting Algorithm using Dynamic Game

The system setup used for implementing the game theoretic model comprised on OpenDaylight (ODL) Medved *et al.* (2014) SDN controller. The SDN controller uses northbound APIs to interact with the application plane and southbound APIs for interfacing with data-plane elements. The SDN controller acts as network manager and orchestrator, as can be seen in Figure 4.3. A Snort-based NIDS Roesch *et al.* (1999) configured to tap network traffic in mirroring mode.

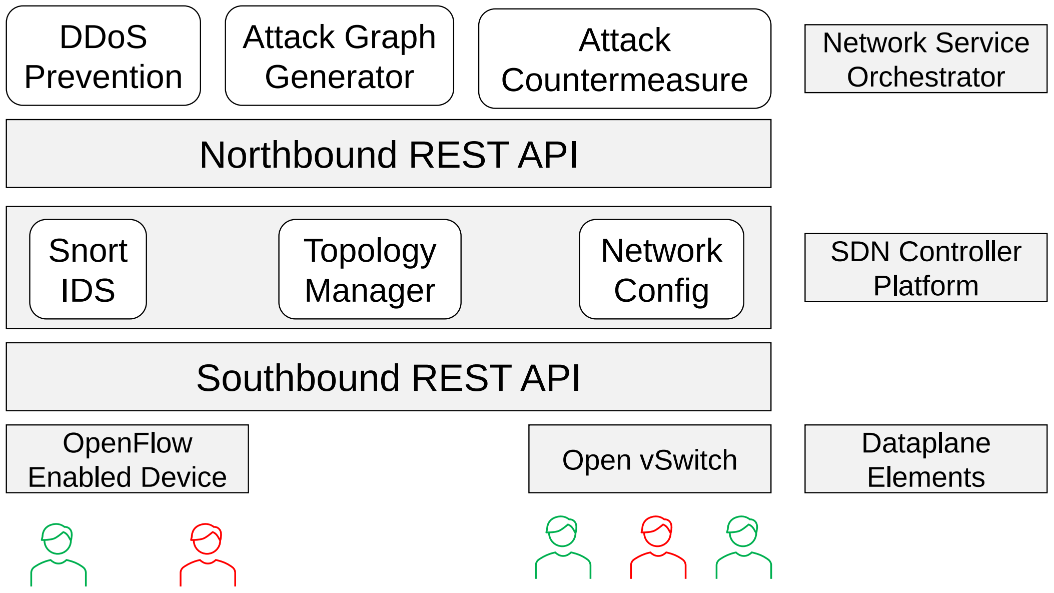


Figure 4.3: SDN System Architecture Utilized for Implementing Rate-Limiting Algorithm in an OpenFlow-enabled Network. Red Color indicates Attacker Device, and Green Color refers to Normal User Device.

```
alert tcp $HOME_NET any -> $HOME_NET 80
(flags: S; msg:"Possible TCP DoS";
flow: stateless; threshold: type both,
track by_src, count 70, seconds 10;
sid:10001;rev:1;)
```

Figure 4.4: An Example Snort Rule which is used for Detection of DDoS Attack.

The network traffic is categorized as malicious or benign using the Snort IDS signature match. Northbound REST APIs present is used to pass the attack pattern information to application layer functionality, such as DDoS prevention. I consider three variants of DDoS attacks TCP SYN-Flood attack, UDP Flood attack, ICMP Flood attack. The feedback received from Snort IDS is utilized to update the rate limit option present in the OpenFlow table, based on the traffic match between source and destination host address. The different fields of the SDN OpenFlow table are pre-

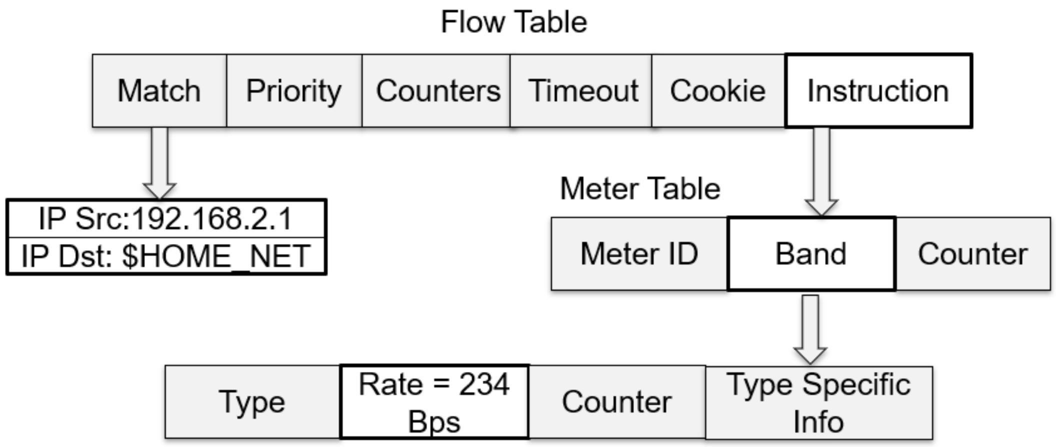


Figure 4.5: SDN-based Traffic Rate-Limiting for OpenFlow-based Devices.

sented in Figure 4.5. The ingress port of the traffic entering the network is identified using *match* field of the OpenFlow table. For instance, if Snort alert consists of IP address *192.168.1.2*, this value is inserted as an OpenFlow rule, The *Instruction* field of the flow entry is added as a field of flow rule. The rate limit of the network traffic with a source IP address matching the IP address present in Snort alert is set to the value decided based on the Nash-Folk theorem based game model. The value is pushed to the OpenFlow device using the SDN controller via Southbound REST API.

OpenFlow Rate Limiting Algorithm

The OpenFlow based rate-limiting algorithm 2 utilizes the procedure *SET-RATE-LIMIT-METER* for invoking the meter with a specified meter ID in the corresponding flow table. The meter table is an optional segment of the flow table. It will not be populated by default. The bandwidth of the host under observation is set to normal value if the host is behaving normally. In a DDoS attack, a malicious host is identified using Snort IDS alert with a meter ID. The malicious host is kept under observation

Algorithm 2 SDN-DDoS-Rate-Limit-Algo

```
1: procedure Set-Rate-Limit-Meter(meterName, bandID, bandRate)
2:     MeterName ← this.meterName
3:     mbh ← MeterBuilder.meterBandHeader()
4:     mbh.setBandID(this.bandID)
5:     mbh.setBandRate(this.bandRate)
6: end procedure
7: procedure Nash-Folk-Rate-LimitB(
       )
8:     $u_i(coop, coop) \leftarrow B_c$ {Host $P_i$ cooperates}
9:     $u_i(coop, def) \leftarrow \{B_{cd} > B_c\}$ {Host $P_i$ defects}
10:    $u_i(def, coop)] \leftarrow \{B_{dc} < B\}$ {Host $P_i$ defected at $t_{k-1}$}
11:    $u_i(def, def) \leftarrow \{B_{dd} < B_{dc}\}$ {Host $P_i$ defected at $t_{k-1}$ and $t_k$ }
12:    for i ∈ [0,n-1] do
13:        $ft \leftarrow FlowTable_i$
14:        if ft.match.src_ip ∈ DDoSTrigger(src_ip) and $\sum_{t=0}^{k} \delta^t u_i(coop, coop) \leq u_i(coop, def) + \sum_{t=1}^{k} \delta^t u_i(def, \{coop, def\})$ then
15:            x ← ft.Instruction()
16:            x.SET-RATE-LIMIT-METER("RLMeter", 1, $u_i(det, coop)$)
17:        else
18:            x.SET-RATE-LIMIT-METER("RLMeter", 1, $u_i(det, det)$)
19:        end if
20:    end for
21: end procedure
```

for several units of a discrete-time interval. Based on the behavior of the host, the network admin decides to cooperate or defect (lines 8-11). The strategy (cooperate, cooperate) means that the dataplane host is behaving normally, and both control and dataplane will have bandwidth value set to $\frac{B}{2}$ as shown in Figure 4.1. In case the host behaved maliciously in a given round, the controller will receive diminished bandwidth $\frac{B}{4}$, and the malicious attacker will consume $\frac{3B}{4}$ (cooperate, defect strategy) of total bandwidth. The attacker will be allocated reduced bandwidth in the subsequent round as a punishment mechanism, i.e., network admin plays *defect as strategy.* If the attacker chooses cooperate, he is still punished for misbehavior in the previous round (defect, cooperate) = $\{\frac{B}{4}, \frac{3B}{4}\}$. The punishment for defecting in the current round (defect, defect) is even worse for the attacker. As the game progresses, the attacker finds that his strategy to keep defecting diminishes returns.

The procedure NASH-FOLK-RATE-LIMIT based on the greedy approach loops through all flow tables in invoking meter with rate-limiting threshold if matching source host is found in the list of malicious hosts from a Snort IDS (lines 12-20). The administrator does not wants to punish a misbehaving node infinitely, hence the procedure of punishment is carried out for k instances of time where value of k is determined by equation $\sum_{t=0}^{k} \delta^t u_i(coop, coop) \leq u_i(coop, def) + \sum_{t=1}^{k} \delta^t u_i(def, \{coop, def\})$ in line 13. This linear equation ensures that the defecting host is no better than normal behavior at the end of k periods of punishment.

4.2.3 Experimental Analysis for Rate Limiting

I used network simulator Mininet (2015) and the ODL controller on Ubuntu 16.04 OS for empirical evaluation. The first experiment uses Algorithm 2 to deal with ICMP flood attacks. The second experiment uses the same algorithm for TCP SYN flood and UDP flood based attacks on a fat-tree topology. The variation in topologies for both experiments is used to check the generality of the proposed solution.

ICMP Flood DDoS Attack on Linear Topology

# Attacking Hosts	ICMP Traffic (Mb/s)	ICMP Traffic After (Mb/s)
50	39.49	1.33
100	79.85	2.70
200	163.69	5.54
300	241.17	8.122
400	321.96	10.83
500	467.16	15.69

Table 4.2: Number of Hosts vs ICMP Traffic at T=30s Post Attack.

A python attack script was used for generating DDoS traffic. The script utilized multiprocessing to spawn shell for each host and send ICMP traffic of large packet sizes to a single host in the network. The traffic was port mirrored to a dummy port. The IDS intercepted the attack signature for ICMP flood DDoS attack and passed information to the ODL controller. ODL application for DDoS mitigation decreases the traffic rate by a factor δ consecutively until the long term average for traffic is within typical traffic burst from a provided host. In this particular experiment, I used the value of damping factor $\delta = 0.8$. This scheme punishes all the attacking hosts by degrading traffic throughput gracefully, instead of blocking the traffic entirely or rate-limiting to a fixed value, affecting the traffic from legitimate users.

The table 4.2 shows that traffic burst at target for 100 hosts is 79.85 Mbps when there is no attack prevention mechanism to deal with DDoS attack. However, once IDS sets the Rate Limit trigger, the traffic decreases to 2.70 Mbps, which reduces the rate by a factor of 30. Similarly, as the number of attacking hosts increases from 100 to 500, the throughput of DDoS attack increases from 79.58 Mbps to 467.16 Mbps, which shows a linear scaling in attack traffic. The Rate Limit (RL) algorithm quickly adapted to increased traffic and decreased the corresponding traffic limit for 500 hosts to a value of 15.69 Mbps, a decrease by a factor of 29. The experiment shows a successful countermeasure using a game-theoretic approach of punishing the attacker on a sufficiently large network.

TCP/UDP Flood DDoS Attack on Fat Tree Topology

Most organizations' attacks target some DNS servers to send large bursts of TCP or UDP packets to the target host. Since data centers follow fat-tree topology architecture, I conducted an experiment to test the proposed algorithm on a fat-tree topology using mininet, with $depth = 3$, and $fanout = 3$. The damping factor value was set to $\delta = 0.9$ for this experiment.

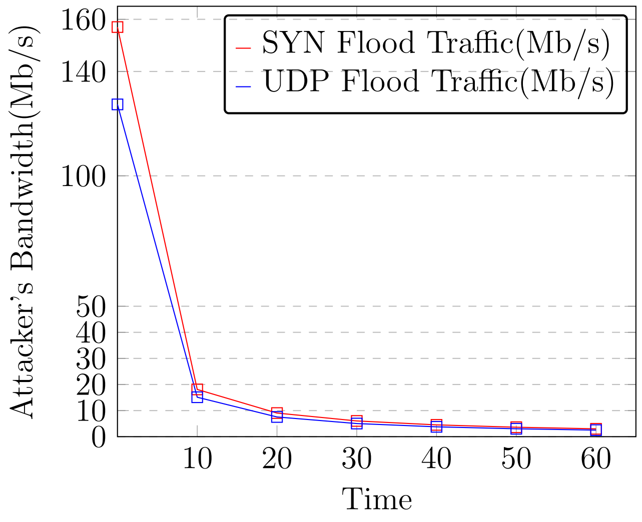


Figure 4.6: TCP and UDP Flood Attack mitigation on FAT Tree Topology.

In this experiment, the usually allowed limit for TCP and UDP traffic set by the SDN controller was 3.0 Mbps. TCP SYN Flood DDoS attack was conducted using python script on a topology of 64 hosts. The traffic decay, once the rate-limiting algorithm based attack countermeasure mechanism is triggered based on IDS alerts, is plotted in red in Figure 4.6. Initially, DDoS traffic is 157.03 Mbps, which violates the permissible limit. The SDN controller pushes the flow to deal with this attack, and the traffic is reduced to 18.11 Mbps at t=10s. The traffic burst reaches a value of 3.02 Mbps at t=50s, nearly equal to the regular traffic rate allowed on this network. Similarly, the traffic pattern for a UDP flood attack starts around ∼127.38 Mbps. The rate limit algorithm decreases this value to ∼15.12 Mbps at t=10s. Traffic rate is further reduced to 4.52 Mbps at t=40s. Finally, traffic burst reaches a cost of ∼3.01Mbps at t=60s, and the algorithm stops further enforcement of the rate limit after t=60s. Thus, it can be seen from this experiment that the algorithm will take less than one minute to mitigate TCP and UDP based DDoS attacks on a sufficiently large network.

4.3 Summary and Discussion

In this chapter, I considered a dynamic game for modeling the interactions between the attacker and defender in a network managed by the SDN controller. A greedy algorithm was employed for calculating the optimal rate limit against a malicious attacker as a punitive mechanism for misbehaving players in a dynamic network game. The optimization algorithm described in this chapter is based on the Nash Folk theorem. This allowed the SDN controller to degrade network bandwidth gracefully, without applying a static hard limit on network traffic. The algorithm can deal with DDoS attacks based on alerts received from the SDN controller. The framework proposed leveraged the benefit of network optimization and programmability offered by SDN quite well. The proposed algorithm can adapt well to varying topologies, as demonstrated by empirical evaluation.

A high value of damping factor $\delta = \{0.8, 0.9\}$ was utilized in this experimental work to put more weight on the future punishment-based payoff to the attacker. The average bandwidth, which I used as a baseline for threshold bandwidth, was selected by observing standard TCP, UDP traffic in a medium sized network for a time duration of about 10-15 minutes. Both these parameters can have an impact on the final results and the convergence time of the algorithm. The motivation behind using a signature-based IDS was to deal with DDoS attacks whose signature can be easily identified. Most of the anomaly detection methods that were analyzed before the experimental setup of this work suffered from false alarms. A limitation of the empirical evaluation is the number of host subprocess we can spawn using the multiprocessing thread, which is currently limited to around 500. A cloud framework based on OpenStack can be used to deal with this scalability concern. In the next chapter, I will consider the games that explore the concept of multi-stage attacks.

Chapter 5

SDN-BASED MOVING TARGET DEFENSE FOR MULTI-STAGE ATTACKS

Moving Target Defense (MTD) techniques have been devised as a tactic wherein a system's security is enhanced by having a rapidly evolving system with a variable attack surface, thereby giving defenders an inherent information advantage. An effective countermeasure used in MTD is network address switching, which can be accomplished in SDN with great ease. SDN-enabled devices can delay the network attack propagation by hiding the real response and replying with a random response to confuse the attacker. As a result, the attacker will believe that random ports are open in the target environment. The attack cost will be increased since the attacker will need to distinguish the real reply from the fake reply. SDN-enabled devices can also introduce random delays in TCP handshake requests that will disrupt the attacker's reconnaissance process for the identification of TCP services. The *cost-benefit* analysis of MTD adaptations against network mapping attempts has been discussed in Kampanakis *et al.* (2014). In this chapter, I elaborate on SDN-based MTD for multi-stage attacks. I mainly discuss Advanced Persistent Threat (APT), which can be modeled using a two-player Markov Game model. I describe a domain-specific reward model and propose an optimal countermeasure selection strategy for the Markov game, which outperforms min-max pure strategy and uniform random strategies.

5.1 SDN-based Markov Game Modeling for Multi-Stage Attacks

The differences between multi-stage attacks like APTs and common cyberattacks make it arduous to use traditional defenses and pre-specified threat models to address APTs as a whole. In this regard, proactive defenses can prove to be effective against

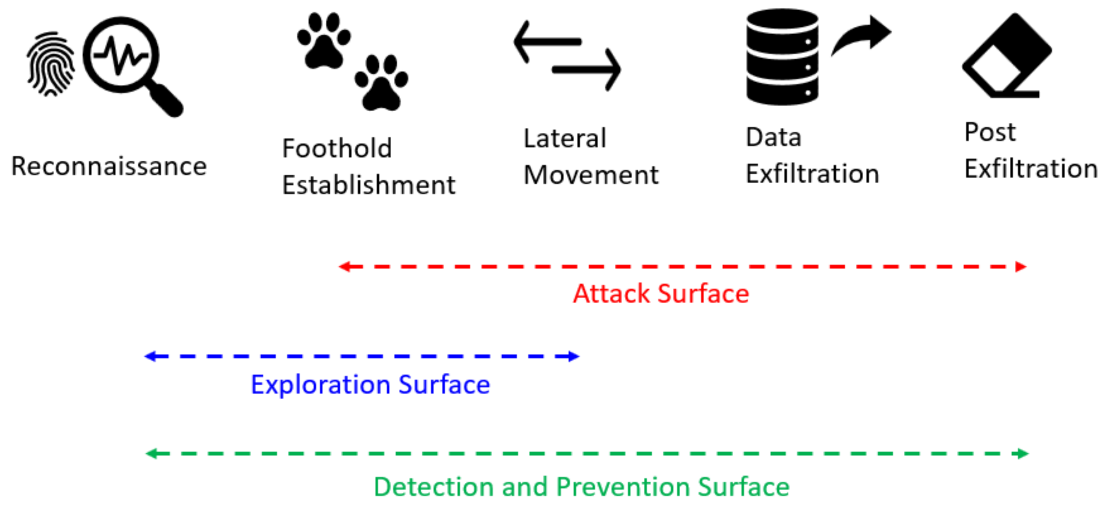


Figure 5.1: A Mapping of Different Phases of Multi-Stage Attack Advanced Persistent Threat (APT) and Various Surfaces of a Cyber System that MTD Seek to Move.

APTs. While MTDs, as will be discussed later in this chapter, make it difficult for APT attackers by dynamically shuffling various system components (see Fig 5.1). The other proactive defenses, such as cyber-deception, can prove to be effective in gathering threat-model information. For example, Shu *et. al.* propose a cyber deception to protect *FTP* services against APT attackers Shu and Yan (2018). In their research work, a defender reroutes attack traffic to a host, which may be a honeypot. The defender ensures that an attacker cannot notice a connection difference between the real IP address and the honeypot trap. By observing the attacker's behavior on the honeypot, the defender updates the threat-model and, in turn, hardens their FTP services. A key aspect of this work is to make the attackers continuously believe that they are interacting with the original environment instead of a honeypot. Having defined the notion of Attack Graphs in earlier chapters, I will now introduce the concept of Markov Games, that can be used for efficiently modeling multi-stage attacks like APT.

We will use the information obtained attack graphs such as the attacker's state, vulnerability severity, access complexity, Common Vulnerability Scoring System (CVSS) score to define the various aspects of the Markov Game.

5.1.1 Markov Game

Definition 5.1.1 *Shapley (1953) defines Markov Game for two players P_1 and P_2 using the tuple $(S, A_1, A_2, \tau, R, \gamma)$ where,*

- $S = \{s_1, s_2, s_3, \dots, s_k\}$ *are finite states of the game*
- $A_1 = \{a_1^1, a_1^2, \dots, a_1^m\}$ *represents the possible finite action sets for P_1*
- $A_2 = \{a_2^1, a_2^2, \dots, a_2^n\}$ *are finite action sets for P_2*
- $\tau(s, a_1, a_2, s')$ *is the probability of reaching a state $s' \in S$ for state s if P_1 and P_2 take actions a_1 and a_2 respectively*
- $R^i(s, a_1, a_2)$ *is the reward obtained by P_i if in state s, P_i and P_{-i} take the actions a_1 and a_2 respectively, and*
- $\gamma^i \mapsto [0, 1)$ *is the discount factor for player i. In the rest of the chapter, I assume $\forall i\ \gamma^i = \gamma$.*

The concept of the optimal policy for a player P_i in this game is defined as selecting the action that optimizes the value of a being in any state s. The optimal policy performs reasoning over the expectation of (1) underlying domain stochasticity (defined by τ and similar to Markov Decision Processes) and (2) reasoning over the other's player P_{-i} action space. This reasoning is generally done by finding a min-max policy over the action spaces of both the players in each state, similar to solution strategies in normal (i.e., matrix) or extended form games Littman (1994).

Notice that in a two-player Markov Game, each state represents a Matrix Game. The policy in each game is based not only on maximizing the reward in this game but also on the reward to go, which depends on the games that you are yet to play. Thus, the min-max strategy seeks to maximize the max player's value, given that the min player selects the pure strategy that gives the minimum pay-off to the max player.

To prevent being second-guessed by the min player, the max player should play a mixed strategy, i.e., have a probability distribution over the actions it can play. To formalize this, let us define the Q-values for an action a_1 taken by the max player P_1 in-state s, given that P_2 selects a_2, is defined as,

$$Q(s, a_1, a_2) = R(s, a_1, a_2) + \gamma \sum_{s'} \tau(s, a_1, a_2, s') \cdot V(s') \tag{5.1}$$

Let the mixed policy for state s as $\pi(s)$, which is a vector of length m that represents the probability distribution that P_1 can has over the possible m actions it can take in-state s. I can now define the value of state s for P_1 using the equation,

$$V(s) = \max_{\pi(s)} \min_{a_2} \sum_{a_1} Q(s, a_1, a_2) \cdot \pi_{a_1} \tag{5.2}$$

5.1.2 Scoring Metrics for Vulnerabilities and Exploits

Software security metrics are defined in terms of Confidentiality, Integration, and Availability McCumber (1991). In a broad sense, an attack on a web application is defined as a *act* that compromises any of these characteristics. The proposed model identifies the vulnerability properties of the VMs (VM1, VM4) present on tenant 1 and tenant 2 in Figure 2.1. It is possible to map vulnerabilities present in each of them to (known) CVE. These vulnerabilities correspond to the attacker's actions, along with a brief description shown in Table 5.1. The use of the Common Vulnerability Scoring System (CVSS) for rating attacks is well studied in security Houmb *et al.* (2010). For (most) CVEs listed in the NVD database, I consider a six-dimensional CVSS v2 vector. This vector can be decomposed into multiple components that represent Access Complexity (AC), i.e., how difficult it is to exploit a vulnerability, and the impact on Confidentiality, Integrity, and Availability (CIA) gained by exploiting a vulnerability.

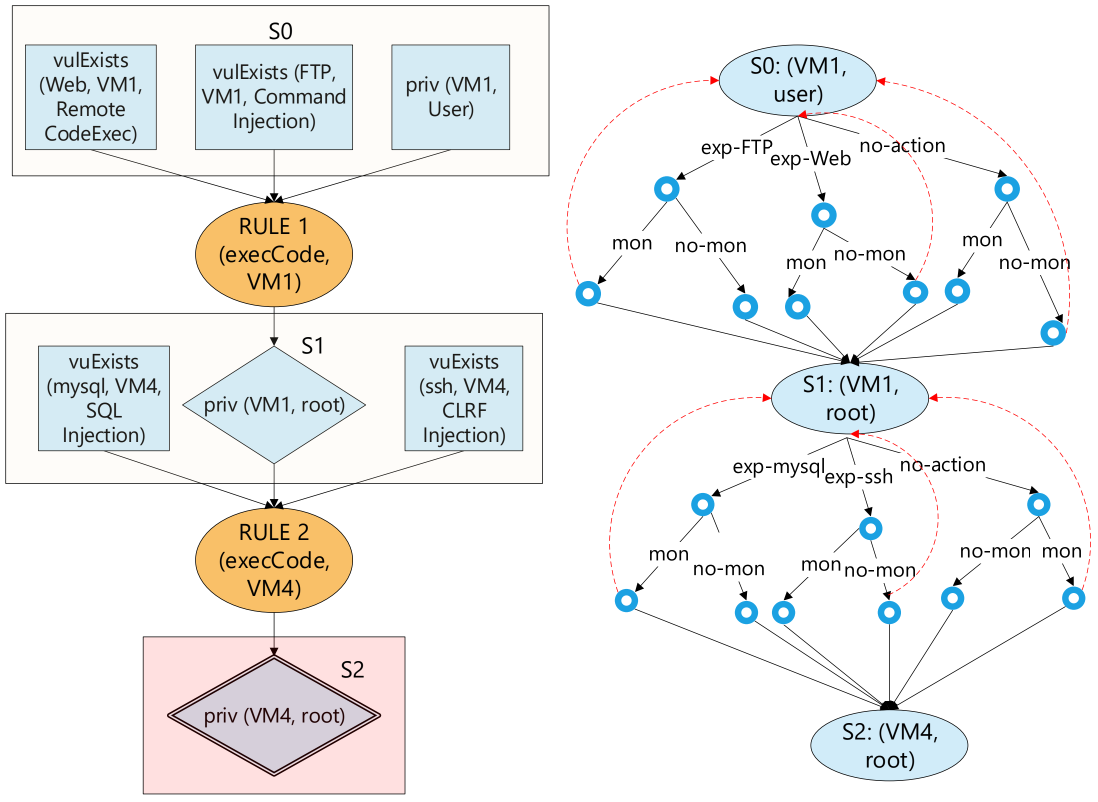
VM	**Vulnerability**	**CVE**	**CIA**	**AC**
VM1 (Web Server)	Cross Site Scripting	CVE-2017-5095	7.0	LOW
VM1 (FTP Server)	Remote Code Execution	CVE-2015-3306	10.0	MEDIUM
VM4 (SSH Server)	CLRF Injection	CVE-2016-3115	5.5	HIGH
VM4 (MySQL Server)	SQL Injection	CVE-2018-11309	7.5	HIGH

Table 5.1: Vulnerability Information for the Cloud Network.

The values of AC are categorical {`EASY, MEDIUM, HIGH`}, while CIA values are in the range $[0, 10]$ and are shown for each CVE in Table 5.1. The AC values can be converted to probability values between $(0, 1]$ to measure how difficult it is for an attacker to exploit a vulnerability, i,e., {`EASY=0.9, MEDIUM=0.6, HIGH=0.3`}. I have utilized these values as transition probabilities from one state to another when the attacker and defender take a particular action in a state.

5.2 Game Theoretic Modeling

Using the example cloud network in previous sections, I described MTD countermeasure selection using a game-theoretic framework. I will formulate a *zero-sum* Markov Game to showcase a situation-aware mechanism for monitoring the cloud network. Markov Game model described it with the following implicit assumptions – (1) It is possible to model cyber attacks using the Markovian Model. I used attack graphs to identify different parameters of the Markov Game (2) Both attacker and the defender have a fully observable environment (3) It is not possible to fix known vulnerabilities because of potential network downtime, fear of misconfiguration that can result from change or lack of human resources (4) Attacker remains



(a) Attack Graph (b) Markov Game Graph

Figure 5.2: The (a) Attack Graph and (b) Markov Game Graph Corresponding to the Threat Model in the Cloud Network.

undetected until he/she attempts to exploit the existing vulnerability, i.e., stealthy attacker Venkatesan *et al.* (2016)

5.2.1 States

The Markov Game model considers the states of the Markov Game as an abstraction over a set of $N_f \cup N_d \cup N_r$ nodes in the Attack Graph (AG). In the example - Figure 5.2, the nodes highlighted by rectangles, serve as the states. I define three possible states shown for the attacker, i.e., $S_0 = \{priv(VM1, User\}$, $S_1 = \{priv(VM1, root\}$, $S_2 = \{priv(VM4, root\}$. The state space formulation of Markov Game based on

		P_1 (Defender)		
		no-mon	mon-Web	mon-FTP
	no-op	$0, 0$	$2, -2$	$2, -2$
P_2 (Atk.)	exp-Web	$7, -7$	$-5, 5$	$10, -10$
	exp-FTP	$10, -10$	$10, -10$	$-7, 7$

		P_1 (Defender)		
		no-mon	mon-ssh	mon-mysql
	no-op	$0, 0$	$3, -3$	$3, -3$
P_2 (Atk.)	exp-ssh	$5.5, -5.5$	$-5.5, 5.5$	$5.5, -5.5$
	exp-mysql	$7.5, -7.5$	$7.5, -7.5$	$-7.5, 7.5$

Table 5.2: Reward (R_2, R_1) for States s_0 (Top) and s_1 (Bottom).

attack graph, must satisfy two two properties:

1. **Uniqueness:** Highlighted privileges node, e.g., $priv(VM1, User)$ from the attack graph map to individual state in the Markov Game. This eliminates the possibility of duplicate counting of the exploits.

2. **Completeness:** Each conditional node, marked in blue in Figure 5.2 (a), maps to a state in the Markov Game. This ensures, no known vulnerability, exploit is missed.

5.2.2 Players and Action Sets

The action set for the player P_1 (defender) A_1 involves selecting the appropriate countermeasure, e.g., placing a monitoring agent (e.g., NIDS) for detecting exploits against vulnerable services. As can be seen in Figure 5.2 (b), the action $a_1^2 =$ `mon-Web` means defender has deployed a *Snort IDS* for monitoring traffic on the port hosting web service. Table 5.2, shows possible actions for defender along the column for both states, i.e., s_0 and s_1.

The action set for the player P_2 (attacker) A_2 has been defined along the rows of table5.2. For instance in state s_0, attacker can either exploit web service $a_2^2 =$ `exp-Web` or FTP service, i.e., $a_2^2 =$ `exp-FTP`.

5.2.3 Transitions

The possible transitions in the Markov Game are shown in Figure 5.2(b). The attacker has user access on the VM1 initially, i.e., $s_0 =$ `priv (VM1, User)`, and the eventual goal of the attack is to obtain root access on VM4. The multi-stage attack mounted by the attacker needs to compromise VM1 and get elevated privileges before compromising VM4.

The transitions $s_0 \rightarrow s_1$ and $s_1 \rightarrow s_2$ depends upon the actions of each player. For instance, if defender, P_1 chooses not to monitor any service in state s_0, whereas, the attacker P_2 chooses to attack web service, with a high probability attacker will be able to transition to state s_1. I assume the probability of a successful attack depends upon access complexity, i.e., if access complexity defined for vulnerability in Table 5.1 is `EASY`, attack success probability will be high. Note that I described *Access Complexity* (AC) can be related to transition probabilities {`EASY=0.9, MEDIUM=0.6, HIGH=0.3`}

Thus, $a_1^1 =$ `no-mon` , $a_2^1 =$ `exp-Web`; $s_0 \times a_1^1 \times a_2^1 \times s_2 = 0.9$

5.2.4 Rewards

Since I use a `zero-sum` Markov Game, the reward for the defender will be negated value of the attacker's reward. Table 5.2 shows reward values for each action combination of both players. The reward values are obtained using (1) impact score `IS` of each attack (2) the performance cost induced by the detection agents in the network. The reward value of the attacker is max. of the parameters performance cost, Impact Score (IS). Consider the second row and second column in the top table, which corresponds to state s_0. The defender chooses to monitor web service, `mon-web`. The attacker tries to exploit Web Server, i.e., `exp-web`. The attacker incurs a negative reward, `-7`, and under a zero-sum game, the defender gains a positive reward `7`. Similarly, if the defender chooses to monitor web server or FTP server, i.e., `mon-web`, `mon-ftp` in-state s_0, top table, cells (1,2), (1,3), he incurs penalty corresponding to performance cost, i.e., `-2`. In a different combination of action sequences, consider row 3 and column 2 of the bottom table. The defender chooses to monitor MySQL service, while the attacker decides to exploit ssh, i.e., `exp-ssh`. The defender incurs cost corresponding to exploited vulnerability on MySQL service worth `-7.5`. The attacker gains the positive reward of `7.5`. In effect, the payoff's for cell `(3,2)` are `7.5,-7.5` for the attacker and the defender respectively. The penalty of monitoring services closer to the goal node, i.e., `mon-mysql` and `mon-ssh`, is higher compared to the top table since I consider services to be critical for the customers. Hence, the performance penalty is weighted higher for them, i.e., `-3` in the first row of the bottom table, compared to `-2` in the top table.

Next, I use the OpenStack cloud network, shown in Figure 2.1, to show the effectiveness of the optimal Markov Game strategy against naive baseline methods that are popular in the cybersecurity community.

Min-Max Pure Strategy (MMPS). The defender selects a pure strategy a_1 given that the attacker selects the action that gives the defender the minimum value. This is similar to the min-max computation for fully observable, deterministic games like chess and can be mathematically represented by replacing $\pi(s)$ in equation 5.2 with a_1 to obtain,

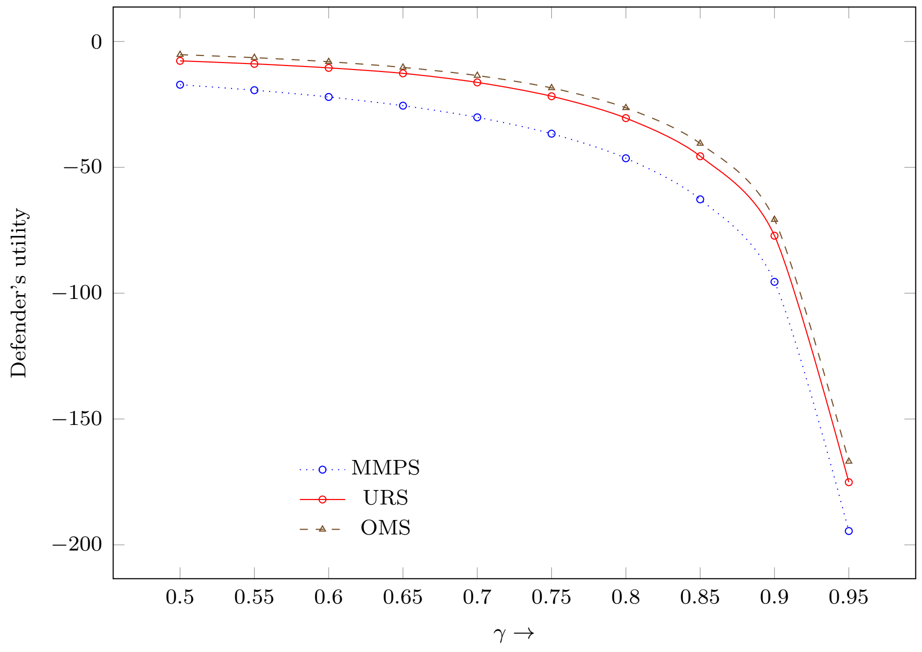
$$V(s) = \max_{a_1} \min_{a_2} Q(s, a_1, a_2) \tag{5.3}$$

If there exists a pure strategy min-max equilibrium for the Markov Game, i.e., a static placement of IDS that clearly dominates any other placement in regards to security and performance, this would have been the optimal strategy. I do not expect this to happen in real-world scenarios and, thus, introduce the notion of Moving Target Defense (MTD) that argues in favor of a mixed strategy that, as opposed to a pure strategy, makes it harder for the attacker to second guess the defender's move. Having said that, MMPS is the best static placement strategy that a defender can come up with performance constraints. In most cases, this strategy is better than strategies used by many network administrators. Thus, MMPS acts as a reasonable baseline.

Uniform Random Strategy (URS) In URS, the defender uses a uniform probability distribution over its actions (or pure strategies) in a state. For example, consider state s_1 shown in Table 5.2. The defender uniformly chooses the mixed strategy of monitoring the `mysql` server, the `ssh` server, or none of them. Thus, in any round, the defender rolls a three-sided fair dice and does whatever comes up. Many researchers had claimed that selecting between what to choose when shifting attack surfaces should be done using a pure (or uniformly) random strategy Zhuang *et al.* (2014). This has been disproved later by Sengupta *et al.* (2017). In this work, I use this as a baseline to reiterate that such strategies based on intuition, as opposed to careful modeling of the problem at hand, can do more harm than good.

5.2.5 Results

I compared the results of *Optimal Mixed Strategy* (OMS) Chowdhary *et al.* (2018d) with *Uniform Random Strategy* (URS) and *MaxMin Pure Strategy* (MMPS) as shown in the Figure 5.3. At higher values of the discount factor (near 0.85), the high magnitude of reward in the terminal state affects the values of other states, thereby increasing the magnitude of gain. When the discount factor is small (near 0.5), the rewards in the future state does not have a substantial impact on the immediate value of a state, thereby reducing the magnitude of gain. In both states s_0 and s_1, the defender's reward is maximum using OMS strategy, e.g., for discount factor



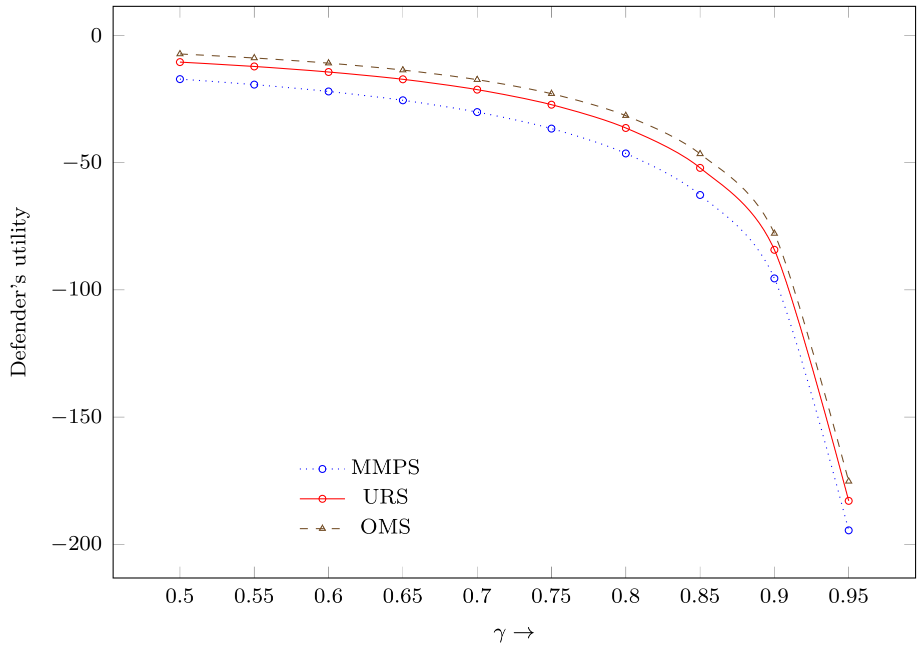


Figure 5.3: Defender's Values for States s_0 (Left), s_1 (Right).

$\gamma = 0.9$, the defender gets -40.5 in-state s_0, and -46.5 s_1 as shown in the Figure 5.3. The defender gets rewards $s_0, s_1 =$ (-52.0, -66.0) using URS and $s_0, s_1 =$ (-60.0, -62.0) using MMPS. Clearly, rewards for defender are better when using OMS based on value-iteration methods I used in this research work.

The optimal strategy for the defender for the states s_0 and s_1 is as follows for the discount factor $\gamma = 0.8$:

```
pi(0) : {'no-mon': 0.43, 'mon-web': 0.26, 'mon-ftp': 0.30}
pi(1) : {'no-mon': 0.0, 'mon-ssh': 0.5, 'mon-mysql': 0.5}
```

The optimal strategy on convergence for the zero-sum Markov game is to perform FTP monitoring *mon-ftp* with probability 0.30 and Web monitoring *mon-web* with probability 0.26, while no monitoring action *no-mon* has been suggested in optimal strategy for state s_0 with probability 0.43. Similarly, the optimal strategy for the defender is to monitor both the SSH server and SQL server with equal probabilities in-state s_1. In this particular case, these results indicate that a defender can focus on performance at places near the entry points but should prioritize security in the states close to the goal.

5.3 MTD Against Advanced Persistent Threats

The attacker performs a multi-stage attack, targeting the services at the gateway of the network first and then trying to penetrate into internal network services. The goal of this attack is to exfiltrate as much information as possible while maintaining persistence over a long period of time. Most attack detection tools just utilize signature-based tools in order to identify the data at the border of the network. Additionally, the tools are configured in an ingress filtering mode. Hence the data going out of the network is left unexamined.

Based on the standards defined by NIST and other organizations Brewer (2014), the attack analysis from a defender/security administrator's perspective takes place in five steps, namely: 1) Reconnaissance/ Intelligence Gathering 2) Threat Modeling 3) Vulnerability Scanning and Analysis 4) Exploitation 5) Post Exploitation

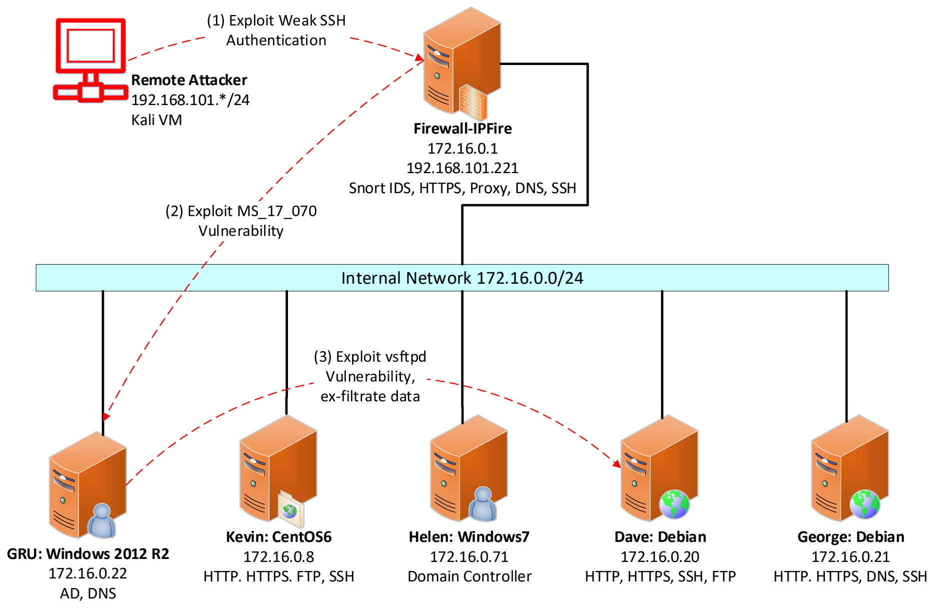
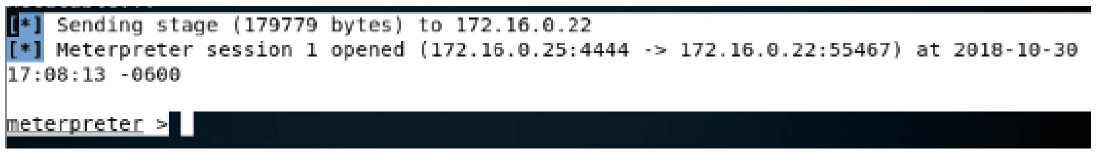


Figure 5.4: An Advanced Persistent Threat (APT) Scenario.

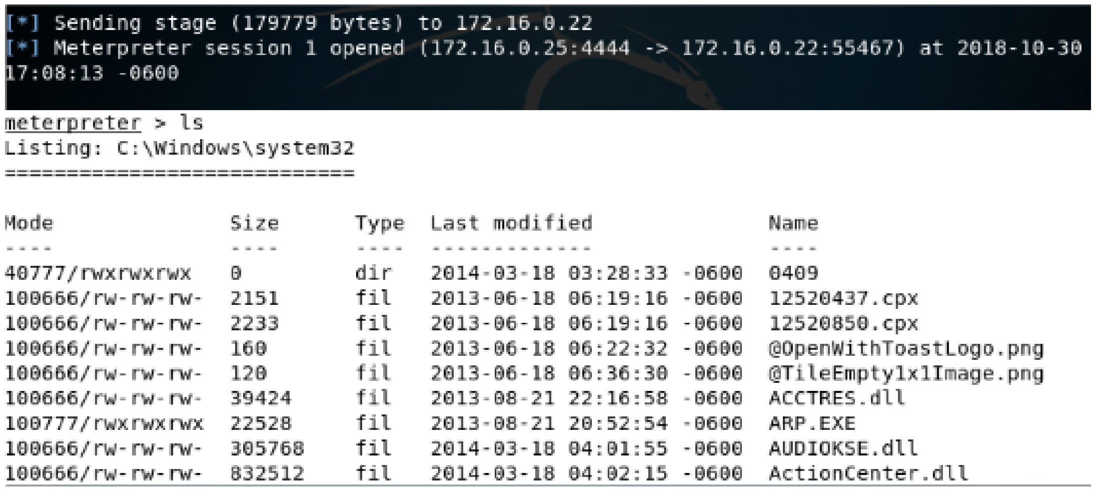
In order to simulate an APT scenario, I created a flat network using the VM images from the Western Region Cybersecurity Defense Competition (WRCCDC) Competition (2018). The competition consists of eight Blue Teams from different regions who face a team of experienced hackers (Red Team) from the Industry. The goal of Blue teams is to maintain service availability while ensuring malicious attempts by Red Team members are logged and reported properly. In the experimental analysis, I focused on how effectively the model can detect attacks by the Red teams.

I used the VM images from the competition and created a similar environment in ASU's Science DMZ Chowdhary *et al.* (2017a). I created a flat network with *IPFire (Next-Generation Firewall)* hosted at the gateway of the network. The VM has the capability to implement traditional Firewall filtering capability. Additionally, the VM has an integrated VPN, Snort IDS, Web Proxy for threat detection at different levels of the protocol stack. Now, I describe the various stages of APT (loosely based on the NIST model) carried out by the Red Teams over an extended period of time.



```
[*] Sending stage (179779 bytes) to 172.16.0.22
[*] Meterpreter session 1 opened (172.16.0.25:4444 -> 172.16.0.22:55467) at 2018-10-30
17:08:13 -0600

meterpreter >
```



```
[*] Sending stage (179779 bytes) to 172.16.0.22
[*] Meterpreter session 1 opened (172.16.0.25:4444 -> 172.16.0.22:55467) at 2018-10-30
17:08:13 -0600

meterpreter > ls
Listing: C:\Windows\system32
============================

Mode               Size     Type  Last modified              Name
----               ----     ----  -------------              ----
40777/rwxrwxrwx    0        dir   2014-03-18 03:28:33 -0600  0409
100666/rw-rw-rw-   2151     fil   2013-06-18 06:19:16 -0600  12520437.cpx
100666/rw-rw-rw-   2233     fil   2013-06-18 06:19:16 -0600  12520850.cpx
100666/rw-rw-rw-   160      fil   2013-06-18 06:22:32 -0600  @OpenWithToastLogo.png
100666/rw-rw-rw-   120      fil   2013-06-18 06:36:30 -0600  @TileEmpty1x1Image.png
100666/rw-rw-rw-   39424    fil   2013-08-21 22:16:58 -0600  ACCTRES.dll
100777/rwxrwxrwx   22528    fil   2013-08-21 20:52:54 -0600  ARP.EXE
100666/rw-rw-rw-   305768   fil   2014-03-18 04:01:55 -0600  AUDIOKSE.dll
100666/rw-rw-rw-   832512   fil   2014-03-18 04:02:15 -0600  ActionCenter.dll
```

Figure 5.5: Stage 2 of the APT Scenario Described Above.

Stage 1: Slow and Low Weak Authentication Exploit The attacker performs social engineering on website forums frequented by employees of the company. One of the developer's posts a question regarding a key update function for OpenSSH functionality with a specific version OpenSSH v3.3. The attacker identifies this version as being vulnerable to authentication based attack, which can exploit a buffer-overflow vulnerability by sending a well-defined payload to the SSH server hosted at the gateway of the network. In this particular case, I already knew the vulnerable OpenSSH service. I consider this as the first step of a multi-stage attack (see Figure 5.4). This represents a scenario of how both the players become aware of a known vulnerability present in the system.

Stage 2: Exploiting Windows 7 VM 172.16.0.22 The attacker probes the network and identifies the services and OS versions running on the hosts in the network. In the experimental setup, the corporate access control policy allows only Windows

VM	Vulnerability	CVE	CIA	AC
Firewall	SSH Buffer Overflow	CVE-2017-6542	7.5	MEDIUM
Win 2012	Eternal Blue SMB	MS17-010	9.3	HIGH
	Remote Code Execution	MS15-034	10.0	HIGH
Debian	Anonymous FTP Login	CVE-1999-0497	6.4	MEDIUM
Win 7	MSRPC Service Enumeration	CVE-2008-4250	5.0	MEDIUM
	NVT OS End of Life	CVE-2008-4114	10.0	HIGH
CentOS 6	OpenSSL MITM	CVE-2017-3737	6.8	MEDIUM

Table 5.3: Vulnerability Information for the APT Scenario.

systems to interact with resources such as FTP, Web Servers. Thus, the attacker needs to obtain access to a root shell on one of the Windows machines. In order to accomplish this, the attacker must target the `MS_017_10` vulnerability present on a Windows 2012 R2 server-GRU as shown in Figure 5.5, which hosts other services such as Active Directory and Domain Name Server (DNS).

Stage 3: Exploiting *vsftpd* vulnerability and exfiltrating data The vsftpd service running on machine Dave has a Debian operating system. The vulnerability on the FTP server can be exploited by the attacker, and they can create a backdoor channel to exfiltrate data from the FTP server to their command and control center ($C\&C$). Since most organizations have no egress filtering policies for the corporate firewall, so data exfiltration often goes unnoticed. Additionally, the attacker can distribute the data transfer over a period of several weeks, even if there is some signature-based rule on IDS to prevent data exfiltration.

Stage 4: Post Exploitation The attacker can either use the *meterpreter* (a Kali Linux tool) shell on Windows host to perform privilege escalation and disrupt services if they are a rogue insider or use the windows machine as a jump point for exploiting other machines. The rationale behind exploiting the Windows machine first is that Windows acts as a domain controller for many other machines in the network.

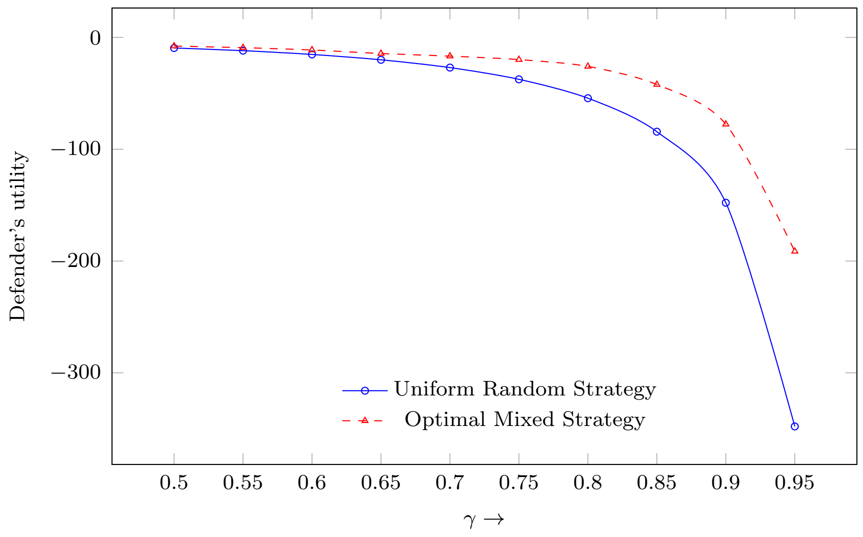


Figure 5.6: Defender's Value in Each of the State s_2 as Discount Factor increases. In this State, we Consider a Sub-system of the entire Cloud Network in which the Defender can place Five Possible Detection Systems but Chooses to Place Two Out of the Five for Performance Considerations.

5.3.1 Attack Analysis and Results

The Blue team identified the following vulnerabilities on the network VMs as shown in the Table 7.1. The attacker can have one or more attack goals. One goal of the attack is to ex-filtrate files from the Debian machine (Dave). Another goal can be to target CentOS 6 (Kevin) and disrupt the Domain Name Server (DNS) for the private network. This will, in effect, lead to service unavailability.

I present the values a defender, i.e., the Blue Team, obtains if they use an optimal strategy for placement of detection systems for state s_2. This state had five possible vulnerabilities and, thus, five possible IDS for detecting them (see Figure 5.6). The model lets the defender provide a limit on the number of IDS systems they can place in this state or sub-net (which was two). It can be seen that, in comparison to the Uniform Random placement strategy in the sub-network represented by state s_2, the optimal strategy for the Markov Game yielded better values. Note that the Markov

Game formulation treats the number of IDS placed in this subnet independently, regardless of how many IDS systems are placed in other parts of the cloud system. This addresses a major shortcoming of previous research Venkatesan *et al.* (2016) in which some pure strategies can place multiple IDS on the same subnet, thereby affecting its performance, which, quality of service in other subnets are not impacted. For APT scenarios, even though the Blue team needs comprehensive logging and monitoring using IDS systems at the granularity of each subnet as well as hosts, monitoring every packet in a cloud network is wasteful in terms of networking and compute resources. I found that the min-max strategy proposed by Markov Game solver in this work helps in optimizing the number of detection agents while ensuring a high detection rate.

5.4 Moving Target Defense Research

Kampanakis *et. al.* Kampanakis *et al.* (2014) discuss SDN-based random host and route mutation. These SDN-based MTD countermeasures increase attacker uncertainty by misreporting information. The blind mutation of key network services can lead to network performance hit. CHAOS Shi *et al.* (2017) analyzes how the delay intentionally introduced by MTD impacts the packet count in an SDN-managed network. The SDN controller utilizes host-mutation and decoy-severs to deceive an adversary. Feedback-driven multi-stage MTD has been proposed by Zhu and Başar (2013) for dealing with multi-stage attacks like Stuxnet Langner (2011). The authors quantify the damage or cost caused by an attacker at different stages of the network. The game between attacker-defender is modeled as a finite zero-sum matrix game with a bounded cost function, and a mixed-strategy Saddle Point Equilibrium (SPE). In Taylor *et al.* (2016), the authors provide metrics for MTD evaluation and risk analysis. For risk metrics, they proposed statistical metrics to study the effect of how the attacker can quickly conduct and succeed in adversarial attacks. The authors assumed the system will always have a running task that can be measured. In my previous research works, I have considered network performance impact while implementing MTD countermeasures Sengupta *et al.* (2018). Predictability oriented defense against adaptive adversaries Colbaugh and Glass (2012) combines game theory and machine learning. The authors model attackers action in ML feature space to provide defense against current and future attacks. I identified an adaptive MTD strategy against multi-hop monotonic attacks for cloud networks, which optimizes for performance while providing gains in security.

The categorization of the various software surfaces opens up the possibility of considering other logical surface level distinctions and, thus, MTDs that shift these surfaces. For example, *Microsegmentation* Mämmelä *et al.* (2016) is a method of creating secure zones in data centers and cloud deployments to isolate workloads from one another and secure them individually. With MTD formalism, one can test the existing hypothesis and develop new ones for micro-segmentation. I believe that formal modeling, in line with Chowdhary *et al.* (2018d), one might discover that advanced services will be more effective when applied at a granular-level (as close to the application as possible in a distributed manner).

5.5 Summary and Discussion

The Configuration Set: Works in MTD have concentrated mostly on the movement of the exploration, the detection, and the attack surface. The movement of the prevention surface, which comprises of security modules such as Firewalls, IPS, , *etc.* has only been investigated by a couple of works. An MTD research can consider exploring Next-Generation Firewall (NGFW) architectures that combine security modules such as firewall, content filter, anti-virus *etc.* to provide a multi-layered defense-in-depth solution. Some current implementations of NGFW that can be leveraged for testing the effectiveness of these defenses are *Cisco ASA* Frahim *et al.* (2014) and *PAN Firewall* Alto (2018).

The Timing Function: The timing problem has mostly been ignored in previous works on MTDs. While some works perform empirical studies to test the best (constant) time-period for switching, they can be highly specific to the threat model and the elements being shifted. Note that the handful of works that empirically determine time-periods are by no means complete. An extensive study is required just to come up with reasonable time periods for particular surfaces, how it affects that attack model, and provide guidelines on efficient implementation methods. On the other hand, a few existing approaches that do address the timing problem theoretically suffer from scalability issues. In Lei *et al.* (2017), the authors land up with an increasing number of states in their Markov Game by including time as a parameter. Inferring the defender's strategy in such Markov Games cripples the MTD to work beyond small networks.

The Movement Function: Several works have argued that often the defender has performance information about their system and known attacks that can be used to exploit their system. In such cases, game-theoretic modeling can result in better strategies that offer higher gains in terms of both security and performance metrics. Unfortunately, the latter works suffer from scalability issues, encouraging practitioners to default to URS. Works that can leverage existing knowledge without suffering from scalability issues can fill an important gap in MTD research.

General-Sum Markov Games In the case of a general-sum Markov Game, one player can infer other players' strategies before making a move. Thus, the min-max strategy becomes sub-optimal in a general-sum Markov game. The equilibriums such as Strong Stackelberg Equilibrium (SSE) dominates min-max equilibrium in these games Vorobeychik and Singh (2012); Guerrero *et al.* (2017). We extend the zero-sum games to general-sum games in my work Sengupta *et al.* (2019) and show that SSEs are a subset of Nash Equilibrium for our Markov Game setting. The empirical evaluation shows that SSE dominates the Uniform Random Strategy (URS) and min-max strategy for the problem of strategic placement of monitoring systems in a network.

Although I can select states for the Markov Game such that the number of actions in each state is restricted to allow this computation, such abstraction of the Attack Graph may not be practically meaningful. I hope I can investigate and address this issue in the future.

Current research often makes strong assumptions about the threat model. This makes results regarding the effects of such defenses questionable. In the future, I hope to see more studies that try to figure out realistic attack scenarios such as the simulated APT attack that has been discussed in this research.

Chapter 6

INTENT-DRIVEN SECURITY POLICY MANAGEMENT FOR SOFTWARE DEFINED SYSTEMS

In a multi-domain software-defined system (SDx), different network controllers are utilized to manage networking resources. However, these controllers operate by using a different high-level language (intent), and the administrator needs to do cross-layer translation from the user requirements to the underlying network controller format, which brings increased human-in-the-loop overhead. There are two primary security and management challenges involved in managing multi-domain controllers. The first challenge is how to design an SDN controller language that can effectively convert human specified networking policies at the control plane into network flow-level rules at the data-plane. The second challenge is how to reduce the complexity of network flow rules conflict checking at the data-plane. In this chapter, I present a new intent-based security policy enforcement solution, called INTPOL: *a)* INTPOL provides a unified language using INTPOL grammar that abstracts the network administrator from the underlying network controller's rule format. *b)* INTPOL develops a networking service mapping solution to use a bounded formal model for network service compliance checking, significantly reducing the complexity of flow rule conflicts checking from PSPACE to Linear Time. *c)* INTPOL is expendable from a single SDN domain to multiple SDN domains by considering the network service function chaining as inter-SDN-domain policy management.

6.1 SDN Preliminaries and Problem of Policy Management

Software-Defined Networking (SDN) simplifies network management by decoupling the control plane and data plane. The SDN architecture can be divided into three tiers (layers), as discussed by Kreutz *et al.* (2015) - *application plane* - programs that communicate behaviors and needed resources with SDN controller using application programming interfaces (API). The *control plane* comprises of SDN controller, which receives instructions from the application plane, analyzes the requirements, and, based on the controller functionality, populates the data plane forwarding tables. The forwarding decisions are flow-based, i.e., defined by a set of packet field values acting as match (filter) criteria and a set of actions (instructions). The *data plane* (user plane) consists of networking devices (OpenFlow switches) that implement the decisions enforced by the control plane.

In a Software-Defined Networking (SDN) environment, the SDN control-plane manages a global network view. In a multi-domain SDN networking environment, as pointed out in Sherry *et al.* (2012), policy misconfiguration in middleboxes (network functions) is a common cause of failure. The safety and security properties of networks called *Invariant* need to be verified in a scalable fashion to ensure the smooth functioning. To illustrate the described problem, I present the existing network flow rule generation and management in Figure 6.1. The SDN control plane is the interface between the application plane and the data-plane. It is managed by network administrators to translate human specified service requirements and networking policies to the machine-level flow rules.

The SDN applications communicate with SDN controllers situated at the control plane using application programming interfaces (APIs). These applications provide an abstracted view of the network administrator to create different security policies

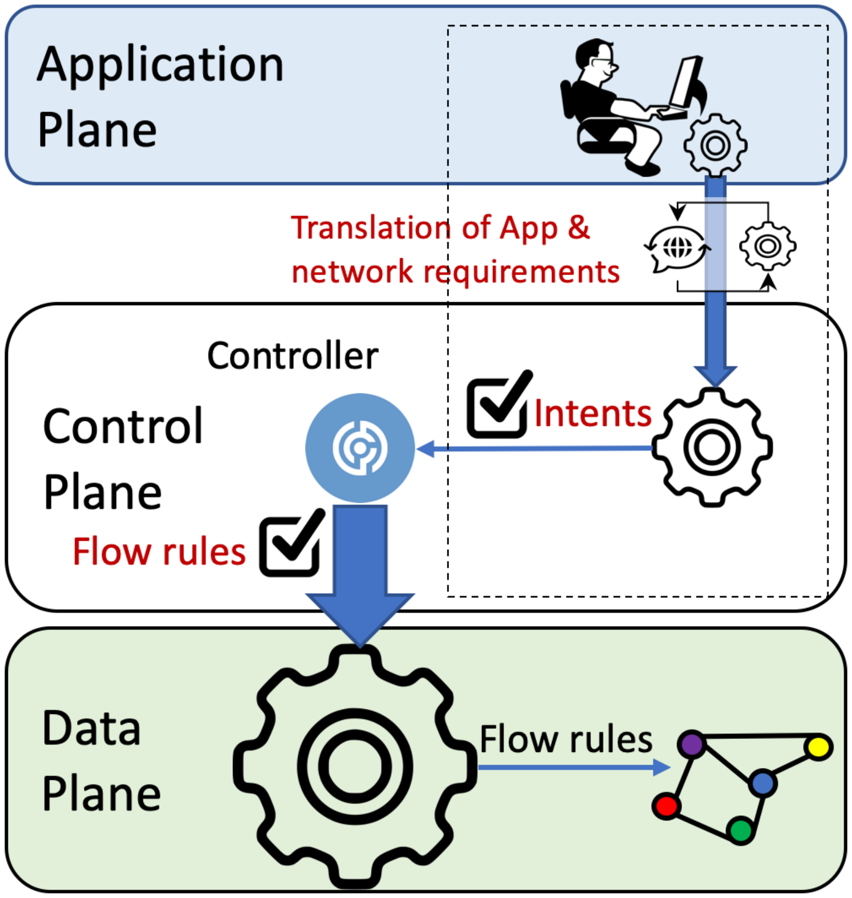
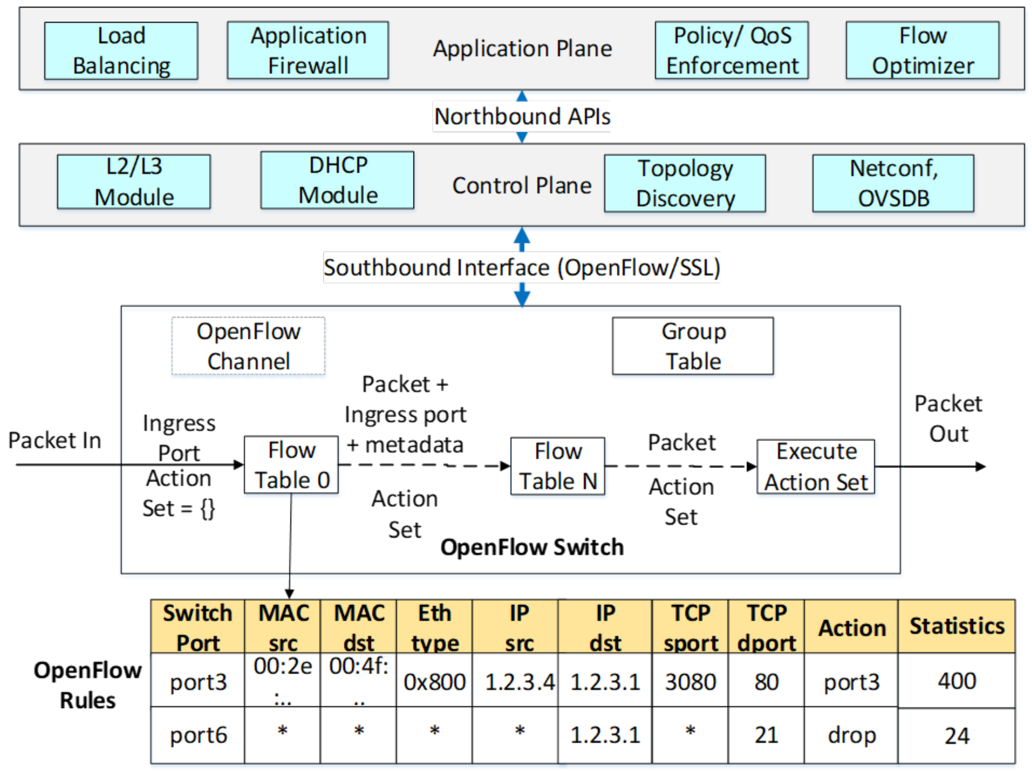


Figure 6.1: SDN Control Flow for Multi-tiered Network Policy Checking Using INT-POL Framework.

and mission requirements. The SDN control-plane consists of an SDN controller with varying modules like layer 2/3 (L2-L3) traffic handler and topology discovery module to listen to any topology changes. The SDN controller also extracts information about data-plane devices and network configuration and relays the information to the application plane using Northbound RESTful APIs, as shown in Figure 6.2.

Existing intent-based networking approaches Jang *et al.* (2014); Pham and Hoang (2016) requires the network admin to do a cross-layer translation from the application and network requirements to the controller's intents. Then the controller can use the plans to generate underlying network flow rules. Current work Pisharody *et al.* (2017)



Switch Port	MAC src	MAC dst	Eth type	IP src	IP dst	TCP sport	TCP dport	Action	Statistics
port3	00:2e:..	00:4f:..	0x800	1.2.3.4	1.2.3.1	3080	80	port3	400
port6	*	*	*	*	1.2.3.1	*	21	drop	24

Figure 6.2: SDN Architecture with Details of Different Components and Communication Interfaces

Hu *et al.* (2014) mostly focuses on the flow-rule level conflict checking to ensure the requested network resources allocation is not compromising existing flow and security policies. No current work has proposed to do the network and system policy analysis at the application and intent level, which leads to a significant network management issue. As a result, network flow rules' complexity is dramatically increased from the application-plane to the data-plane. For example, the experimental evaluation on the Stanford topology Kazemian *et al.* (2012) shows that for a 20 generated intents, there are 413 conflicting flow rules.. Thus, the fundamental issue is if conflicts exist at the application-plane level and are not analyzed until they are translated into flow rules in the data-plane, this will significantly increase the policy conflict checking and management overhead.

To address the described issues, this research addresses the following two questions: 1) *how to design a human to SDN controller language to translate human specified networking policies into network flow-level rules effectively* and 2) *how to reduce the complexity of system policy and flow rule conflict checking*? The proposed solution in this work creates a new Intent-driven policy language *INTPOL*, which allows the network administrator to express security policies at the application plane level. The policy designers can create network management and operation policies while remaining abstracted from the underlying SDN controller. As described above, the user-defined policies are analyzed early in the security policy lifecycle to detect potential conflicts using a lightweight formal model checking framework.

The security policies specified at the application plane are parsed for predicates of INTPOL language. A bounded model representation of state changes of the packet is created using NuSMV Cimatti *et al.* (2002) based framework, which captures the state changes of the packet as it moves between network switches and hosts. The INTPOL grammar translates the security policies into the REST API call format of the corresponding SDN controller. The realization of network policies at the data-plane level introduces a lot more flow rules. Detection and resolution of conflicts amongst flow rules at the data-plane level can impact network services' performance. This problem can be dramatically amplified when expanding SDN systems into multiple SDN domains, where inter-domain network and service policies introduce an additional level of complexity. With INTPOL, I will discuss how this framework provides early detection of network policy conflicts at the application and control-plane. This approach will help reduce the scope of flow rule conflict checking at the data-plane level.

The key technical novelty of INTPOL lies in how to ensure consistent behavior in the network, which has been considered a network-wide invariant verification issue

Khurshid *et al.* (2013). Existing solutions, such as Yuan *et al.* (2020) provide a scalable formal solution for network policy verification using optimizations such as model pre-computation and query containment. The issue with these research works is that traditional models such as Linear Temporal Logic (LTL) and Computational Tree Logic (CTL) are PSPACE complete in the worst case Baier and Katoen (2008). This solution limits the scalability of the model. INTPOL models the problem of verifying network policies using a Bounded Model Checking (BMC) Clarke *et al.* (2001) to check for the existence of network policy violation within the bounds placed on the network, e.g., all paths up to length K. This approach ensures sufficient coverage for checking policy violation issues in the system and reduces the space complexity of network policy verification from PSPACE to linear time (linear in the scale of the number of network states).

6.1.1 Contributions

- INTPOL introduces a new intent-based language for translating network policy requirements into a unified format, thus providing a grammar that abstracts network administrators from the policy specification language of underlying network controllers. The language allows easy expression of complex scenarios such as service function chaining and hybrid networks, including traditional and OpenFlow networks.

- The framework minimizes the overhead of policy conflict checking by early identification and correction of policy conflict issues, i.e., policy conflict handling at the application plane instead of the data plane. This approach reduces the overhead induced by policy conflict checking. INTPOL utilizes bounded model checking (BMC) with bounds determined based on network diameter to provide linear time network invariant checking compared to full-scale model checking frameworks.

- The multi-level policy conflict checking reduces the scope of conflict checking at the data plane by handling most conflicts at the application plane. The empirical evaluation on large scale network Stanford Topology Kazemian *et al.* (2012) shows that conflict overhead at the application plane is minimal $\sim$ 250-800x times lower than policy conflict checking at the data plane.

6.2 INTPOL System and Model Description

In this section, I first describe the INTPOL language and its grammar; then, I elaborate on the INTPOL scalable policy checking framework; finally, I present the intent conflict checking at various levels of SDN infrastructure.

6.2.1 INTPOL Language

The language for expressing network orchestration and monitoring should be simple enough for network administrators to understand and use for practical purposes. Moreover, it should have the capability to cover a broad range of network safety and security issues. While languages such as PGA Prakash *et al.* (2015) provide a simple framework for expressing network policies, the number of policies generated fails to scale on an extensive network. To address the dual problem of simplicity and scalability, the INTPOL framework abstracts the intent specification from intent expression, i.e., I utilized a simple application plane that allows network administrators to express the intents such as *security policies*, *QoS requirements* and *service function chaining (SFC)*. The intent requirements, as well as the network topology, are translated into a scalable formal model. In this work, I utilize bounds based on the network's size to check if the network policies or network safety properties are violated. The network

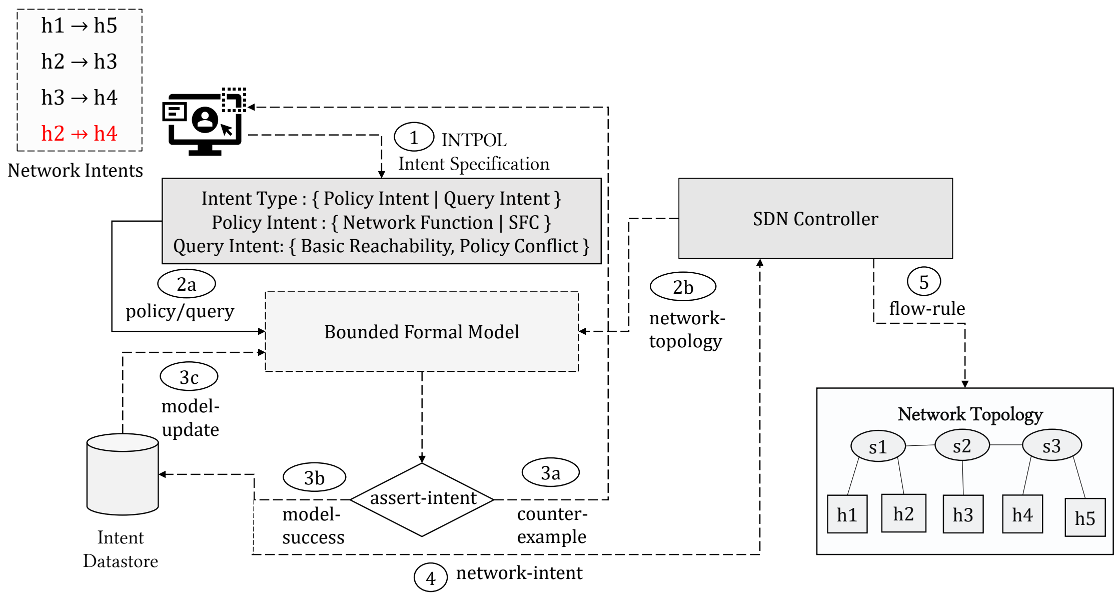


Figure 6.3: Intent Specification and Formal Modeling. The Users can Specify Policy and Query Intent at the Application Plane. The Bounded Model Checking Framework accepts Query Intents to check if Intents Meets Network Reachability and Security Policies.

policies causing conflicts are marked and returned to the user.

The end-to-end handling of network intents at the application plane is depicted in Figure 6.3. Intent can be considered a request similar to Read/Write/Update transactions on the database (underlying network in this research work). The network operator specifies some intents such as traffic allowed between hosts h1 and h5 (h1 → h5), h2 and h3 (h2 → h3), etc, as can be seen from block *Network Intents* in the Figure 6.3.

The intent *Policy Intent* can perform network updates like inserting rule to block traffic (Firewall) or inspecting suspicious traffic (Intrusion Prevention System). The *Query Intent*, on the other hand, helps the network operator in assert safety, auditing access control, and service availability in the network. Once the user specifies a network intent (1), the intent is converted into a query represented as a formal logic formula (2a). The Bounded Formal Model utilizes the network topology information extracted from SDN controller (2b), and existing intents present in an Intent Datastore (2c), to construct/update a bounded Linear Temporal Logic (LTL) model.

The model is evaluated for some basic safety properties such as packet reachability, conflicts with the existing intents in an automated fashion. Suppose the intent violates any network safety properties. In that case, a response is returned to the user with a counter-example from the model, which shows the states in the network that violate network properties (3a). If the intent does not violate any network safety or reachability properties, it is stored in the Intent Datastore (3b), and the bounded model is updated with a new intent (3c). The intent is passed to the SDN controller (4) to create an SDN controller specific intent Pham and Hoang (2016). The SDN controller, in turn, installs flow rules (5) for realizing the user intent at the data-plane level. In the framework discussed in this chapter, the user has the flexibility to specify custom intent queries to check properties for a class of network intents, e.g., "Can all web servers in a network communicate with all database servers?". These queries are translated into a formal model clause and checked over the current model of the network.

Efficient Policy Conflict Detection

Existing research works Pisharody *et al.* (2017); Hu *et al.* (2014) perform the policy conflict checking at the data plane level. I introduce two-level conflict analysis in this research work. The conflict checking at the application plane using a bounded model checking approach can reduce policy conflict detection at the data plane. When an intent is added at the application plane level, it is translated into the controller specific intent command. The SDN controller, e.g., ONOS Berde *et al.* (2014), allows specification of *HostToHost* which allows communication between network hosts, *Point-*

ToPoint intent which allows traffic to pass through two switches' ports. Consider a host intent, e.g., *add-host-intent h1 h5*, based on example network in Figure 6.3. The ONOS controller *a)* identifies the path between hosts, i.e., h1-s1-s2-s3-s5. *b)* generates Openflow rules to establish communication along the path. Additionally, the controller marks *appId* in the flow rule as *org.onos.intent* to show that flow rule was added using intent module. It is noteworthy that for a single host intent, there will be *five* flow rules installed in the example network Figure 6.3 example. If there are conflicts amongst intents at the application plane, it will, in turn, generate flow rules which will conflict with each other.

6.2.2 INTPOL Grammar Description

```
Intent-Type :: Policy-Intent | Query-Intent
Policy-Intent :: Network-Function | Service-Function-Chain
Service-Function-Chain :: Network-Function, Network-Function[*]
Network-Function :: Firewall | IDS | Routing | Load-Balancer
Firewall :: Header, Action
Header :: Hw-Src, Hw-Dst, Src-IP, Dst-IP, Src-Port, Dst-Port,
          Protocol
Action :: Forward | Drop | Modify
Query-Intent :: Check-Conflict | Check-Loop | Check-Reachablity
```

Figure 6.4: INPOL Grammar Describing Different Kind of Intents Which Can Be Specified by the User at the Application Plane. Policy Intent Captures the Network Topology and Packet Propagation Behavior. Query Intent Is Used for Verification of Network Invariants.

Next, I will describe the INTPOL framework's grammar - Figure 6.4, which can be used for interpreting the user requirement at the application plane and translating them to corresponding bounded model program to check necessary network policies. Consider the steps for handling user intents in Figure 6.3. The steps *1-3* are showcasing the application of INTPOL language. If an invariant provided as part of language is violated, the network admin is notified step *3a*. The next example will illustrate the translation of policy intent and query intent into a formal model using Firewall as a sample function. Consider the network topology described in Figure 6.3. In the example rules, there are high-level network intents for hosts, $h1 \rightarrow h5$, $h2 \rightarrow h3$, $h3 \rightarrow h4$, and rule denying traffic between hosts $h2$ and $h4$. Each network host is connected to the network switch using a *layer 2* switch port. Similarly, network switches are also connected with the network controller (ONOS, ODL) using a layer-2 port. The bounded formal model accepts these requirements, and network topology information from the SDN controller, to create a model of packet propagation and changes in the packet state as the packet traverses along different paths from source to destination in a network. Linear Temporal Logic (LTL) Baier and Katoen (2008), a form of model checking technique provided by NuSMV Cimatti *et al.* (2002), in particular, characterizes the linear path induced by the Finite State Machine (FSM) of the network states Baier and Katoen (2008). The next section showcases how this proposed model utilizes the bounds based on the network policies and topology to reduce the number of model checking states at the application plane. Then, I present the overall complexity of handling conflicts at the data plane.

6.2.3 INTPOL Model Checking Framework

The formal models such as LTL suffer from scalability challenge. The total number of model states can be as large as 10^{20} for some models Biere *et al.* (2003). Bounded model checking (BMC) Clarke *et al.* (2001) checks the state space for a counterexample using a user-specified bound k. For each value of k, BMC builds a boolean formula that is satisfiable if a counterexample of length k exists. For a given transition system with s states, the model can be expressed using $k \times s$ variables. Once the bounded model is created, a SAT solver like Vizel *et al.* (2015) is used to check the boolean formula's satisfiability. The bounded model checking formula's completeness is established using techniques such as completeness threshold, liveliness property, and induction tests, as discussed by Biere *et al.* (2003).

Establishing Threshold for Bounded Model Checking

The bound k on model of the network M can be specified in terms of *reachablity diameter rdr(M)*, i.e., minimum number of steps required to reach all reachable states. Another possible mechanism is *recurrence diameter* rdr(M) to utilize minimum number of steps for reaching all reachable states. In the example 6.7, *rdr(M) = 5*, i.e., distance from host *h1* to *h5*, and *rd(M) = 11*, which indicates the number of steps for performing breath first search over the network. I will utilize these threshold baselines to define bounds on the model during empirical evaluation.

Formal Model for Network Verification

Network Invariant refers to the network properties desired for optimal functioning and security of the network, such as virtual network isolation, absence of forwarding loops in the system, and end-to-end packet reachability (absence of black holes), etc. Network verification can be achieved by expressing network invariants based on current topology configuration, traffic management rules, and high-level network requirements. I used temporal logic-based network verification Baier and Katoen (2008) to check if the underlying network meets high-level network requirements. An LTL invariant is evaluated along the linear path. If the invariant state holds for all the paths starting in a given state, I consider the invariant to be true.

The Table 7.2 describes the grammar used for LTL. The LTL model can explain the network invariants, such as global reachability, whitelist policy violations for the underlying network using the queries created using LTL model checking rules

Table 6.1: Formal Semantics of LTL which can be used for Expressing Network Invariants

LTL Rule	**Rule Interpretation**
F p (in the future p)	Condition *p* holds in one of the future time instants.
G p (globally p)	A certain condition *p* holds globally in all future time instants.
p U q (p until q)	Condition *p* holds until a state is reached where condition *q* holds.
X p (next p)	Starting that condition, *p* is true in next state.

(invariants). The invariants serve as *Query Intent* in the presented system. Next, I discuss how service function chain intent and network function intents are represented in the INTPOL system.

6.3 Intent Handling in INTPOL

In this subsection, I discuss how the system handles an intent submitted by the user, as described in Figure 6.4. I describe scenarios - SFC Intent, Network Function Intent such as Firewall (FW), which can be defined individually. The VNFs can serve as part of SFC Intent and Query Intent, allowing users to check end-to-end reachability and policy conflict checking with illustrative examples.

6.3.1 Service Function Chaining Intent

In this example, I consider Service Function Chain (SFC), as shown in Figure 6.5. The example describes the traffic processing between different end-point groups (EPGs) - EPG [1-4]. The network gateway (NAT) in the example also acts as a traffic classifier. There are three separate service chains in this example. The HTTP traffic is classified at the Network Address Translation (NAT) gateway and follows the corresponding service chain path, with Deep-Packet Inspection (DPI), i.e.,

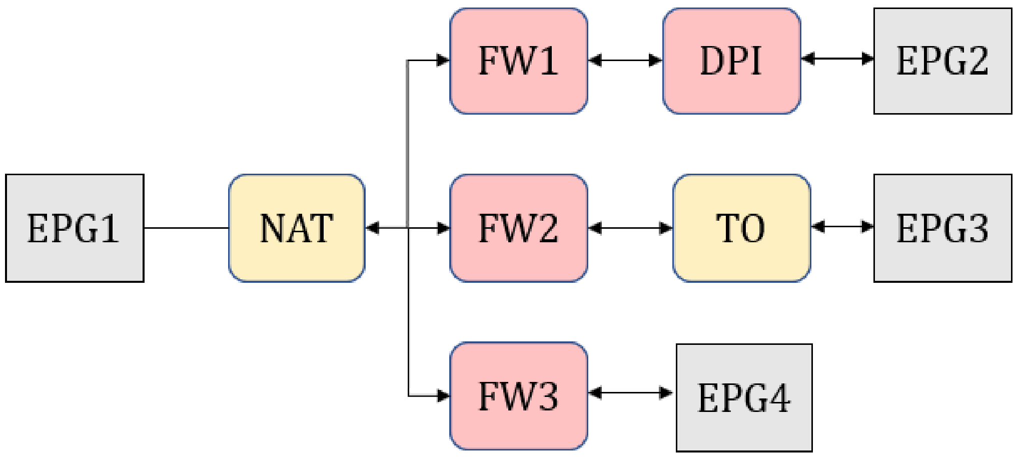


Figure 6.5: A Service Function Chain (SFC) Scenario With Multiple Network Functions (NFs). Each NF can be Described Individually using Network Function Intent.

SFC1: EPG1 → NAT → FW1 → DPI → EGP2. If the traffic is meant for video streaming services it follows an alternate path with the Traffic Optimizer (TO), i.e., SFC2: EPG1 → NAT → FW2 → TO → EGP3. All other traffic follows SFC3: EPG1 → NAT → FW3 → EGP4. The example illustrates the use of INTPOL grammar described in Figure 6.6 to represent the individual service chains. For stronger security, I consider that the network follows a white-listing approach, and the traffic is only allowed between EPG1 as the source and EPG[2-4] as destinations. The traffic between EPG [2-4] is blocked as a part of network policy.

The example in Figure 6.6 illustrates the implementation of complex network service chains. Variables *dst* and *sf* in lines 1-2 are used to define packet destination and service function, respectively. The formal model of packet propagation through a chain of different network functions is described in lines 5-14. For instance, if the packet service function is NAT and destination is EPG2, the packet is forwarded to FW1, DPI, and finally, EPG2 is part of the first service function chain SFC1 - line 6-8. Similarly, lines 9-11 represent the implementation of SFC2, and lines 12, 13 are the implementation of SFC3.

```
MODULE main
VAR dst: {EPG1, EPG2, EPG3, EPG4}
    sf: {NAT, FW1, FW2, FW3, DPI, TO}
ASSIGN
NEXT(sf) := case
    sf = NAT & (dst=EPG2) : FW1;
    sf = FW1 : DPI;
    sf = DPI : EPG2;
    sf = NAT & (dst=EPG3) : FW2;
    sf = FW2 : TO;
    sf = TO : EPG3;
    sf = NAT & (dst=EPG4) : FW3;
    sf = FW3 : EPG4;
    TRUE: NAT;
 esac;
 NEXT(dst) := dst;
 INIT sf = NAT;
```

Figure 6.6: Example Usage of LTL Based Model Checking Framework for Implementing Three Separate Service Function Chains. The NAT Service Function acts as Traffic Classifier and Routes Packets via Corresponding Service Chain Based on Packet Header Match.

6.3.2 Network Function Intent

I consider an individual network function *Firewall (FW)* to check how the rules of network functions can be expressed using the INTPOL framework. Consider two invariants I1, and I2, that can be described using LTL equations (I1), (I2) below:

I1: Traffic sent by host h1 should eventually reach host h5.

$$\forall p \in Packet : G(send(h1, p) \cap any(p)) \rightarrow F(recv(h5, p)) \tag{6.1}$$

I2: any traffic sent by h2 should not reach h4.

$$\forall p \in Packet : G(send(h2, p) \cap any(p)) \rightarrow G(\neg recv(h4, p)) \tag{6.2}$$

The equation (I1) is used for checking reachability property between h1 and h5, and equation (I2) contains the firewall rules between hosts h2 and h4. If there is any network state along the patch h1-h5 which violates the network invariant, it will be produced as a counter-example of the model. This way, the framework can verify higher-level network intents using model checking.

```
1  MODULE main
2     VAR switch: {s1,s2,s3};
3         src: {h1,h2,h3,h4,h5};
4         dst: {h1,h2,h3,h4,h5};
5     ASSIGN
6     NEXT(switch):= case
7          switch = s1 & (dst !=h1 | dst !=h2):s2;
8          switch = s2 & (dst !=h3): {s1,s3};
9          switch = s3 & (dst !=h4 | dst !=h5):s2;
10         TRUE: s1;
11    esac;
12    NEXT(src):= src;
13    NEXT(dst):= dst;
14 INIT switch = s1;
15
16 check_ltlspec_bmc -k 5 "G (src=h1->F(dst=h5))"
17 check_ltlspec_bmc -k 5 "G (src=h2->G(!dst=h4))"
```

Figure 6.7: An Example of Bounded LTL Model That Utilizes Network Topology from SDN Controller to Create a Model Specification. The Last Line Represents Query Intent to Check If Any Packet Starting from (src=h2) Can Eventually Reach (dst=h4).

Consider an example of modeling network intents from Figure 6.3 using a bounded LTL model. Since the intents are representative of Firewall rules, a type of *Policy Intent*, the model considers packet header, based on the grammar described in Figure 6.7. The variables *switch*, *src*, and *dst* in the *VAR* section - lines *2-4* represent the scope of values for switches, source and destination address. In this example, I have used variables to represent the values, but depending on the type of intent, the values can take numeric range, e.g., *src=192.168.1.0/24*, *switch =of:00000001*.

The *ASSIGN* section lines *5-13* checks the next state transition of a network packet. When the packet is located at switch s1, if the packet's destination address is not h1 or h2, i.e., (dst!=h1 or dst!=h2) is forwarded to switch s2. Alternatively, the model can consider the next transition of state *switch* for this packet to be s2. Similarly, based on the packet header match conditions, the state transition of the packet is determined in the program. The block *INIT* - line *14* is used to specify the starting state of the packet. Next, I will illustrate how *Query Intents* can be used within the model checking to identify cases of policy conflicts.

6.3.3 Query Intent: Policy Conflict Checking

In this sub-section I will illustrate the problem that can exist because of mismatch between high-level network security and orchestration requirements (intents). In the example Figure 6.3, high-level network intents for hosts, $h2 \rightarrow h3$, $h3 \rightarrow h4$ will lead to creation of corresponding rules at switches $s1, s2, s3$. The combination of these rules can cause violation of security requirements. It is possible that, $h2 \rightarrow h3 \cup h3 \rightarrow h4 = h2 \rightarrow h4$.

The problem of identifying such network anomalies can become quite involved in a network consisting of thousands of sub-networks, hosts, and switches. Moreover, the example above describes a case of a simple access control list (ACL) intent expressed at the application plane. The advent of network function virtualization (NFV) Han *et al.* (2015) has allowed the creation of network functions such as load balancer, intrusion detection system (IDS), and deep-packet inspection (DPI) as part of a programmable network. Thus, identifying policy inconsistencies across different layers of the network protocol stack and network paths becomes a challenging task. Existing rule-conflict detection mechanisms Pisharody *et al.* (2017); Hu *et al.* (2014) focus exclusively on OpenFlow rule conflicts at the data-plane level. Conflict detection and resolution mechanism at the switch level can introduce unnecessary read-write latency issues and interrupt the network's normal functioning. Symbolic model checking (SMC) can express the network properties at a higher level of abstraction. The model checkers, such as NuSMV Cimatti *et al.* (1999), allow the granular representation of network security and end-to-end connectivity properties. Lines *16-17* from Figure 6.7 check if the IP address from source *h1* eventually reaches destination *h5*, and packet from source *h2* never reaches destination *h4*.

6.4 INTPOL Implementation

I use the administrative panel's intent to create a formal model of network infrastructure and policies. The network policies are checked for the type of intent. If the intent is a *Network Intent* (network function rule, or service function chain requirement), the intent is added to the existing state transition system defined for the formal model. Additionally, the SDN controller is queried for extracting information on network topologies, such as connectivity between hosts and switches. Using the transition system and the topology information, the formal model is formulated. Alternatively, if the type of intent is *Query Intent*, it is used for creating the LTL queries for checking the network invariants. For instance, if the query asks about the possible path between two network hosts, the model makes a query to check if a packet starting from the source address finally reaches the destination, as explained in the previous section. As part of the modeling framework, I also used the network topology information to place the formal model's bounds. The value of the bound depends on the network diameter for the presented model. If the invariant results in a counter-example (violation of network policy), the network admin is informed about the violation. The admin can choose to accept the violation or provide an alternative approach that does not violate the network policy depending upon the policy's criti-

Algorithm 3 Model Generation and Verification

```
procedure NETWORK-INVARIANT (r)
    add-ltl-spec (r)
    K $\leftarrow$ network diameter
    add-model-bounds (K)
    if network-invariant-violation (r) then
        return False
    else
        return True
    end if
end procedure
procedure SERVICE-FUNCTION-CHAIN (SF-List)
    for i $\in$ range (SF-List[1,n]) do
        Extract-Path ($nf_i$, $nf_{i-1}$)
        add-src ($nf_i$), add-dst ($nf_{i-1}$)
        add-state-transition ($nf_i$, $nf_{i-1}$)
    end for
end procedure
procedure NETWORK-FUNCTION (r)
    header, action $\leftarrow$ {match, action} $\in$ r
    add-src (header.src), add-dst (header.dst)
    add-state-transition (header, action)
end procedure
procedure REQUIREMENT-PARSER (R)
    for r in R do
        if r.type $\in$ Service-Function-Chain then
            SF-List $\leftarrow$ r.extract()
            call SFC-Create (SF-List)
        else if r.type $\in$ Network-Function then
            call Network-Function (r)
        else if r.type $\in$ Query-Intent then
        else if call Network-Invariant (r) == False then
            send ($r_c$ $\in$ R to admin)                    ▷ Rules in violation
        end if
    end for
    call RULE-CONFLICT-CHECKING (R)
end procedure
```

cality. Suppose the intent module does not identify any network property violations. In that case, the non-conflict policies are passed to the SDN controller, which calls the *Intent Processing Module*, where the intents are analyzed for second-level policy, i.e., flow rule conflicts.

Algorithm 3 describes the processing of intents at the application plane and generation of a formal model. The rules are sent to `REQUIREMENT-PARSER (R)` function, which parses the submitted network intents. If the type of intent is *Service Function Chain* - lines 25-27, the intent is passed to `SERVICE-FUNCTION-CHAIN (SF-List)` procedure, where source and destination network functions are identified - lines 13-14, and the model is updated with state-transition corresponding to the path between network functions. Similarly, if the type of intent is individual *Network Function*, the call to `NETWORK-FUNCTION (r)` checks the matching criteria for a rule (firewall rule, IDS rule) and corresponding action - lines 19-20. The formal model is updated with the values of the header match and related action - line 21. If the type of intent is *Query Intent*, the call to `NETWORK INVARIANT (r)` procedure invokes the processing of query submitted by the network admin, e.g., checking end-to-end packet reachability, application plane conflict check - lines 4-10. If the network invariant is satisfied, the rule is kept in the list of rules that will be further analyzed by a call to `RULE-CONFLICT-CHECKING (R)` Algorithm 5. If the invariant fails, the admin is notified - line 32, and affected rules (formal model states presenting counter-examples) are removed from the set of non-conflicting rules.

6.4.1 Intent Processing Module

SDN controller's intent framework allows the users to specify their networking and security policies. SDN controller follows a state machine for handling such policies, i.e., intents. When a new intent request is submitted, the intent processing state diagram processes it and computes a flow rule installed in an OpenFlow switches. Consider an intent that allows communication from host h1 to host h5. Intents

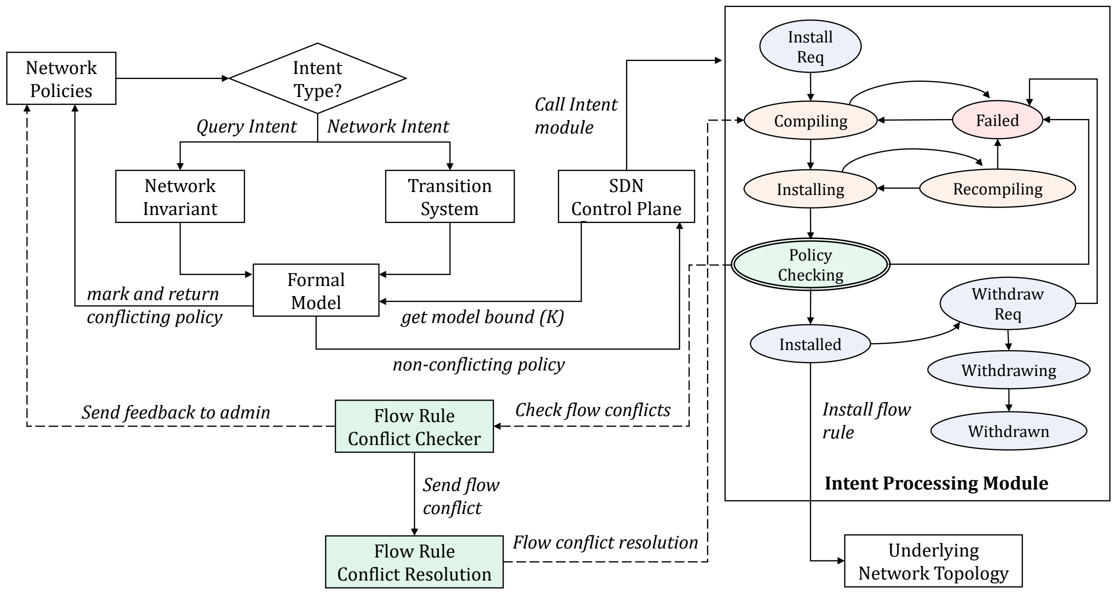


Figure 6.8: INTPOL Data Flow Diagram Describing Multi-level Network Policy Processing. The Formal Model Analyzes the Policies at Application Plane, and Policy Conflict Checker Checks the Conflicting Flow Rules at Control Plane. The Non-conflicting Policies Are Inserted into Switches at Data Plane Level Using Openflow APIs.

are sent asynchronously to *Compiling* stage. The *Compiling* stage performs various checks on the incoming intent, and if the compilation is successful, it returns a list of installable intents. The intents need to be validated for its feasibility, connectivity with regards to the given network topology, network criteria, resource availability, etc. The compilation process computes a primary shortest path and a backup path between the two given hosts. Here the shorted path from host h1 to host h5 would be h1-s1-s2-s3-h5, given all links have the same weights. Suppose a node specified in user intent cannot be reached to the other node specified due to lack of connectivity or link failure or resource unavailability, or any other reason. In that case, that intent fails the compilation. At this stage, it is difficult to say if this failure is due to some temporary network failure or some temporary event. Hence that intent is kept in a

Compiling state. If the compilation fails, the state of the intent is assigned to a *Failed* state. In the event of network topology change, or link re-association, failed intents are considered again for compilation. In the event of topology change, if the network connectivity is regained, then compilation may succeed. The installable intents sent from *Compiling* state to *Installing* state. If the installation fails for any reason, then it goes to *Recompiling* state. Once the intents pass, the *Installing* state is also analyzed for Policy Conflict, as described in the next subsection. In the event of no conflicting policies, the state transitions to *Installed Intent.* These intents are installed as flow rules in the underlying OpenFlow network.

6.5 INTPOL Performance Evaluation

This section describes the evaluation of the INTPOL framework using different application scenarios and network setups. First, I present a case study to show how INTPOL performs in a service function chaining (SFC) scenario, comprising inter-domain and intra-domain policies. In particular, I used a hybrid network consisting of both SDN-based networking and traditional networking. Second, I applied the INTPOL approach to a large network scenario by utilizing the Stanford topology Kazemian *et al.* (2012), which has many hosts, and OpenFlow switches running ACL rules. I used policy conflict checking as an invariant to assess the INTPOL framework's scalability on an extensive scale network with an increase in the number of intents.

I used the INTPOL framework to show that a multi-domain hybrid network scenario, which consists of both SDN-based networking and traditional networking, as shown in Figure 6.9 a particular case of SFC. Some components are software-defined, e.g., Switch (s1-5) and (s11) are managed by ONOS-01, whereas switch (s6-10) is operated by ONOS-02. Next, I expand the INTPOL framework to provide end-to-end

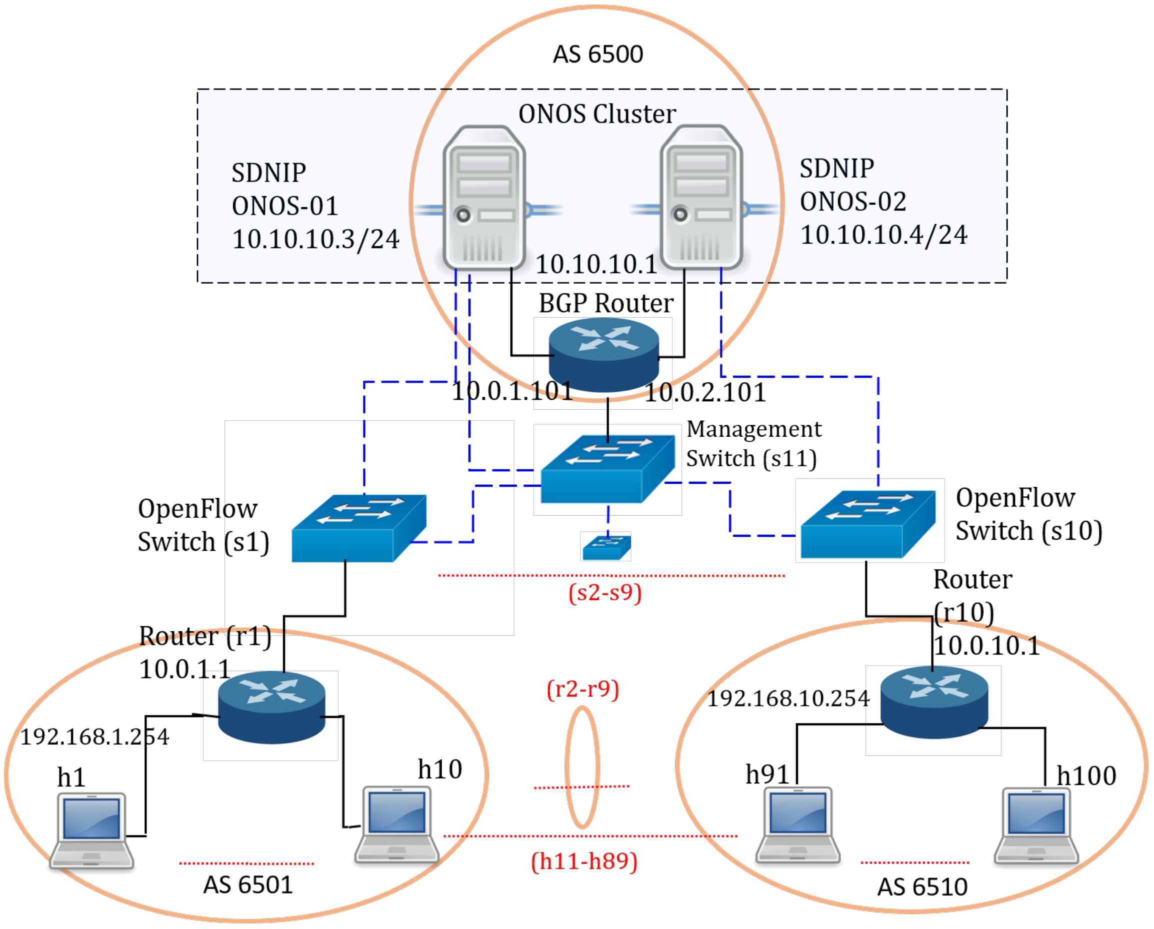


Figure 6.9: A Hybrid Network Scenario with Network Components Based on Traditional Networking (BGP Routing) and Openflow Network (blue dashed lines) the Different Domains Are Represented in by the Orange Circles.

policy verification in a hybrid network scenario with Border Gateway Protocol (BGP) for communication across geographically distributed networks and software-defined networking components (OpenFlow network).

Consider a network consisting of distributed domains, as shown in Figure 6.9. BGP based routing technology is employed to enable communication between the network managed by ONOS-01 and ONOS-02. Moreover, the end hosts (h1-h100) in each system are connected to the SDN networking using routers r1-r10. Hence, the Openflow network can only see the packets coming from routers. The routers apply

NAT functionality to masquerades the local IP address into the public domain IP to enable BGP communication.

In this example, I assume that each router manages one domain, e.g., AS6501 contains hosts h1-10. Internally each router has a corresponding OpenFlow switch providing layer-2 connectivity. I tested the INTPOL performance regarding the proposed BMC for the example in Figure 6.9. The model has two checking levels, at the intra-domain (within each AS) and in the inter-domain (across AS).

Inter-Domain Communication

The overall network described in Figure 6.9 can be represented in the INTPOL framework using two-level of abstraction, i.e., SFC intents and Network Function (NF) intent. I consider the inter-domain network as a special case of Service Function Chain (SFC), where routers (r1-10) can be interpreted as NAT providers. The switches (s1-10) perform packet switching, and basic traffic filtering reduces the overall overhead of checking network policies. The end-hosts can be represented as an abstracted group, i.e., $\{h1, .., h10\} \in AS6501$, {ONOS-01, ONOS-02} $\in AS6500$, and so forth.

The intra-domain level is the autonomous system (AS) level, which has the SDN controllers and the edge routers *r1-r10*. I consider this a network function NAT within each AS. The administrator requirements are specified such that the end-hosts (*h1-h100*) can reach each other, or some of them should never be communicating. Note that there are two levels of policy checking at the SDN controller level and the individual traditional routing domains in this scenario. On one hand, the SDN level domain checks for the policies across the AS and handles the OpenFlow rules between ASes. On the other hand, each AS (*r1-r10*) is checking the inter-domain invariant. This allows the admin to ensure end-to-end packet reachability within a domain and between the domains. So the evaluation goal of this scenario is to measure the performance of network invariant checking.

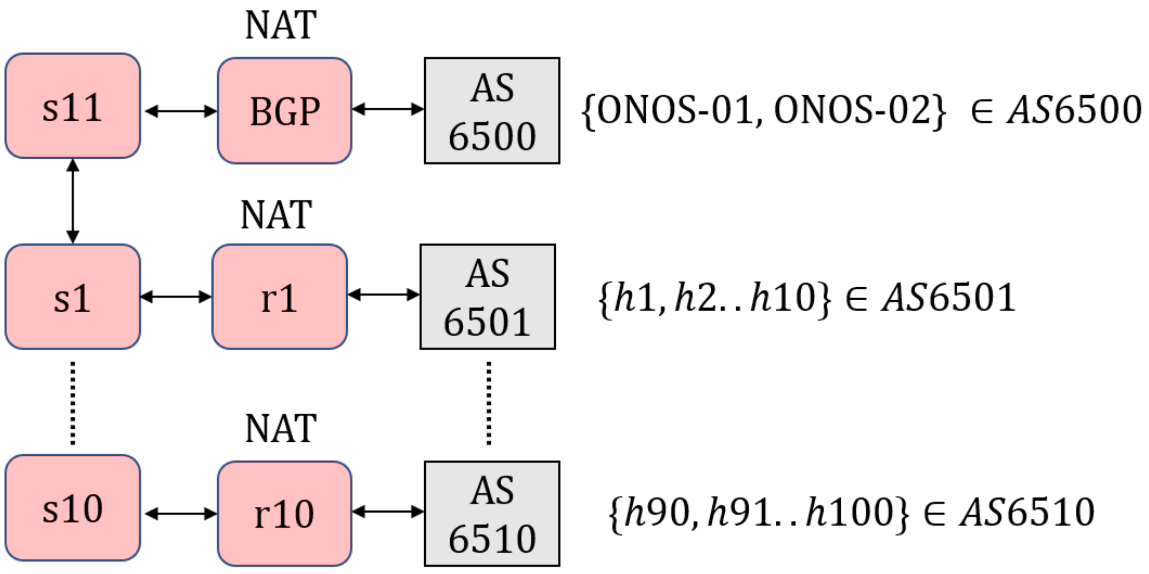


Figure 6.10: An Encapsulated Representation of Hybrid Network Scenario as a Special Case of Service Function Chaining (SFC).

I performed an experimental evaluation of using LTL full-scale model checking, and LTL bounded model checking, i.e., LTL-BMC in a multi-domain scenario described in Figure 6.10. It can be observed that performing network invariant checking by abstracting the packet processing across domains to a special SFC case allows inter-domain packet switching and routing policies to be analyzed in a fast and efficient manner. Each network domain comprised ten hosts. I incremented the number of domains from 2-10 to observe the time required for LTL and LTL-BMC model construction and packet reachability property checking, as shown in Figure 6.11.

The value of network bound K was selected based on the diameter of the current network, i.e., if the maximum path length between two hosts is 10, I utilized K=*10*. The invariant checking using BMC for #domains=2 finished in 0.048(s), whereas LTL required 0.108(s). I observed a similar trend as I increased the number of domains to 10. For #domains=10, BMC required 0.07(s), whereas the LTL finished in 0.016(s).

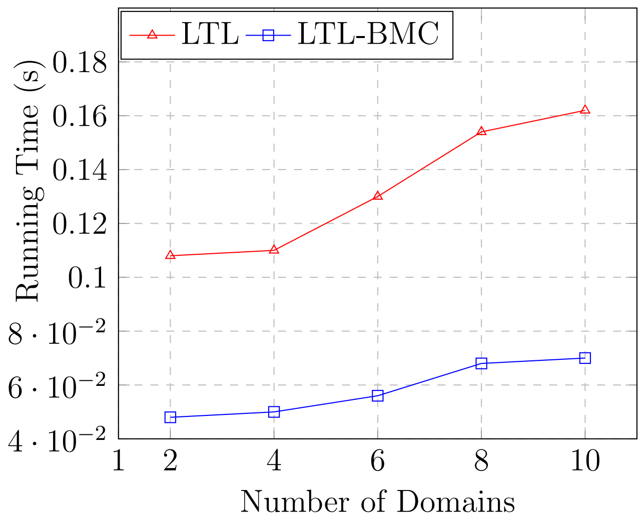


Figure 6.11: The Experimental Analysis of INTPOL in Inter-domain Hybrid Network. The LTL-BMC Scales Well as the Number of Domains Increase in the Network.

Overall the BMC was >2x faster in terms of invariant checking compared to the LTL framework. This shows that BMC scales well in a multi-domain scenario, and the INTPOL framework is generalizable to incorporate multiple types of network setups.

Intra-Network Communication

The hosts h1-10 utilize the traditional routing for communicating with each other, which makes the SDN network oblivious to the traffic managed by routers r1. Since the network in consideration is a geographically distributed network, e.g., 192.168.1.0/24 is managed locally by r1-5. Each router is connected to a switch, e.g., r1-s1. The switches s1-10 are connected to switch s11, which is connected to the *bgp* router (s11-bgp). The BGP router is connected to ONOS-01, and similarly, 192.168.10.0/24 is managed by r6-10, which is connected to ONOS-02. I used the SDNIP application Lin *et al.* (2013) to allow ONOS to run and communicate with BGP router through iBGP protocol and communicate with routers *r1-r10* using eBGP protocol. I refer the reader to SDNIP details in Lin *et al.* (2013); Jonathan Hart (2020) for details

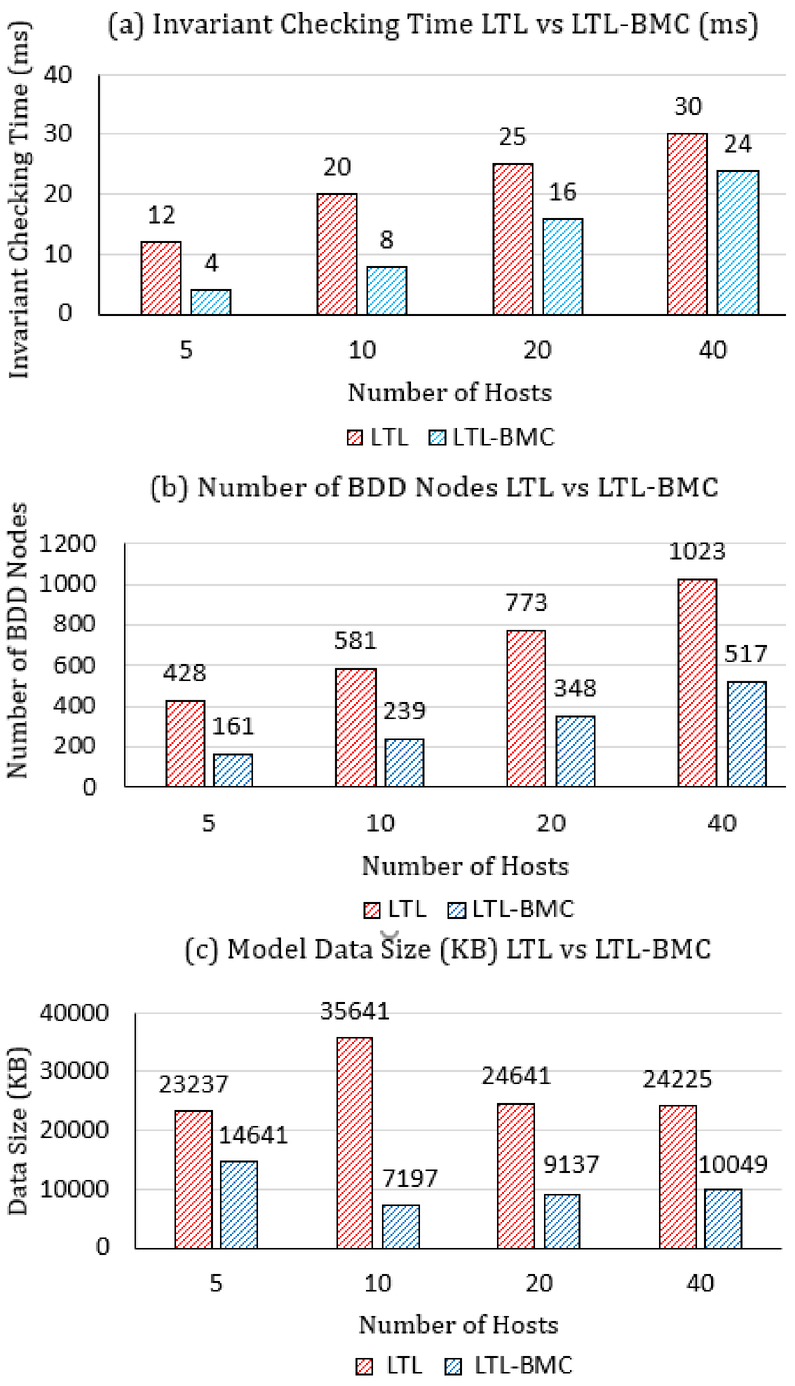


Figure 6.12: INTPOL Model Checking Framework Evaluation in a Single Domain Environment. As I Increased Number of Hosts in Domain As6501, the Binary Decision Diagram (BDD) Nodes, Data Size of the Model, and Time of Model Checking Is Reduced for Bounded Model Checking (LTL-BMC) Compared to Full Scale LTL.

about connecting inter-domains with the ONOS controller. Next, I checked the scalability of INTPOL by increasing the number of hosts within one domain. I consider switching, and access control functions within AS6501 managed as part of the routing domain of router r1.
To evaluate the proposed INTPOL framework's scalability within a single domain AS6501, I experimented with measuring the BMC model's performance compared to the LTL model. Specifically, using NuSMV I evaluated using the NuSMV program the number of Binary Decision Diagram (BDD) nodes, the data size, and the time of both the LTL model and the BMC model. Figure 6.12 shows the comparison between the LTL and the BMC model in four different scenarios, where the number of hosts in the network initially is set to 5 and gradually increases up to 40 hosts. The time required for checking invariant (end-to-end reachability) in the case of LTL is 12 (ms) for five hosts, whereas, for LTL-BMC, it is 4(ms). The overall time of LTL-BMC is less compared to LTL as the number of hosts is increased in the model- Figure 6.12(a). This is because the network width is used as a model bound for LTL-BMC; the state space for invariant checking is smaller. It can also be noticed that the number of BDD nodes in the BMC model is much less than the LTL model - Figure 6.12(b). Moreover, the data size for the LTL model is more extensive in all cases. The BMC model reduced the data size with a reduction of 37% in the five hosts case and 60% in the 40 hosts case - Figure 6.12(c). The main reason for the reduction is that I simulated and accounted for all the states generated by the model in the LTL model. In contrast, in the BMC model, I can set K=10, which means the model is simulated with provided bound on states or steps to find the counterexample. This helps the model scale well as the network hosts increase within a single domain.

6.5.1 Policy Conflict Checking for Large Network Scenario

To analyze the scalability of INTPOL on an extensive network, I used the Stanford Topology Kazemian *et al.* (2012) for the experimental analysis. The topology consists of 14 Operational Zones (OZ) routers, ten switches, and two backbone routers connected to OZ's via switches. There are 1500 ACL rules and 757k forwarding entries in the simulated network Kazemian *et al.* (2012). For this case-study, I wanted to evaluate the performance of INTPOL in terms of policy translation scalability and how would INTPOL behave when the network in question is large. Moreover, I wanted to show how the proposed BMC module will behave in a large system compared to the traditional LTL model for checking network invariant.

Table 6.2: INTPOL Model Checking Framework Applied to Stanford Topology Kazemian *et al.* (2012). The Overhead of Generating Model using LTL-BMC is Lower Compared to LTL Framework.

# Intents	**Intent Gen. Time**	**LTL Nodes**	**LTL-BMC Nodes**
20	3.42	9768	3170
40	4.33	13936	3354
60	5.91	18713	3532
80	8.525	19230	3492
100	11.29	13936	3849

I performed two-level of policy conflict checking to establish the benefit of checking conflicts at the application plane. I utilized Stanford Topology for this purpose Kazemian *et al.* (2012). The first level of policy conflict checking is at the application plane, where the policies specified are converted into a formal model by checking user requirements and network structure. The system policies are converted into intents based on Algorithm 3. The network intents are matched for conflict, reachability, end-to-end packet flow using network invariant. In the case of Stanford topology, I selected the hosts and checked conflict-free policies between them. Each end-to-end packet reachability requirement between two randomly chosen hosts is inserted into the formal model as a network intent. Rules for checking conflict between user policies are added as a network invariant. For instance, when the number of intents (# Intents) is 20, I created host to host reachability intents between randomly selected pairs of hosts in the network.

It can be seen from Figure 6.13, runtime increases with the number of intents, i.e., 3.42(s) for 20 intents and 11.29(s) for 100 intents. I analyzed the case of conflict checking at the application plane using a formal LTL model and an LTL model with bounds placed on the model (LTL-BMC). The number of nodes expanded in the case binary decision diagram (BDD) created as a part of the LTL model is ∼9-14K table 6.2 (column 3) when the number of intents increases from 20 to 100. The nodes' size expanded for a bounded model table 6.2 (column 4) is significantly less ∼ 3-4K than the LTL model. This result is expected because I placed a bound on the diameter of node expansion by using the bounded model. The overhead of conflict checking in the unbounded model is 0.18(s) for 20 nodes, and it increases to value 0.225(s) for 100 intents. In using BMC, the conflict checking routing takes a smaller amount of time, i.e., 0.1(s) for 20 intents and 0.20(s) for 100 nodes.

Scalability of Conflict Checking at Application Plane

It can be observed from Table 6.2 that the overhead associated with the identification of conflicts at the application plane using the bounded model is quite limited. I will now discuss the overhead associated with conflict checking at the data plane. I assumed that the application plane (no formal model) is present in the system. The intents are inserted into OpenFlow switches using the intent processing module described in Figure 6.8. It can be observed that the number of flow rules (#Flow Rules) generated for 20 intents is 413. This result is because, for realizing an intent, the SDN controller needs to add flows along the path between two hosts. For example in case of example network presented in Figure 6.3 if I add a host intent `add-host-intent h1 h5`, I need to insert four flow rules, rule 1 (flow between h1 and s1), rule 2 (flow between s1-s2), rule 3 (flow between s2-s3), and rule4 (flow between s3-h5). Thus in the case of hosts selected for a scalable topology such as described in this experiment, there can be multiple flow rules for each intent—these flow rules in-turn conflict with each other.

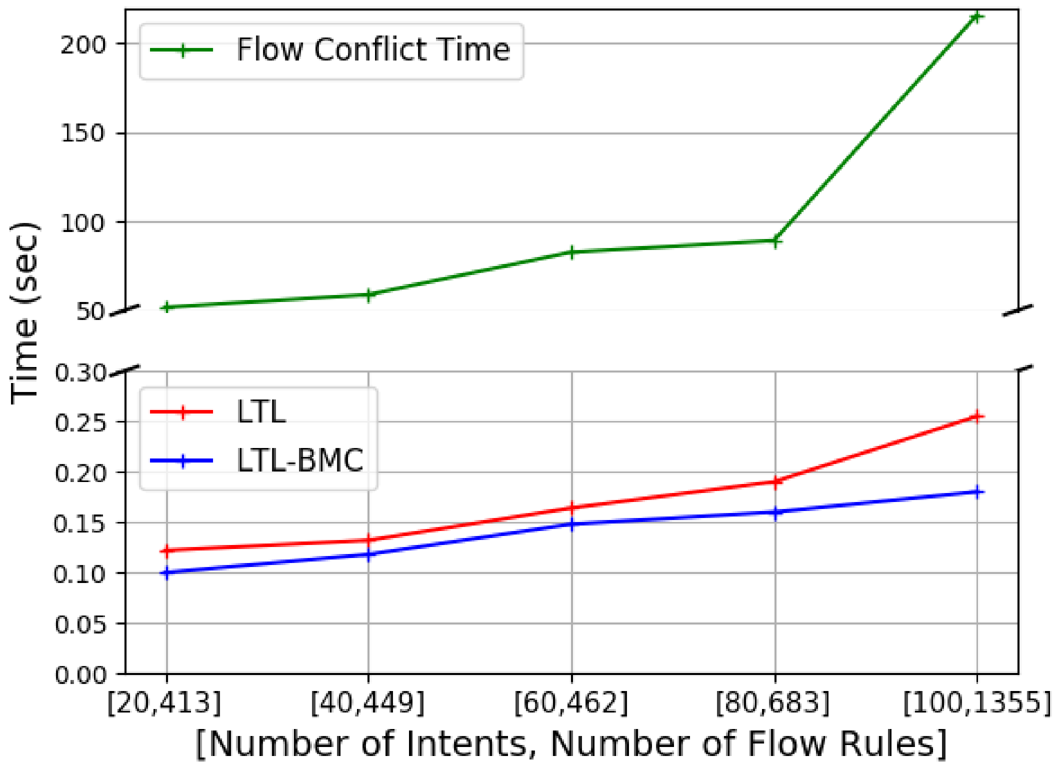


Figure 6.13: A Comparison Between the Flow Rule Conflict Detection Time, and the LTL and LTL-BMC Time. The Data-plane Level Conflict Detection Time is Significantly Larger in Absence of INTPOL Framework.

Furthermore, it can be observed from the Figure 6.13 that conflict checking time is 51.68(s) for 413 flow rules. This result is about 510x higher than the conflict checking overhead at the application plane using BMC. As I increased the number of intents from 20-100, the number of flow rules generated is increased to 1355. The time required for checking conflict among flow rules is also increased to 216.01(s). This measurement result is significantly higher than the time needed to check conflicts at the application plane 0.260(s). Thus, if a model can identify and remove the intents' conflicts at the application plane using the bounded formal model, the number of flow rules generated will be significantly lesser. The overall overhead associated with conflict detection and resolution at the application plane and data plane combined will be significantly reduced.

6.6 Policy Configuration and Conflict Analysis Research

6.6.1 Intent based Policy Configuration

I analyzed the research works involving intent-based policy expression and management. JANUS Abhashkumar *et al.* (2017) builds upon the policy graph abstraction (PGA) framework proposed by Prakash *et al.* (2015), for representing dynamic temporal policies and QoS policies in an intent-based language. Janus also aims to maximize the number of configured policies by utilizing heuristic algorithms. PGA Prakash *et al.* (2015) analyzes the Access Control List (ACL) rules and module them in a graph structure to find conflicts between ACL policies. The graph structure input is the possible communication between the network endpoints and the required service function chain for every communication. Yet, PGA requires users to manually verify the detected conflict's correctness to ensure reachability requirement in the network is satisfied. PGA also does not support stateful capabilities such as an intent that specify stateful firewall service rule.

Han *et al.* (2016) presented a framework for providing an interface to add intents by the user and translate it into network policy. The authors do not consider the scenario of multiple controllers running in the environment, nor do they analyze the added intents' conflicts. Jacobs *et al.* (2018) discussed how AI could be utilized to allow networks to be more intelligent. They showcased how intent-based networking (IBN) can translate high-level policies without the overhead of translating them into network flows. Most existing works lack a framework that can interpret diverse network requirements and perform end-to-end network property verification using a unified language.

INTPOL provides a scalable formal language that allows network operators to specify network intents at the application plane. This abstract the network operator from the details of policy configuration for each network controller. The policy translation and mapping provided by INTPOL takes care of translating network intents into controller specific policies.

6.6.2 Network Policy Conflict Checking

There are several existing solutions on network policy checking and invariant verification, such as Khurshid *et al.* (2013); Beckett *et al.* (2017); Pedrosa *et al.* (2018); Tian *et al.* (2019); Jacobs *et al.* (2018); Arashloo *et al.* (2016). One of the earlier work for network invariant verification is Veriflow Khurshid *et al.* (2013). Veriflow aims to check the network invariant in real-time with the change in the network state. Thus, Veriflow only checks the packet reachability (as one type of invariant) between two machines after the flow has been deployed, according to Beckett *et al.* (2017).

To ensure the network invariant is checked pre-deployment, Beckett *et al.* (2017) have presented an approach for converting the network configuration files to *logical formulae*, which is based on the different communication between the routing protocols. This formula is then described according to the specified constraints into a networking invariant, which will eventually ensure the network performs it's intended purpose in terms of reachability, loop and black-holes free, etc.

Jinjing Tian *et al.* (2019) is a system that aids Alibaba's network operators to automatically and correctly updating ACL configurations in Alibaba's global Wireless Area Network (WAN). The system automatically synthesizes ACL update plans that satisfy the required intent. However, JinJing only considered the ACL rules as one type of network invariant; they did not consider the possibility of invariant conflicts due to the natural differences in multiple controllers in the system.

NetSMC Yuan *et al.* (2020) uses an existential first-order logic and query containment to provide stateful network verification. The framework uses image pre-computation to help provide a scalable formal model. The formal model for network invariant verification suffers from the number of verification states' scalability challenges. This work utilized a bounded model checking approach to address the scalability limitation inherent in existing research works. As a result, INTPOL scales linearly in terms of the number of model states.

To address multi-domain SDN policy management, a recent study by Varadharajan *et al.* (2018) proposed a new application that handles the policies between different SDN domains. The paper uses a policy handle and policy token. The handle is used to show the visited autonomous system (AS) by the flow and the packets. The evaluation of the paper offers an efficient network communication in terms of throughput and delays. One key feature of that work is missing, which is to verify the policies effectively and ensure end-to-end reachability without conflict between the policies as I did in this research. Furthermore, the work Varadharajan *et al.* (2018) does not verify the policies against network invariant, which are desired for optimal functioning and security of the system.

6.7 Summary and Discussion

In this chapter, I discussed a multi-level network policy checking framework for SDN networks. I introduced a new language, INTPOL, for unified interpretation of network-wide security policies and mission requirements in a multi-domain cloud environment. INTPOL framework utilized bounded model checking to limit checking conflicts at the application plane, reducing the number of generated flow rules at the data plane. The presented solution can reduce the overhead of network policy conflict checking significantly. I utilized case studies to show that INTPOL is generalized enough to handle scenarios such as Service Function Chaining (SFC), multiple network functions (Firewall, IDS, DPI), and hybrid networks involving traditional BGP routing and OpenFlow components. The framework scales well on a large enterprise-grade network, as demonstrated by experiments performed on Stanford topology.

Chapter 7

FLOW RULE CONFLICT DETECTION AND RESOLUTION

In this chapter, I elaborate on an alternate model for detecting and resolving conflicts between security policies in an SDN-managed environment, known as the Object-Oriented Policy Conflict (OOPC) framework. OOPC analyzes the rule dependency relationships between the rules of heterogeneous virtual network functions (VNFs) and creates a VNF-Graph. The rules are analyzed using object-oriented dependencies between the address space and actions of VNF rules. OOPC utilizes compact VNF-Graph, which leads to a reduction in search complexity when analyzing new security policies. The security policy using policy graph composition in OOPC achieves 37% lower latency than current works in the field of end-to-end policy composition. The proposed solution performs 20% faster security policy conflict detection on a cloud network with 60k OpenFlow rules than prior frameworks that serve a similar purpose.

7.1 Security Policy Formalism

7.1.1 Network Traffic and Packet Classification

Definition 7.1.1 ***Network Traffic:*** *The traffic in a given network is composed of multiple packets, i.e.,* $\Gamma = \{p_1, ..., p_k\}$. *Each packet has some network fields, i.e.,* $p_i = \{(n_1, v_1), (n_2, v_2), .., (n_{n_p}, v_{n_p})\}$. *The field* n_i, *here refers to packet header field, e.g., source IP address, destination port, and* v_i *refers to value assigned to that network field, i.e.,* $p(n_i) = v_i$.

Definition 7.1.2 ***Packet Classification:*** *The incoming traffic packets for a network, Σ_i, can be classified into subset of rules R_m from the ruleset of the entire network R, i.e., $R_m \subseteq R$, where $R_m = \{\forall_{i=1}^{m} r_i\}$. Furthermore, each rule r_i, can be decomposed based on packet match condition, and actions $r_i = \{m_i, a_i\}$.*

The match field can be further classified into individual headers that are part of the packet. The packet match m_i consists of physical port of incoming traffic δ_i, source and destination hardware address, i.e., α_{si}, α_{di}, source and destination IP address, β_{si}, β_{di}, source and destination port addresses, γ_{si}, γ_{di}, protocol δ_i, priority value ζ_i for a given virtual network function. For instance, a stateless firewall (iptables) allows the assignment of rule priority.

The Virtual Network Functions (VNFs), acting on network traffic $\forall VNF_i^N$, consists of rules from ruleset R, i.e., $VNF_i.r \subseteq R$, where r represents rules of VNF_i, and $VNF_j.r \subseteq R, i \neq j$. Next, I define notation to relate the network traffic with the rules of VNF (entire ruleset R). In particular, I use an equivalence class relationship $\rho = \{=, \neq, \subset, \subseteq, \leq, ..\}$. The use of equivalence relationship on network traffic allows the classification of network traffic, i.e., $\{(\Gamma, \rho, R) \rightarrow \Gamma'\}$.

7.1.2 Policy Graph Composition

In this research, I consider the end-to-end composition of policies in a chain of VNFs. Let's consider there are virtual network functions $\{VNF_1, ..., VNF_N\}$. Each rule from a VNF has a match and an action tuple, i.e., $r_i = \{m_i, a_i\}$. The match refers to the matching header for the network traffic. The policy graph can be formally defined as follows:

Definition 7.1.3 ***Policy Graph*** *can be defined as* $G = \{V, E\}$*, where vertices* V *refers to the rule matching network traffic on encountering the rule, i.e., rule* r *from the list of VNFs acts on network traffic* Γ*, and transforms it into* Γ'*. The transformation relation can be formally represented using notation* $\{(\Gamma, \rho, R = r) \rightarrow \Gamma'\}$*. The edge* $e \in E$ *is present between two vertices. If one or more rules act on the network traffic and either transformed the packet header or forwarded network traffic to another VNF, i.e., e=*$(r1, r2) \iff r1 \subseteq r2 | r2 \subseteq r1$*, an edge is added between those vertices. For instance, consider the traffic* Γ *between hosts 1.1.1.1 and 1.1.2.1 in the policy conflict analysis example below. The packet will undergo transformation based on the rules associated with VNFs present on the path between these two hosts. The load balancer* $r4$ *changes the source IP address to* $1.1.3.1, 2$*, and IDS based on rule* $r5$ *can perform traffic mirroring using action* INSPECT*, as can be seen in the example rules provided in the next subsection.*

7.1.3 Policy Conflict Analysis

The security policy management framework should provide some essential features (a) rule order independence, i.e., rules can be defined in any order without introducing conflicts (b) modular and extensible - rules from different network functions can be reused to prevent the creation of redundant rules, and allow scaling on a large network (c) automated conflict detection and possible resolution.

The application layer in an SDN-managed cloud network consists of APIs to express *security policies* and *mission requirements*. The application layer policies, specified using additional security and traffic optimization modules like stateless firewall, load-balancer, and IDS, can conflict with each other. The header space of the application layer policies may have partial or full overlap, whereas the actions may not be the same for different security policies.

```
# Stateless L3 Firewall Rules
ID    L3src       L3dst       L4src L4dst ACTION
r1  1.1.1.0/24  1.1.2.0/24    *     *   ALLOW
r2  1.1.1.0/24  1.1.2.0/28    *     *   DENY
r3  1.1.3.0/24  1.1.2.0/24    *     *   DENY

# Load Balancer Rules
ID     L3src      L3dst         ACTION
r4  1.1.1.0/24  1.1.2.0/24  SetIP(1.1.3.1,2)

# IDS Rules
ID    L3src       L3dst       L4src L4dst ACTION
r5  1.1.1.0/24  1.1.2.0/24    *     *   Inspect

# Security Policy
  Order: Load Balancer -> L3Firewall -> IDS

# Mission Requirement
# Bidirectional traffic between 1.1.1.0/24
# and 1.1.3.0/24 should be allowed.
ID    L3src       L3dst       L4src L4dst ACTION
r6  1.1.1.0/24  1.1.3.0/24    *     *   ALLOW
r7  1.1.3.0/24  1.1.1.0/24    *     *   ALLOW
```

If I examine the security rules present in the code snippet above, the policies for L3 firewall, *r1, r2* conflict with each other, i.e., $r2 \subseteq r1$, $r2(ACTION) \neq r1(ACTION)$.

Similarly, the rules for the load-balancer *r4*, and IDS *r5* overlap with header space of rules *r1, r2*. The security policy specifies the order of application of network functions. According to the security policy above, the load balancer must be applied before the L3 firewall and IDS.

The rule *r3* prevents traffic between Src IP *1.1.3.0/24* and *1.1.2.0/24*. Whereas, if rule *r4* is applied before rule *r3* as per security policy, the source IP of the traffic from *1.1.1.0/24* will be modified to *1.1.3.1* or *1.1.3.2* as per *r4*. The firewall blocks this traffic from the modified source IP address as per rule *r3*. The mission requirements, however, specify that the traffic between *1.1.1.0/24* and *1.1.3.0/24* should be allowed. The change introduced by ordering constraints as per the security policy violates the mission requirement.

To preserve the end-to-end application policies, I identify the object-oriented relations between different security and traffic optimization policies and resolve them before implementing the corresponding OpenFlow rules at the control plane level. Next, I describe how to analyze the OpenFlow rules for possible conflicts.

Definition 7.1.4 ***Conflict Detection:*** *Eppstein and Muthukrishnan (2001) seeks to find the rules* $r_i, r_j \in R_m$*, that are conflicting with each other, i.e.,* $(\rho_i = \rho_j) \wedge (h_i \cap h_j \neq \emptyset) \wedge (a_i \neq a_j)$*. I will use the variables described earlier to illustrate and example of policy conflicts.*

Table 7.1: Motivating Scenario: Conflict Detection

Flow-ID	Src-IP	Dst-IP	Src-Port	Dst-Port	**Action**
1	1	[0-10]	2	[0-100]	$\{(drop)\}$
2	1	[0-100]	[2,4]	[0-100]	{(set srcip 5)}
3	[5,8]	[0-100]	[0,8]	[0-100]	{(set srcip 2), (fwd)}
4	2	[0-100]	[2,4]	[0-100]	$\{(fwd)\}$

Consider, Table 7.1, I used simple numeric values for source and destination addresses for concise representation and consider other OpenFlow fields, e.g., layer 2 source and destination addresses to be wildcarded. There are two types of violations here.

Coverage Violation: The rules 1 and 2 have overlapping header space. The Src-IP of the rules is the same. The destination IP of rule 2 is a superset of rule 1. The actions of both rules are different. This is a case of conflict amongst flow rules 1 and 2. Suppose the traffic source is [1-10] and destination is '2'. In that case, the traffic may be dropped (assuming the whitelisting policy of OpenFlow framework), even though the network administrator intends to send traffic between these sources and a destination address range(s).

Transitive Violations: According to the rule with Flow-ID '1' present in the table, every packet from Src-IP 1 towards Dst-IP 2 must be dropped. However, rule 2 in the table allows modification of source IP to value 5 and rule 3 sets the source IP of any field between [5-8] to the value 2, and rule 4 forwards traffic to Dst-IP [2,4]. Thus, using rules 2,3, and 4 the traffic between Src-IP 1 and Dst-IP 2 is allowed.

Research works, Flowguard Hu *et al.* (2014), and Brew Pisharody *et al.* (2017, 2016), focus only on Firewall as a use-case for security policy conflict detection, and fail to identify *Transitive Violations.*

7.2 Object-Oriented Conflict Checking Framework

In the object-oriented paradigm (OOP), a class is defined in terms of network elements, e.g., a virtual network function (VNF) - Firewall/IDS, subnet range (192.168.1.0/24), security configuration - firewall or network address translation (NAT) rules. Moreover, the class can be defined at the granularity of security policy rules. Next, I will discuss the object-oriented dependency relations between network elements.

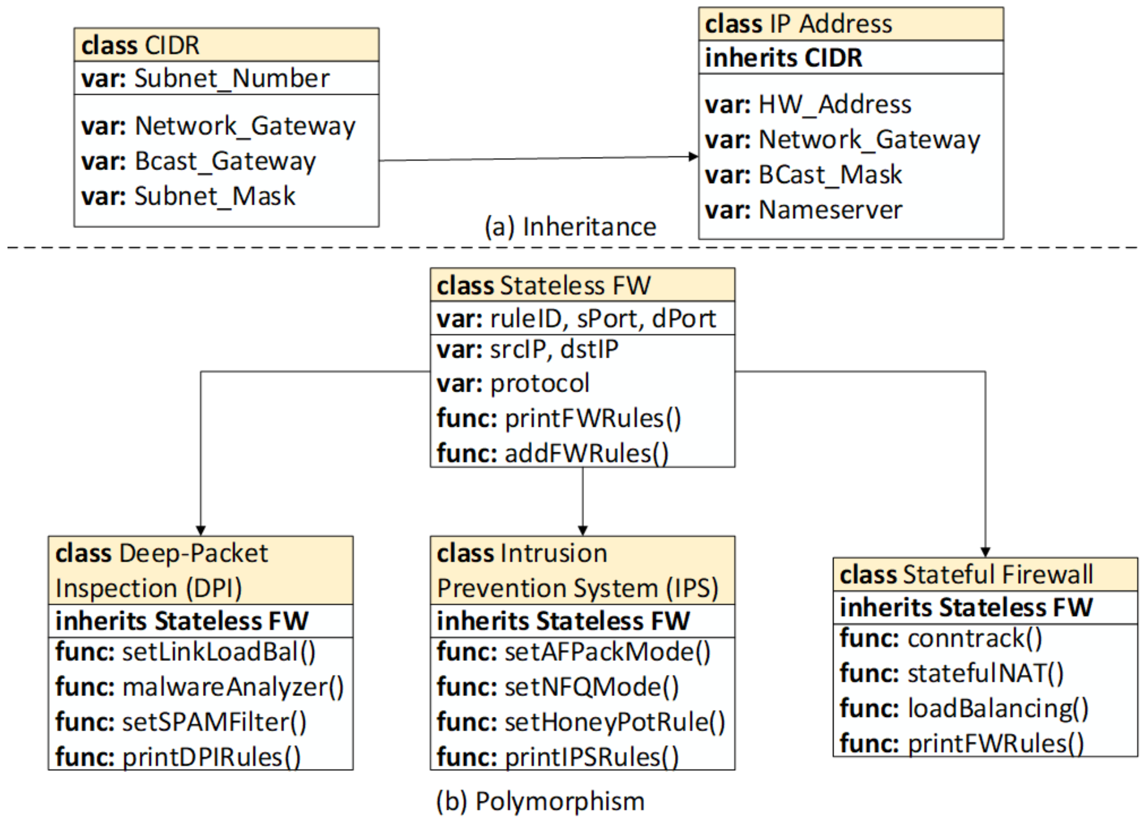


Figure 7.1: Object-oriented Relations (a) Inheritance - IP Address Inherits Properties of Classless Inter-domain Routing (CIDR) Superclass (B) Polymorphism - Stateless Firewall can be Specialized as Stateful Firewall, Intrusion Prevention System (IPS), or Deep-packet Inspection (DPI) Module.

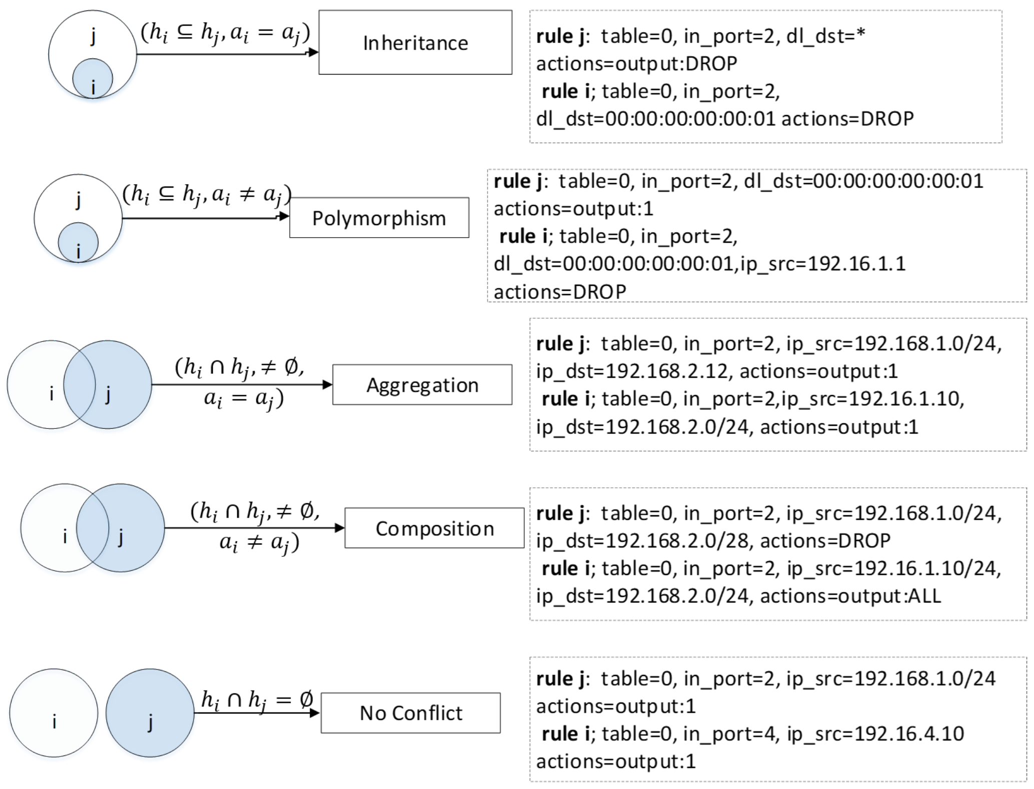
7.2.1 Object Oriented Relations

Class: The basic building block of object-oriented programming is class. A class holds its data members and member functions. In this object-oriented framework, I consider each virtual network function (VNF) as a class in the described object-oriented policy conflict model. For instance, a VNF *Stateless Firewall* is defined as a class. The class can have variables such as source IP address (srcip), source port (sport), and member functions such as *addRules()*, *updateRule()* - Figure 7.4 (a).

Objects: are self-contained components of the class that can access class methods and variables. In this research, I used the object term to refer to a virtual network function instance. The objects allow the rules to be added, deleted, or updated. For instance stateless firewall can have object *f1* that can access member functions *f1.addRule()*, as shown in Figure 7.4 (a). This function allows the object of the class stateless firewall to add a rule. The domain of an object is a subset of object values, i.e., $dom \quad objects \subseteq o.Values$.

Inheritance: This relation helps in identifying sub-class dependencies between network elements. Subclass statements, can take form $C_1 \subseteq C_2$, which means, C_1 is a subclass of C_2. A class that inherits the base class's properties is known as *subclass*, and the base class is referred to as *superclass*. Consider Figure 7.1 (a), *classless inter-domain routing* (CIDR) provides a basic layout of the network domain, possible sub-networks, subnet mask and broadcast domain. The class *IP Address* inherits features such as *Subnet Mask* from the class CIDR. Additionally, IP Address class also adds additional features such as *IPv4 Address* (192.168.1.12), *Hardware Address*, and *Name Server* (8.8.8.8). An object of class IP Address is called an instance of the class. A specific host with the class variable values defined will act as an instance of this class.

Polymorphism: This is one of the features in the object-oriented paradigm that allows a single action to be performed in different ways. As shown in Figure 7.1 (b), polymorphism in network elements allows the creation of one interface, for instance, Firewall. This interface can be realized in different ways, depending upon the application requirement. In the smart firewall architecture like Cisco-ASA Frahim *et al.* (2014), several security features such as intrusion detection, anti-malware protection, and VPN service are implemented together. The stateless-firewall module's basic features, such as network address translation (NAT), traffic filtering, can be specialized to support features of the smart-firewall architecture. For instance *Stateful Firewall*, *Intrustion Prevention System* (IPS), can inherit class *Stateless Firewall* and add new


j
i
$(h_i \subseteq h_j, a_i = a_j)$
Inheritance
j
i
$(h_i \subseteq h_j, a_i \neq a_j)$
Polymorphism
i
j
$(h_i \cap h_j, \neq \emptyset,$
$a_i = a_j)$
Aggregation
i
j
$(h_i \cap h_j, \neq \emptyset,$
$a_i \neq a_j)$
Composition
i
j
$h_i \cap h_j = \emptyset$
No Conflict

```
rule j: table=0, in_port=2, dl_dst=*
actions=output:DROP
 rule i; table=0, in_port=2,
dl_dst=00:00:00:00:00:01 actions=DROP
```

```
rule j: table=0, in_port=2, dl_dst=00:00:00:00:00:01
actions=output:1
 rule i; table=0, in_port=2,
dl_dst=00:00:00:00:00:01,ip_src=192.16.1.1
actions=DROP
```

```
rule j: table=0, in_port=2, ip_src=192.168.1.0/24,
ip_dst=192.168.2.12, actions=output:1
 rule i; table=0, in_port=2,ip_src=192.16.1.10,
ip_dst=192.168.2.0/24, actions=output:1
```

```
rule j: table=0, in_port=2, ip_src=192.168.1.0/24,
ip_dst=192.168.2.0/28, actions=DROP
 rule i; table=0, in_port=2, ip_src=192.16.1.10/24,
ip_dst=192.168.2.0/24, actions=output:ALL
```

```
rule j: table=0, in_port=2, ip_src=192.168.1.0/24
actions=output:1
 rule i; table=0, in_port=4, ip_src=192.16.4.10
actions=output:1
```

Figure 7.2: OpenFlow Rule Conflict analysis. I Identified Dependencies - Inheritance, Polymorphism, Aggregation, and Composition between OpenFlow Rules by Checking Overlap in Header Space and Actions of the Rules.

features corresponding to characteristic of each specialized module. The class IPS in the example above, adds the methods - *setAFPackMode()*, and *setNFQMode()* as shown in the Figure 7.1, in addition to the basic features of Stateless Firewall.

Aggregation: A weak form of association, which enables VNFs to utilize one another without having to re-implement, the functional logic in the original VNF. For instance, Next-Generation Firewall (NGFW), as shown in Figure 7.3(a), typically comes with features such as a virtual private network (VPN) and deep-packet inspection (DPI). The NGFW and VPN can, however, function as standalone VNFs even if either is missing in security architecture. These weak associations allow the re-utilization of desired features amongst VNFs using a *has-A* relationship. Consider the Figure 7.3 (a), *has-A (NGFW, VPN, encryption)*, means NGFW utilizes VPN for

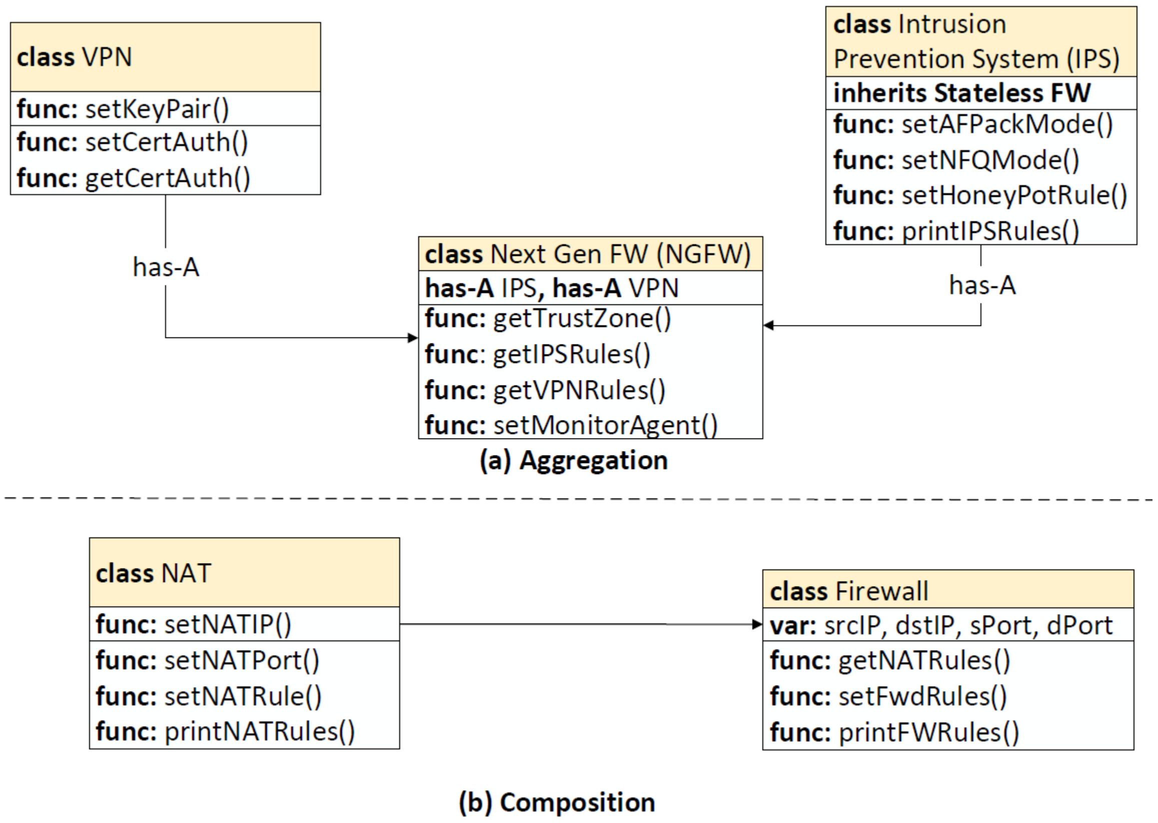
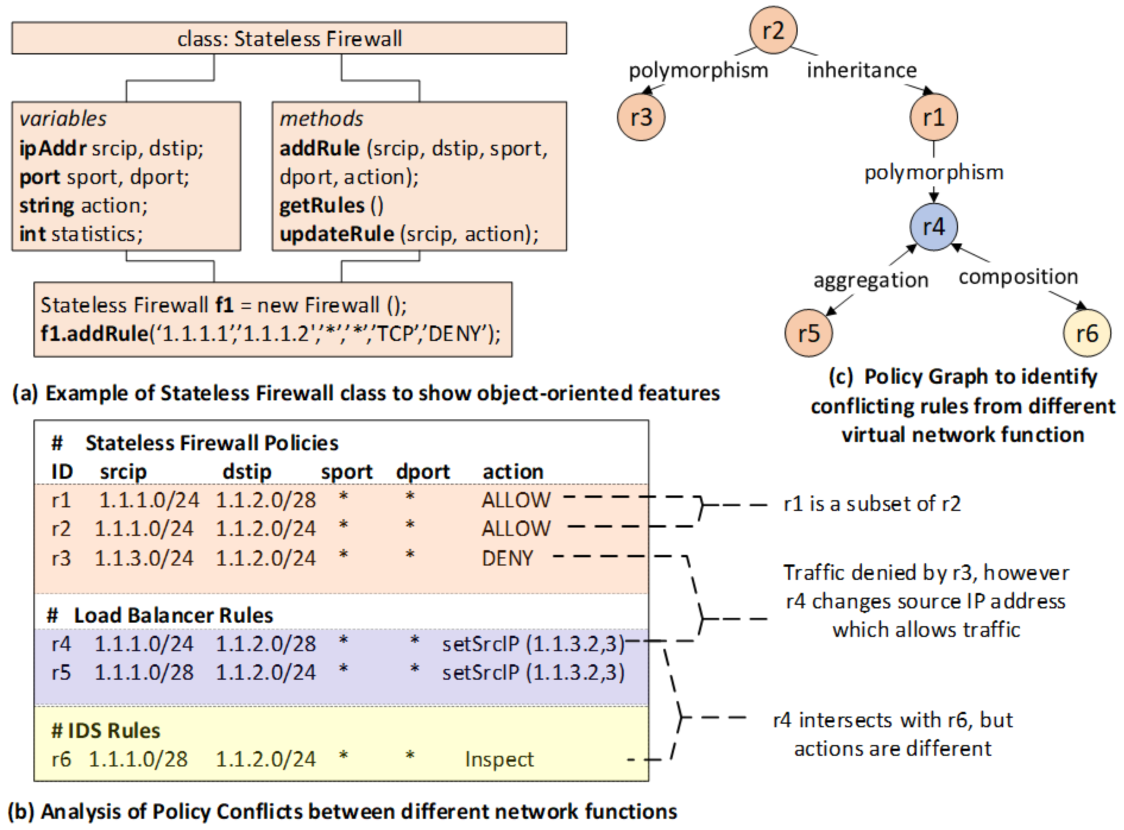


Figure 7.3: Object-oriented Relations (a) Aggregation - Defines Has-a Relation Between Network Elements - a Next Generation Firewall can have both IPS, and VPN Functionality (B) Composition - Part-Of Relation, e.g., set of Functions for Network Address Translation (NAT) are part of Firewall. A subset of Network Features such as NAT can be Part of More than one VNFs.

encryption purpose. Aggregation can be used for representing uncertainty over the partial decomposition. At a more granular level, the next-generation firewall rules can be defined using the has-A relation. For example, *has-A (NGFW, rules, N)*, shows that class NGFW has N number of rules. The rules can be used to ALLOW, FORWARD, or DENY the network traffic between the segments. The encoding of NGFW in terms of weights/priority assignment to individual rules can become a scalability challenge.

Composition: This is a stronger form of association, usually represented by *part-Of* relationship. The functionality of network address translation (NAT) cannot exist by itself, and it requires the presence of Firewall VNF, as shown in Figure 7.3(b). The firewall module can call *setNATIP()*, and *setNATPort()* functions in the class *network address translation* (NAT) in order to allow NAT feature mapping an external IP address/port to an internal IP address/port in addition to other features such as port forwarding and blocking a certain type of network traffic.

7.2.2 *Case Study: Service Function Chaining and Rule Conflicts*


class: Stateless Firewall
variables
methods
r2
polymorphism
inheritance
r3
r1
polymorphism
r4
aggregation
composition
r5
r6
(a) Example of Stateless Firewall class to show object-oriented features
(c) Policy Graph to identify conflicting rules from different virtual network function
r1 is a subset of r2
Traffic denied by r3, however r4 changes source IP address which allows traffic
r4 intersects with r6, but actions are different
(b) Analysis of Policy Conflicts between different network functions

```
variables
ipAddr srcip, dstip;
port sport, dport;
string action;
int statistics;
```

```
methods
addRule (srcip, dstip, sport, dport, action);
getRules ()
updateRule (srcip, action);
```

```
Stateless Firewall f1 = new Firewall ();
f1.addRule('1.1.1.1','1.1.1.2','*','*','TCP','DENY');
```

ID	srcip	dstip	sport	dport	action
# Stateless Firewall Policies					
r1	1.1.1.0/24	1.1.2.0/28	*	*	ALLOW
r2	1.1.1.0/24	1.1.2.0/24	*	*	ALLOW
r3	1.1.3.0/24	1.1.2.0/24	*	*	DENY
# Load Balancer Rules					
r4	1.1.1.0/24	1.1.2.0/28	*	*	setSrcIP (1.1.3.2,3)
r5	1.1.1.0/28	1.1.2.0/24	*	*	setSrcIP (1.1.3.2,3)
# IDS Rules					
r6	1.1.1.0/28	1.1.2.0/24	*	*	Inspect

Figure 7.4: (a) Object-oriented Fundamentals - Class, Variables, Methods for Virtual Network Function (VNF) Stateless Firewall (B) Analysis of Ruleset from Background Section from Object-oriented Relationship Aspect (C) Policy Graph Identifies the Dependencies in the VNF Rules, this will allow the Elimination of Redundant Rules, and scalable Conflict Detection.

Service Function Chain (SFC) refers to an ordered set of service functions that should be applied to the classified traffic. The order of application of abstract service functions can be sequential or parallel, based on the network requirements, e.g., Firewall and IDS can be used in serial order for SFC. There can be policy conflicts induced by the overlapping rules of different Virtual Network Functions (VNFs).

I define the object-oriented fundamentals in this sub-section, and illustrate the possible relationships between different VNFs (Firewall, IDS, IPS). Consider Figure 7.4 (a), the class can be defined as *Stateless Firewall.* The class can have several *variables* such as source and destination IP address (srcip, dstip), *methods* - addRule(srcip, dstip, sport, dport, protocol, action), updateRule(srcip, dstip, ..), getRule(). The class methods accept the variable names as arguments, but some variables can be wildcarded as well. For instance incoming/outgoing traffic between two hosts is allowed on all ports, I can mark sport = '*', dport = '*', in that case. The class can be instantiated to create an object. The class objects in this framework are identified as the specific rules, obtained on substitution of variable values, as shown in Figure 7.4 (a) above. Similarly, consider class as a wrapper for each virtual network function as shown in Figure 7.4 (b). The rules r1,r2,r3 are objects of class *Stateless Firewall*, rule r4, r5 are object of class *Load Balancer*, and the rule r6 is an example of class *IDS* object. As was also discussed in the previous section, rules $r1 \subset r2$, and rules $r3.header \cap r4.header \neq \emptyset$. Moreover, $r4.header \cap r5.header \neq \emptyset$. The object-oriented relations between the rules can be identified using a *Policy Graph*, as shown in the Figure 7.4 (c).

- Rules *r1*, and *r2* can be identified of type *inheritance* since r1 inherits methods of r2, and actions are same for both.

- Rules *r2*, and *r3* have non-empty header, and the actions for both rules are different. These conflicts can be classified under *polymorphism*, as shown in the Figure 7.4.

- Rule r4 inherits all attributes of r1, whereas the actions are different for r4 (action=DENY). Hence this conflict can be classified as a polymorphic dependency.

- Rule r4 and r5 have partial overlap in the header space, and actions are the same. Hence these rules can be classified under *aggregation*.

- Rule r4 and r6 have partial overlap in the header space since source IP is different for both rules, and actions are also different. Hence this dependency can be identified as a *composition* relation, as shown in the policy graph.

The creation of a policy graph will allow the elimination of duplicate rules. For instance, if IDS is adding a new rule, the OOPC framework can proactively identify this as a possible conflict and prevent that particular rule's insertion.

7.2.3 OpenFlow Rule Conflict Detection

The class hierarchy described for different policies in the previous sections can be used to illustrate the process of flow rule conflict identification. Consider the overlap in the action fields. As shown in Figure 7.2, there are four different cases of conflicts in the object-oriented framework, which can cover different scenarios of flow rule conflicts.

As shown in the example above, the header fields $h_i \subseteq h_j$ and actions of both rules are the same, thus rule i is a specialization of rule j, such OpenFlow rules can be classified under *Inheritance* conflict. The example showcases two rules, where rule i inherits the header values of rule j. However, the action fields are different. This is similar to polymorphism property in the object-oriented design. Thus such cases of rules can be classified as *polymorphic* conflicts. Rules *(i, j)* in the example of aggregation have overlapping header fields, i.e., $h_i \cap h_j \neq \emptyset$. However, both rules have similar actions. Thus a third rule, *k*, can replace both rules, but this does not occur automatically in flow tables. These conflicting scenarios can be classified as *aggregation*. Rules *(i, j)* in this conflict scenario have overlapping header fields and conflicting actions. Thus the intersecting part of both rules, $h_i \cap h_j$, is composed of aggregated actions from both rules. This type of rule conflicts can be classified into the *composition* category.

Algorithm 4 Flow Rule Conflict Detection Algorithm

```
1: procedure RULE CONFLICT CHECKING(R)
2:     $R \leftarrow$ current flow rules
3:     $R = \{match(R), A(R)\}$
4:     C $\leftarrow$ Conflict Set
5:     for $i \in$ {1,n} do
6:         for $j \in$ {1,n} do
7:             if $match(R_i) \subseteq match(R_j)$ OR $match(R_j) \subseteq match(R_i)$ AND $action(R_i) == action(R_j)$ then
8:                 C.add(Inheritance)
9:             else if $match(R_i) \subseteq match(R_j)$ OR $match(R_j) \subseteq match(R_i)$ AND $action(R_i) \neq action(R_j)$ then
10:                C.add(Polymorphism)
11:            else if $match(R_i) \cap match(R_j) \neq \emptyset$ AND $action(R_i) == action(R_j)$ then
12:                C.add(Aggregation)
13:            else if $match(R_i) \cap match(R_j) \neq \emptyset$ AND $action(R_i) \neq action(R_j)$ then
14:                C.add(Composition)
15:            end if
16:        end for
17:    end for
18: end procedure
```

Algorithm 4 showcases Conflict Detection based on the current policy graph rules. The policy rules from different security components such as Firewall, IDS, DPI are extracted using 'Flow Conflict Analyzer APIs', described in Figure 7.6. The *lines 7-8* in the algorithm checks if one rule is a subset of another, and if actions are the same. This helps in determining redundant rules in the conflict-resolution phase. The conflict type Inheritance is the conflict category corresponding to such flow rules.

Similarly, *lines 9-10* check for polymorphic conflicts, where one rule is a subset of another, but the actions are different for both rules. Lines *11-14* are used to check partially overlapping rules with the same or different actions - Aggregation and Composition, as discussed in the examples - Figure 7.5, above.

7.2.4 OpenFlow Rule Conflict Resolution

The goal of the flow rule conflict resolution is to construct conflict-free rule sets for the conflict resolver working in a small rule space. Any conflict resolution mechanism employed in the case of OpenFlow should consider factors like overhead associated with the resolution, the importance of a particular rule in terms of volume of traffic processed by rule, the idle time associated with the rule, rule priority, etc. Consider the flow rule conflict resolution mechanism using inherent features of the flow rules, (1) pre-defined priorities, (2) encapsulation, and (3) decomposition. This process is further explained in the algorithm 5.

- **Pre-Defined Priority:** In top of Figure 7.5, rule i is a subset of rule j, and the identified conflict type is polymorphism. The OpenFlow rules *(i,j)*, can use predefined priority to allow traffic corresponding to a given OpenFlow rule and reject the other rule, with lower priority. This can help in resolving *Polymorphism* rule conflicts, since there is a full overlap between the flow rules header space (h_i, h_j), or one rule is a complete subset of another rule, e.g., $r1 \subseteq r2$ in Figure 7.5. The method utilized for priority assignment is dependent on the administrator. For instance, assigning a high priority to a rule originating from a load-balancer is desired from a performance point of view, whereas, if the security is assigned higher importance, an IDS rule with polymorphism conflict with other rules will get a higher priority.

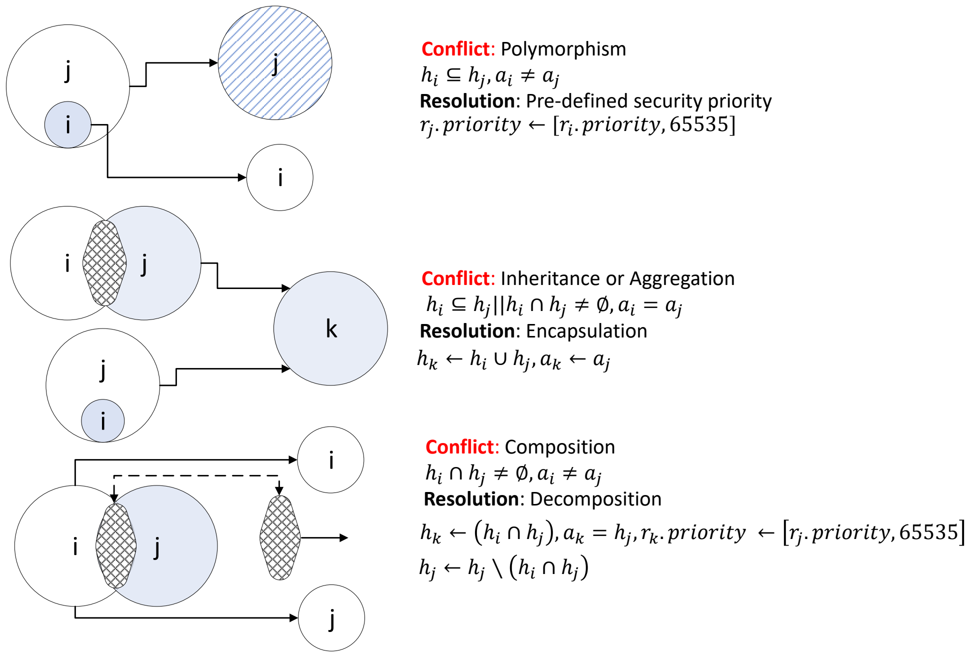

j
i
j
i
Conflict: Polymorphism
$h_i \subseteq h_j, a_i \neq a_j$
Resolution: Pre-defined security priority
$r_j.priority \leftarrow [r_i.priority, 65535]$
i
j
k
j
i
Conflict: Inheritance or Aggregation
$h_i \subseteq h_j || h_i \cap h_j \neq \emptyset, a_i = a_j$
Resolution: Encapsulation
$h_k \leftarrow h_i \cup h_j, a_k \leftarrow a_j$
i
i
j
j
Conflict: Composition
$h_i \cap h_j \neq \emptyset, a_i \neq a_j$
Resolution: Decomposition
$h_k \leftarrow (h_i \cap h_j), a_k = h_j, r_k.priority \leftarrow [r_j.priority, 65535]$
$h_j \leftarrow h_j \setminus (h_i \cap h_j)$

Figure 7.5: Openflow Rule Conflict Resolution. The Priority Assignment can be Used for Resolving the Polymorphism (Full Overlap, Different Actions) Conflicts. The Assignment of Priority Depends on the Network Administrator. The Conflicts with Same Action, i.e., Inheritance, and Aggregation by Using the Object-oriented Concept Encapsulation, i.e., Rule k Header $r_k = r_i \cup r_j$. The Conflicts, where there is Partial Overlap in Header Space but the Actions are Different - Composition, the Concept of De-composition Can Be Used to Resolve These Conflicts (Rule k, Header $h_k = h_i \cap h_j$, and assign Higher Priority to Overlapping Header Space.

- **Encapsulation:** A flow rule conflicting with a large number of rules, i.e., aggregation relation with a large number of conflicting rules, can be eliminated to reduce the number of flow rule conflicts when the action space is the same for the conflicting flow rules. The conflicts *Inheritance* and *Aggregation*, where the header space of the rules is overlapping, and the actions are the same, can be encapsulated into coarser rules. This helps in the elimination of redundant rules, and reduction in the end-to-end flow processing time, as shown in Figure 7.5.

- **Decomposition:** If two OpenFlow rules' header space overlap with each other, but their actions are different, i.e., composition conflict, then such conflicts can be resolved through the decomposition of two rules into three rules. After decomposition, the priorities can be assigned separately to both rules. The decomposition relation needs to recursively check if the decomposed rules conflict with any other rules.

Algorithm 5 Flow Rule Conflict Resolution Algorithm

```
1: procedure Rule Conflict Resolution(R)
2:     $R \leftarrow$ current flow rules
3:     for $i \in \{1,n\}$ do
4:         for $j \in \{1,n\}$ do
5:                                                  ▷ Priority based Resolution
6:             if $\{R_i, R_j\} \in Polymorphism$ then
7:                 $R_i.setPriority([R_j.priority(), 65535])$
8:                                                  ▷ Encapsulation
9:             else if $\{R_i, R_j\} \in (Inheritance||Aggregation)$ then
10:                $R_k.match() = R_i.match() \cup R_j.match()$
11:                $R_k.action() = R_i.action()$
12:                                                 ▷ Decomposition
13:            else if $\{R_i, R_j\} \in Composition$ then
14:                $R_j.match() = \{R_j.match()/\ (R_i \cap R_j).match()\}$
15:                $R_k.match() = (R_i \cap R_j).match()$
16:                $R_k.setPriority([R_j.priority(), 65535])$
17:                $R_k.action() = R_j.action()$
18:            end if
19:        end for
20:    end for
21: end procedure
```

Algorithm 5 shows how OOPC framework utilize the conflict resolution mechanism. The algorithm first obtains all current flow rules and start to identify what type of OO conflict is there, if any. In lines *5-7*, if two rules have polymorphism relationship, then the conflict resolution can be achieved by assigning a higher priority to rule i (although this is an administrator choice to set it either to i or j). If the OO relationship is identified as either inheritance or aggregation, and since the header space $h_i \subseteq h_j \,||\, h_i \cap h_j \neq \phi \;\&\; a_i == a_j$, then a new rule can be defined as R_k to encapsulate the two rules in it. The new rule's header space is assigned the union of the two rule's header space, and the action to either rule *i or j's*, as shown in line *8-11*. Finally, if the two rules have a composition OO relationship, where $h_i \cap h_j \neq \phi \;\&\; a_i \neq a_j$, then the algorithm performs a decomposition of the two rules into three rules, *i,j, and k*. Rule j's header space is equal to the original *j* rule without the mutual attributes with rule *i*, and rule k's header space is equal to the mutual attributes between *i & j*. The actions of rules *i & j* remain the same, whereas rule *k* will have the action of rule *j* since it has higher priority than rule *i* (line *12-17*). Note that rule *i* remains the same.

7.2.5 OOPC Framework Architecture and Data Flow

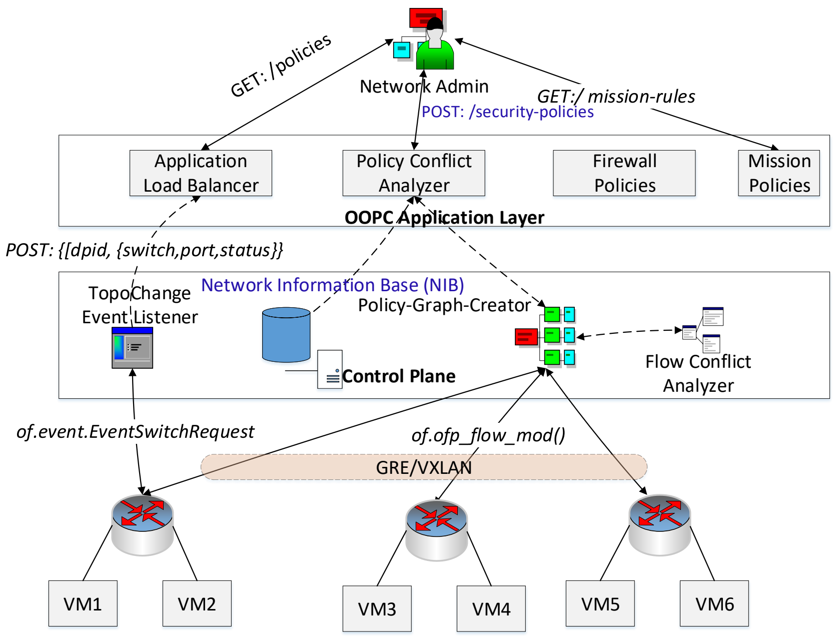


Figure 7.6: OOPC System Architecture. The Application Plane can be Used for Accepting Security Policies and Mission Requirements.

The OOPC architecture in Figure 7.6 is primarily divided into three planes, i.e., *application plane*, responsible for user-interface, through which the user can enter higher-level security policies and mission requirements. The *control plane* consists of modules responsible for the translation of higher-level security policies into OpenFlow rules and identifying and conflicts between OpenFlow rules and security policies. The *data plane* consists of OpenFlow switches with state-tracking capability. Each OpenFlow switch acts as a firewall module for inspecting traffic between the hosts connected to the switch and the traffic between switch and control plane.

Policy Conflict Analyzer: checks the status of security policies across different segments of the cloud network and identifies possible conflicts between security policies and mission requirements at the application plane. The policy conflict analyzer utilizes the object-oriented dependency analysis to check the relationship between the security policies and mission requirements, e.g., Firewall rules related to Load Balancer rules using *Aggregation* relation.

Network Information Base (NIB): acts as a middleware between the application plane implementing distributed firewall policy and local event listeners on each switch. NIB notifies the local-agents on each switch about any new application security policies and maintains synchronization between different agents. NIB has been implemented using Zookeeper Hunt *et al.* (2017).

Policy-Graph-Creator: checks the dependencies between requirements of different security policies and creates end-to-end conflict-free Policy-Graph to direct traffic between different hosts in a data-center. The control plane utilizes this Policy-Graph to modify OpenFlow tables' flow rules, using OpenFlow message $ofp\_flow\_mod()$ and creates an end-to-end traffic flow. This module checks the dependencies between different policies' requirements, utilizing object-oriented (OO) principles such as inheritance, composition, etc. The result of this process is Policy-Graph. The control plane utilizes this Policy-Graph to modify OpenFlow tables' flow rules, using OpenFlow message $ofp\_flow\_mod()$.

Traffic Statistics: The controller consists of topology change event listeners, which listens on the events such as port status (UP/DOWN), switch status, and port information of hosts connected to switches. If there is any topology change, the event listener utilizes a PUSH notification to notify the application plane, which updates the visualization and traffic statistics.

Flow Conflict Analyzer: The conflict analysis module utilizes REST API to fetch the Flow rules from OpenFlow switches using REST API. The rules are analyzed for potential overlap in the header match and action fields, leading to the violation of end-to-end security policies or service delivery. The conflicting rules are corrected using the conflict resolution algorithm, which I will discuss in the subsequent sections.

7.2.6 OOPC Data Flow Diagram

The data flow diagram of OOPC - Figure 7.7 depicts how the incoming network traffic is processed by the OOPC framework. I discuss two main components from the architecture in this subsection, i.e., *Policy-Graph-Composer*, and *Flow Conflict Analyzer*.

- The SDN uses *Packet-In* event to detect new flow in the network. If the traffic has no matching flow rule, i.e., *New-Flow=Y*, the traffic is forwarded to *SFC Composer*.

- The *Classifier* component matches traffic to check policies to be applied to the dynamic network traffic. If there is any conflicting security policy at the application level from other service chains, the resolution mechanism is applied to prevent policy violations, and Policy-Graph is updated. If there is no conflict

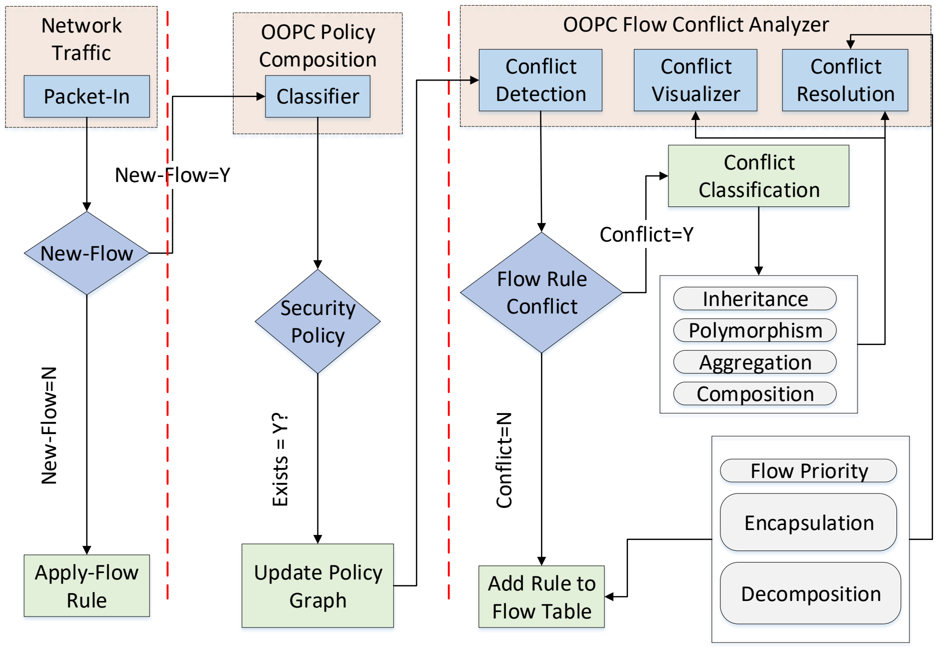


Figure 7.7: OOPC Data-flow Diagram. The New Security Rules or Mission Requirements are First Analyzed for Possible Conflicts by OOPC Policy Composition Module, before Inserting the Corresponding OpenFlow Rule.

involved at the policy level, new nodes are added to Policy-Graph.

- The *OOPC-Event-Listener* checks the policy conflict resolution and additionally listens for any stateful events on the traffic directed from the classifier resolution component. If there is any Intrusion Event or data-plane security attack, the OOPC selects the appropriate flow rule and sends it to the *Conflict Detection* module.

- The *Flow Rule Conflict Analyzer* module receives a Policy-Graph update event and converts the newly added or old-updated policy from the graph to the flow rule as described in previous sections. The new flow rule is checked with existing flow rules to detect and classify the type of conflict - Inheritance, Polymorphism, etc. If there is no conflict, the flow rule generated is added to the flow table.
- The flow rule which has conflict is tagged with conflict type and sent to *Conflict Visualizer* module for visualization, and *Conflict Resolution* module to check the best-fit resolution/mitigation mechanism such as assigning higher-priority, deleting rule with low traffic burst or object-oriented (OO) conflict resolution.

7.3 OOPC Framework Experimental Analysis

In this section, I provide details of the experimental setup, evaluation metrics to show the generalizability and scalability of the OOPC framework. The framework considers security policies from Snort IDS/IPS, stateful Linux firewall, Nginx load balancer, NAT rules from router-based virtual network function (VNF).

Table 7.2: OOPC Components Used in Implementation

Component	**LOC/Version**	**Language / Framework**
SDN Controller	OpenDaylight Carbon, ONOS	Java, REST APIs
Policy-Graph	500	python with Flask APIs
Flow Conflict Analyzer	700	python, networkx
Flow-Visualizer	250	python, d3, REST APIs
Data-Plane	200	Linux container LXC-3.0
Frontend/UI	400	php-lavarel

I utilized an OpenStack based cloud network comprising of two Dell R620 servers and two Dell R710 servers all hosted in the data center to create a system with different VNFs. Each Dell server has about 128 GB of RAM and 16 core CPU. The SDN controller Opendaylight-Carbon was provided network management and orchestration in our framework. In addition to these components - Table 7.2, I used the latest version of Open vSwitch (OVS 2.9.0), with a conntrack module enabled to support the data plane connection tracking.

7.3.1 Policy Composition Time Comparative Analysis

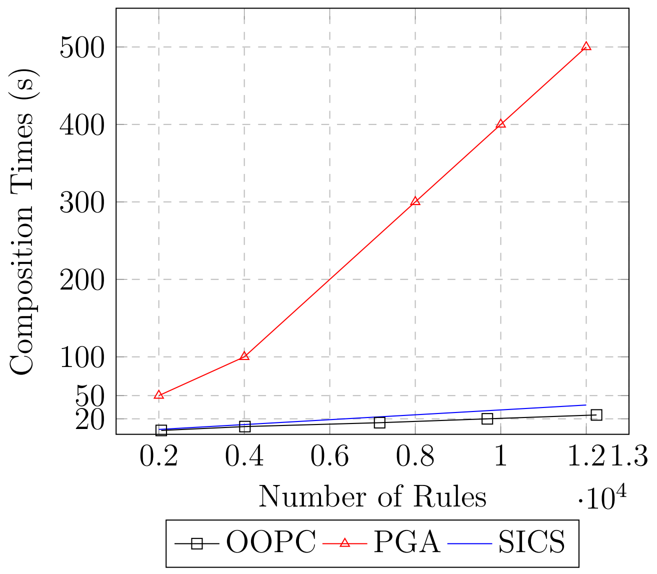


Figure 7.8: Number of Rules vs Composition Time - OOPC, PGA Prakash *et al.* (2015), SICS Wang *et al.* (2016a)

I performed a comparative analysis of composition time for OOPC against policy composition time of PGA Prakash *et al.* (2015) and SICS framework Wang *et al.* (2016a). I used rules as a generic term to define PGA nodes, SICS rules, and OpenFlow rules and to have a standard comparison format. It can be observed that OOPC achieves faster composition time - 20s for 10k rules and 25s for about 12k rules. SICS's composition time was slightly higher than the OOPC framework, i.e., 31.5s for 10k rules and 37.5s for 12k rules. The overall performance gain for OOPC is 37% compared to SICS. The composition time for PGA scales poorly with the number of rules shown in Figure 7.8. PGA takes about 400s for the composition of 10k rules and 500s for 12k rules. The performance degradation in SICS can be attributed to encryption overhead. In contrast, in PGA, the poor scaling is because of duplication of service functions (SFs) across the network. The comparison of OOPC with these frameworks shows that OOPC will scale well with the number of policy rules. The performance of OOPC can be attributed to the fact that the OOPC framework uses encapsulation methods over individual security functions, at a class level, based on a coarse-grained traffic match. Hence the policy composition operation induces limited overhead. Thus, the policy composition does not need to check each security policy against the entire set of security policy database - a procedure of order $O(N^2)$, where N is the number of classes. The policy search operation is reduced to depth-first search (DFS) along with the graph when inserting a new security policy, which is a linear-time operation.

7.3.2 Flow Rule Conflict Analysis

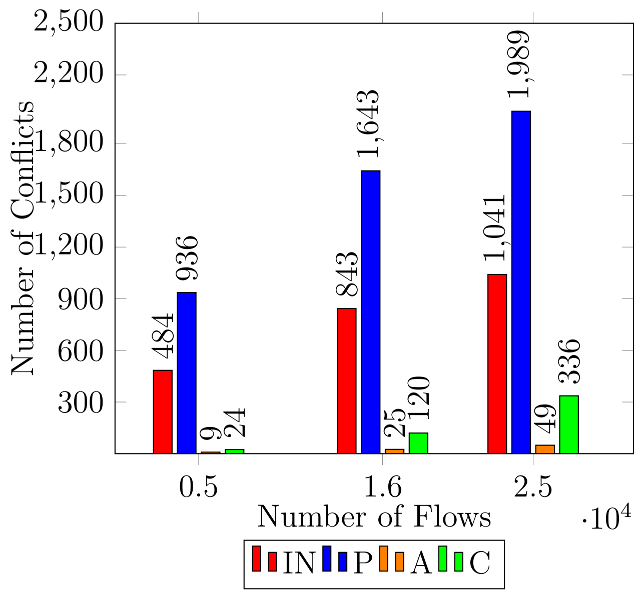


Figure 7.9: Number of Conflicts in OpenFlow Rules

I performed experiment to analyze the number of conflicts - *Inheritance (IN)*, *Polymorphism (P)*, *Aggregation (A)* and *Composition (C)* in the translated OpenFlow rules. The x-axis in the Figure 7.9 denotes the number of OpenFlow rules - 5k, 16k, and 25k. As the number of OpenFlow rules increased, it can be observed that an increase in the number of conflicts. For the dataset with 5k OpenFlow rules, I identified 484 conflicts because of inheritance dependency, 936 polymorphism related conflicts, 9 aggregation conflicts, and 24 composition conflicts. OOPC conflict checking algorithm identified 1041 inheritance conflicts, 1989 polymorphism conflicts, 49 aggregation conflicts, and 336 composition conflicts in 25k OpenFlow rules. The experiment demonstrates that managing conflicts for even a few thousand rules manually can be quite challenging. Hence, using automated detection and resolution framework for flow rule conflicts is a better mechanism for resolving conflicts.

I compared the flow rule conflict detection algorithm with OpenFlow conflict checking research works Brew Pisharody *et al.* (2016) and Flowguard Hu *et al.* (2014). Object-oriented conflict detection can detect transitive conflicts using multi-level inheritance, which both works have not considered. Additionally, the OOPC framework is generalizable to many different VNFs, whereas Brew and Flowguard only analyzed policy conflict issues in a firewall. The experimental results show that there can be several conflicts in flow rules that can be identified only by automated composition and conflict analysis using an object-oriented framework.

7.3.3 Flow Rule Conflict Analysis Scalability

In this experiment, I utilized the Stanford University backbone network topology Kazemian *et al.* (2012) for analyzing the conflict detection algorithm's scalability. The network consists of multiple layers of switches and routers, about 13k routes and 757k forwarding rules, 100 VLANs, and 900 ACL rules. The network was simulated using mininet, routers, and switches were replaced with OVS, and the flow rules from the existing system were inserted using a python script.

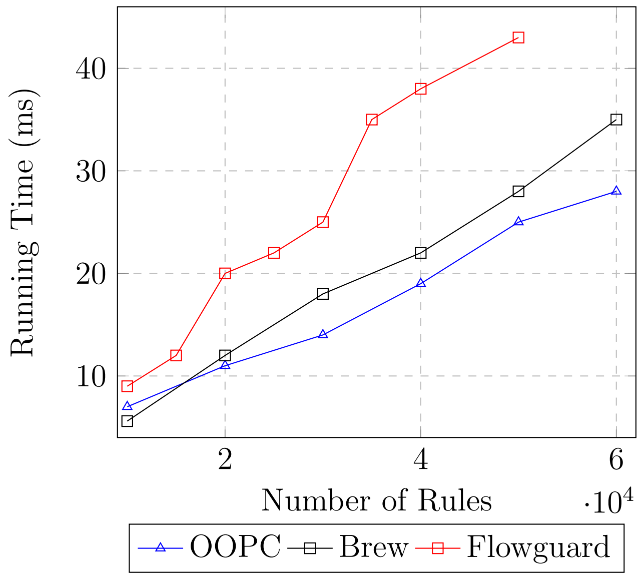


Figure 7.10: Number of Flow Rules vs Policy Conflict Detection Time - OOPC, Brew Pisharody *et al.* (2017), Flowguard Hu *et al.* (2014)

I compared the running time for detecting conflicts of the object-oriented policy conflict detection method, with existing policy conflict detection works, Brew Pisharody *et al.* (2017) and Flowguard Hu *et al.* (2014), which utilize Stanford topology for experimental analysis. The performance of OOPC is slightly slower than Brew for 10k rules ∼9ms, but as the size of the flow rules increases, the OOPC performs better than both Brew and Flowguard. Flowguard only considered conflict detection for about 40k rules, the performance of OOPC conflict checking procedure (19ms) is significantly better than Brew (22ms) and Flowguard (39ms) for 40k rules. The results from the Figure 7.10, shows that the running time for 50k flow rules is 25ms, and about 28ms for 60k flow rules, which is clearly 20% performance gain over Brew (35ms). Hence the flow rule conflict detection algorithm based on object-oriented principles scales well on the large network. The performance gain of OOPC can be

Table 7.3: Conflict Resolution Time Experiment based on Stanford dataset Kazemian *et al.* (2012); Wuyangjack (2018) for Different Conflict Types.

Conflict Type	**No. Rules**	**Resolution Time (sec)**
Composition	7846041	290
Encapsulation	85910	3
Polymorphism	57	5

attributed to the fact that once Policy-Graph is constructed, the search for conflicts in hierarchical structure when a new rule is added, is a trivial operation, compared to the matching of new rule against every other rule in case of Brew, and Flowguard.

7.3.4 Flow Rule Conflict Resolution Analysis

To evaluate proposed approach in Algorithm 5 of resolving convolutions, I created an experiment to analyze the effectiveness of the proposed method in terms of how much time is required to resolve a conflict between a set of flow rules. For this purpose, I used the Stanford Backbone topology dataset Kazemian *et al.* (2012); Wuyangjack (2018) to generate flow rules and examine the conflict between them. I applied proposed Algorithm 5 for conflict resolution and calculated the time required to resolve a conflict that is identified as either *Polymorphism, Encapsulation,* or *Composition.* The results of this experiment are shown in Table 7.3. OOPC based policy conflict resolution mechanism identified over 7 million conflicts of type composition, over 85 thousand of type encapsulation, and 57 of type polymorphism conflicts. A large number of flow rules conflicts identified as Composition or Encapsulation is because in worst case, the number of conflicts can get is up to $O(n^2)$, moreover the actions associated with rules from different network segments lead to overlapping header space, but different actions. OOPC can identify conflicts of polymorphism in about 5 seconds, whereas Brew Pisharody *et al.* (2017) was able to identify the conflict for the same number of flow rules in over 100 seconds. Moreover, OOPC conflict resolution mechanism is able to identify conflicts of different types such as composition where the header space of the flow rules overlap but the actions of the rules are different. On the contrary Brew Pisharody *et al.* (2017) only identify conflicts of priority, which is equivalent to polymorphism conflict type. In effect, OOPC identifies and corrects broad range of policy conflicts, not present in current research work.

7.4 Summary and Discussion

In this research work I presented the design and implementation of object-oriented policy checking (OOPC) framework, which addresses the multi-tier security policy conflict issue, and at the same time scales well on a large cloud network. The security policies and the mission requirements, implemented using application plane in form of network intent or access control, can conflict with each other. The presence of multiple controller, further complicates the issue, with varied semantics of policy specification presented by each SDN controller. Moreover, when security policies are converted into OpenFlow rules at infrastructure level, they can have overlapping address space and conflicting actions. Using the simplified object-oriented dependencies between different policies it is easier to identify which policies conflict with each other, and their relative importance in form of dependency weights. Once the policies are converted into OpenFlow rules, I identified object-oriented relationships between OpenFlow rules, to classify potential conflicts. Finally, I provided guidelines for policy conflict resolution. I plan to extend this work to incorporate stateful policies and implement a fully functional stateful distributed firewall as a future work. These topics have been described next.

Distributed Stateful-SDN Security is required to deal with attacks originating in SDN data-pane, as discussed by Bosshart *et al.* (2013). Openstate Bianchi *et al.* (2014) extended the OpenFlow switch to define a state-transition variable and extended finite-state-machine (XFSM) table, which is able to handle scenarios such as port knocking and TCP SYN-ACK message verification. The design is, however, based on centralized firewall architecture. I plan to extend the work presented in this paper based on the recommendations defined in NIST 800-125b Chandramouli and Chandramouli (2016) for protecting workloads within the data-center and implement a next-generation distributed firewall (NDFW) model. Onix Koponen *et al.* (2010) uses a distributed control plane design for the SDN environment. In this work, I have used similar design principles for OOPC, like a distributed virtual switch and network information base (NIB). P4 Bosshart *et al.* (2014) is a programming language that allows protocol-independent packet processing and stateful packet inspection. I plan to extend the current work and develop a programming platform based on distributed firewall architecture.

Service Function Chaining introduces ordering and placement issues, e.g., heterogeneous throughput and resource configurations. Ghaznavi *et al.* (2016) considers the optimal deployment of the service chain as a mixed-integer linear programming problem. Second, the order in which virtual network functions are applied can introduce challenges, e.g., Firewall being deployed before a load-balancer can cause service outages. Moreover, inefficient decomposition of VNFs can incur communication costs on the network, as highlighted by Xu *et al.* (2017). The OOPC framework analyzed the rules in terms of conflicts introduced by rules from individual VNFs; however, identifying the correct order of VNFs itself has not been considered in the current research work. Sahhaf *et al.* (2015) propose an alternative strategy for minimizing mapping between service function requirements and infrastructure capabilities using Optimal decomposition of network service chains. This research has not considered service function decomposition and correct ordering in the current scope.

Chapter 8

CONCLUSION

The management of security in an enterprise cloud network is quite a challenging task. It is difficult to model the network infrastructure, vulnerabilities, possible attack paths in a scalable fashion. In the first part of this thesis, I investigated scalable attack representation methods using an attack graph as a tool to model network connectivity and vulnerability information. While traditional attack graph generation methods suffer from scalability challenges, in Chapter 2, I discussed how the partitioning of large attack graphs, based on distributed hypergraph partitioning scheme, and using resilient distributed datasets provide a scalable cumulative attack graph. The attack graphs are a good modeling tool for multi-stage attacks. In particular, Advanced Persistent Threat (APTs) are an interesting category of multi-stage attacks. The existing research uses anomaly detection schemes based on semi-supervised machine learning models to detect APT attacks. In Chapter 3, I proposed metrics to benchmark existing semi-supervised machine learning models. I highlighted that the datasets used to benchmark ML models for APT attack detection are themselves limited. The existing datasets either do not cover all phases of APT attacks or have easily identifiable attack patterns, which is not a characteristic of the APT attack. I contributed DAPT 2020, an APT dataset, which can be used for benchmarking existing ML models. This dataset covers all the phases of an APT attack. Moreover, DAPT 2020 will help develop better machine learning models for the detection of APT attacks. The limitation of machine learning models also serves as a motivation for using proactive defense methods for dealing with cyberattacks such as APT.

In the second part of this thesis, I highlighted the importance of moving target defense (MTD), a proactive mechanism for dealing with cyber-attacks. This section explains how programmable networks such as Software-defined Networks (SDN) can be combined with game-theoretic models to understand the attacker-defender interactions and select MTD countermeasures. In Chapter 4, I provided a categorization of factors associated with different decision models used for MTD deployment, i.e., Configuration Set (what), Timing Function (when), and Movement Function (how). I considered single-stage attacks - Distributed Denial of Service (DDoS) attacks in this chapter. The attacker defender interactions were modeled as a two-player dynamic game, and the optimal countermeasure was selected by calculating a Nash-Equilibrium over repeated interactions of the two players. The QoS (rate-limit) used in OpenFlow networks to control the traffic bandwidth was used as a reward metric. The results showcase how to deploy SDN-based rate-limiting countermeasures to deter a malicious attacker from mounting DDoS attacks against the critical network applications. In Chapter 5 I expanded the model from a single-stage attack to a multi-stage attack setting. The key motivation behind this was to use attack graphs from Chapter 2 as a basis for formulating a two-player Markov Game. An SDN-framework where the network controller has a complete view of the underlying infrastructure can generate an attack graph, identify an MTD strategy, and deploy MTD countermeasure against multi-stage attacks. This chapter also argues that a domain-specific reward modeling that we use for modeling attacker and defender rewards and transition probabilities (based on CVSS metrics) provides a real-world model for multi-stage attacks. I considered the problem of APT attacks as a special case of multi-stage attacks, which we can model using a Markov Game framework. The optimal mixed strategy that I calculated outperformed min-max pure and uniform random strategies.

The proactive and reactive defense mechanisms help either completely stop the attack progression (best case scenario) or slow down the attacks (average-case scenario). However, we need to ensure the deployed countermeasures are not introducing any network performance and security issues. SDN-based security countermeasures are implemented in the form of security policies. The third part of this thesis is dedicated to multi-stage security policy analysis in a software-defined system such as SDN. I proposed INTPOL, an intent-driven security policy enforcement framework in Chapter 6. INTPOL provides a policy management grammar and abstracts the network admin from different SDN controllers' implementation details. I used a bounded formal model based on Linear Temporal Logic (LTL) to identify issues associated with network policy verification, policy conflict detection, and validation in a scalable fashion. The empirical evaluation in this section establishes the generalizability of INTPOL to both OpenFlow (SDN-based) and hybrid networks (a mixture of traditional and OpenFlow network). The simulation results on a scalable network show the performance improvement obtained by using the INTPOL framework for detecting policy conflict issues at the application plane level. In Chapter 7, I introduced an OpenFlow rule conflict detection and resolution framework based on an Object-oriented paradigm (OOP). I discuss a novel concept of Virtual Network Function (VNF) graph, which helps identify dependencies between different virtual network functions such as Firewall, IDS, and IPS. The properties of an object-oriented framework can be used to classify different kinds of flow-rule conflicts. I also proposed a policy conflict resolution mechanism for each type of flow-rule conflict. The framework's scalability compared to the state of the art policy composition and rule conflict detection frameworks are established using both formal proofs discussed in Appendix B, and the simulation network of Stanford Topology a common benchmarking standard for flow-rule conflict detection and resolution.

While I covered a wide array of security research topics in this thesis, some interesting research questions remain unexplored. I plan to investigate cyber-deception Jajodia *et al.* (2016), a proactive defense mechanism that combines information disclosure, human elements, and social influence to project fake information as real to an attacker to mislead the attacker Wang and Lu (2018). The current research work in cyber deception has utilized controlled network settings to evaluate the effectiveness of cyber deception. I plan to evaluate existing game-theoretic models - Signalling games, Stackelberg game, Markov game, Min-max strategy games in scalable real-world network settings, with humans in the loop. I also plan to explore the topic of micro-segmentation, discussed by Huang *et al.* (2018), which can help segment large scale network into more manageable regions and perform a fine-grained implementation of security policies. Details of some of my research works have been provided in Appendix C. I plan to expand my research work on some of these topics soon.

8.1 Significant Accomplishments

- Presented "Autonomous Security Analysis and Penetration Testing" at DEFCON Red Team Village, 2020.
- Invited by popular information security podcast "Paul's Security Weekly" to talk about use of AI and Machine Learning in Cybersecurity, 2020
- Invited to present research talk "Deception-NET: Build Your Own Deception" at CactusCon 2019.
- Co-founder and Vice President of hacking club DevilSec aimed at teaching offensive and defensive security, 2019. The club has 250+ active members.

- Member of Western Region Cybersecurity Defense Competition (WRCCDC) Black team, responsible for competition design and infrastructure setup, 2019-2020.
- Captained ASU WRCCDC team 2015-2018. Team stood 1^{st} in business injects in 2017, and 3^{rd} in incidence response in 2018.
- ASU New Venture Challenge Award for startup CyNET LLC - ASU 2017. Awarded to startup I co-founded (CyNET LLC) with Dr. Dijiang Huang and James Chung from among 100+ participating ventures.
- Tech Connect Defense Innovation Summit Award for CyNET LLC, Tech Connect, Tampa, FL 2017.
- Computer Science Outstanding Teaching Assistant, Fulton School of Engineering, ASU 2016.
- Co-author, Textbook, Software-Defined Networking and Security: From Theory to Practice. CRC Press, 2018
- ASU Research Computing Governance Board, Student Member, 2018-2019.
- Reviewer, IEEE ComSoC, IEEE TNSM, IEEE TDSC, ACM MTD.

8.2 Research Work Since Proposal Defense

- Ankur Chowdhary, Sabur, Abdulhakim, Dijiang Huang, Neha Vadnere, James Kirby, Myong Kang , "INTPOL: Intent-Driven Security Policy Management for Software Defined Systems", 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2020 [under-review]
- Ankur Chowdhary, Sabur, Abdulhakim, Dijiang Huang, James Kirby, Myong Kang , "Object-Oriented Policy Conflict checking in Cloud Environment

(OOPC)", IEEE Transactions on Dependable and Secure Computing (TDSC), [under-review, 2^{nd} round to be submitted]

- Ankur Chowdhary, Dijiang Huang, Jayasurya Sevalur Mahendran, Daniel Romo, Yuli Deng, Abdulhakim Sabur "Autonomous Security Analysis and Penetration Testing", The 16th International Conference on Mobility, Sensing and Networking (IEEE MSN 2020), 2020.

- Ankur Chowdhary*, Sowmya Myneni*, Abdulhakim Sabur, Sailik Sengupta, Garima Agrawal, Dijiang Huang, Myong Kang, "DAPT 2020 - Constructing a Benchmark Dataset for Advanced Persistent Threats", ACM MLHat 2020.

- Sengupta, Sailik*, Ankur Chowdhary*, Abdulhakim Sabur, Dijiang Huang, Adel Alshamrani, and Subbarao Kambhampati. "A Survey of Moving Target Defenses for Network Security, IEEE Communications Surveys & Tutorials", 2020.

- Sabur, Abdulhakim, Ankur Chowdhary, Dijiang Huang, Myong Kang, Anya Kim, and Alexander Velazquez. "S3: A DFW-based Scalable Security State Analysis Framework for Large-Scale Data Center Networks. In 22nd International Symposium on Research in Attacks, Intrusions and Defenses" (RAID 2019). 2019.

- Sengupta, Sailik, Ankur Chowdhary, Dijiang Huang, and Subbarao Kambhampati. "General Sum Markov Games for Strategic Detection of Advanced Persistent Threats using Moving Target Defense in Cloud Network." In International Conference on Decision and Game Theory for Security, Springer, Cham, 2019.

"PySpark, `https://spark.apache.org/docs/0.9.0/python-programming-guide.html`", URL `https://spark.apache.org/docs/0.9.0/python-programming-guide.html` (2016).

Abhashkumar, A., J.-M. Kang, S. Banerjee, A. Akella, Y. Zhang and W. Wu, "Supporting diverse dynamic intent-based policies using janus", in "Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies", pp. 296–309 (2017).

Al-Shaer, E., Q. Duan and J. H. Jafarian, "Random host mutation for moving target defense", in "International Conference on Security and Privacy in Communication Systems", pp. 310–327 (Springer, 2012).

Albanese, M., A. De Benedictis, S. Jajodia and K. Sun, "A moving target defense mechanism for manets based on identity virtualization", in "2013 IEEE Conference on Communications and Network Security (CNS)", pp. 278–286 (IEEE, 2013).

Algin, R., H. O. Tan and K. Akkaya, "Mitigating selective jamming attacks in smart meter data collection using moving target defense", in "Proceedings of the 13th ACM Symposium on QoS and Security for Wireless and Mobile Networks", pp. 1–8 (ACM, 2017).

Alshamrani, A., S. Guha, S. Pisharody, A. Chowdhary and D. Huang, "Fault tolerant controller placement in distributed sdn environments", in "2018 IEEE International Conference on Communications (ICC)", pp. 1–7 (IEEE, 2018).

Alshamrani, A., S. Myneni, A. Chowdhary and D. Huang, "A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities", IEEE Communications Surveys & Tutorials **21**, 2, 1851–1877 (2019).

Alto, P., "Palo alto next generation firewall", (2018).

Ammann, P., J. Pamula, R. Ritchey and J. Street, "A host-based approach to network attack chaining analysis", in "Computer Security Applications Conference, 21st Annual", pp. 10–pp (IEEE, 2005).

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).

Antoniou, A., A. Storkey and H. Edwards, "Data augmentation generative adversarial networks", arXiv preprint arXiv:1711.04340 (2017).

Arashloo, M. T., Y. Koral, M. Greenberg, J. Rexford and D. Walker, "Snap: Stateful network-wide abstractions for packet processing", in "Proceedings of the 2016 ACM SIGCOMM Conference", pp. 29–43 (2016).

Baier, C. and J.-P. Katoen, *Principles of model checking* (MIT press, 2008).

Basu, K., “Stackelberg equilibrium in oligopoly: an explanation based on managerial incentives”, Economics Letters **49**, 4, 459–464 (1995).

Beckett, R., A. Gupta, R. Mahajan and D. Walker, “A general approach to network configuration verification”, in “Proceedings of the Conference of the ACM Special Interest Group on Data Communication”, pp. 155–168 (2017).

Berardi, D., D. Calvanese and G. De Giacomo, “Reasoning on uml class diagrams”, Artificial intelligence **168**, 1-2, 70–118 (2005).

Berde, P., M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O’Connor, P. Radoslavov, W. Snow *et al.*, “Onos: towards an open, distributed sdn os”, in “Proceedings of the third workshop on Hot topics in software defined networking”, pp. 1–6 (2014).

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).

Biere, A., A. Cimatti, E. M. Clarke, O. Strichman and Y. Zhu, “Bounded model checking”, (2003).

Borghesi, A., A. Bartolini, M. Lombardi, M. Milano and L. Benini, “A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems”, Engineering Applications of Artificial Intelligence **85**, 634–644 (2019).

Bosshart, P., D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, “P4: Programming protocol-independent packet processors”, ACM SIGCOMM Computer Communication Review **44**, 3, 87–95 (2014).

Bosshart, P., G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica and M. Horowitz, “Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn”, in “ACM SIGCOMM Computer Communication Review”, vol. 43, pp. 99–110 (ACM, 2013).

Brewer, R., “Advanced persistent threats: minimising the damage”, Network security **2014**, 4, 5–9 (2014).

Cappers, B. C. and J. J. van Wijk, “Understanding the context of network traffic alerts”, in “2016 IEEE Symposium on Visualization for Cyber Security (VizSec)”, pp. 1–8 (IEEE, 2016).

Carasso, D., *Exploring splunk* (CITO Research New York, USA, 2012).

Carter, K. M., J. F. Riordan and H. Okhravi, “A game theoretic approach to strategy determination for dynamic platform defenses”, in “Proceedings of the First ACM Workshop on Moving Target Defense”, pp. 21–30 (ACM, 2014).

Casado, M., M. J. Freedman, J. Pettit, J. Luo, N. McKeown and S. Shenker, “Ethane: Taking control of the enterprise”, in “ACM SIGCOMM Computer Communication Review”, vol. 37, pp. 1–12 (ACM, 2007).

Chandola, V., A. Banerjee and V. Kumar, “Anomaly detection: A survey”, ACM computing surveys (CSUR) **41**, 3, 1–58 (2009).

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

Chowdary, A., D. Huang, J. S. Mahendran, D. Romo, Y. Deng and A. Sabur, “Autonomous security analysis and penetration testing”, (2020).

Chowdhary, A., *Secure Mobile SDN* (Arizona State University, 2015).

Chowdhary, A., A. Alshamrani and D. Huang, “Supc: Sdn enabled universal policy checking in cloud network”, in “2019 International Conference on Computing, Networking and Communications (ICNC)”, pp. 572–576 (IEEE, 2019a).

Chowdhary, A., A. Alshamrani, D. Huang and H. Liang, “Mtd analysis and evaluation framework in software defined network (mason)”, in “Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization”, pp. 43–48 (ACM, 2018a).

Chowdhary, A., V. H. Dixit, N. Tiwari, S. Kyung, D. Huang and G.-J. Ahn, “Science dmz: Sdn based secured cloud testbed”, in “IEEE Conference onNetwork Function Virtualization and Software Defined Networks”, (2017a).

Chowdhary, A. and D. Huang, “Sdn based network function parallelism in cloud”, in “2019 International Conference on Computing, Networking and Communications (ICNC)”, pp. 486–490 (IEEE, 2019).

Chowdhary, A., D. Huang, G.-J. Ahn, M. Kang, A. Kim and A. Velazquez, “Sdnsoc: Object oriented sdn framework”, in “Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization”, pp. 7–12 (2019b).

Chowdhary, A., D. Huang, A. Alshamrani, M. Kang, A. Kim and A. Velazquez, “Trufl: Distributed trust management framework in sdn”, in “ICC 2019-2019 IEEE International Conference on Communications (ICC)”, pp. 1–6 (IEEE, 2019c).

Chowdhary, A., D. Huang, A. Alshamrani, A. Sabur, M. Kang, A. Kim and A. Velazquez, “Sdfw: sdn-based stateful distributed firewall”, arXiv preprint arXiv:1811.00634 (2018b).

Chowdhary, A., S. Pisharody, A. Alshamrani and D. Huang, “Dynamic game based security framework in sdn-enabled cloud networking environments”, in “Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization”, pp. 53–58 (ACM, 2017b).

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 (2018c).

Chowdhary, A., S. Sengupta, D. Huang and S. Kambhampati, “Markov game modeling of moving target defense for strategic detection of threats in cloud networks”, arXiv preprint arXiv:1812.09660 (2018d).

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).

Cimatti, A., E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella, “Nusmv 2: An opensource tool for symbolic model checking”, in “International Conference on Computer Aided Verification”, pp. 359–364 (Springer, 2002).

Cimatti, A., E. Clarke, F. Giunchiglia and M. Roveri, “Nusmv: A new symbolic model verifier”, in “International conference on computer aided verification”, pp. 495–499 (Springer, 1999).

Clarke, E., A. Biere, R. Raimi and Y. Zhu, “Bounded model checking using satisfiability solving”, Formal methods in system design **19**, 1, 7–34 (2001).

Colbaugh, R. and K. Glass, “Predictability-oriented defense against adaptive adversaries”, in “2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)”, pp. 2721–2727 (IEEE, 2012).

Competition, W. R. C. D., “WRCCDC”, `https://archive.wrccdc.org/images/2018/` (2018).

CSE-CIC-IDS2018, “A collaborative project between the communications security establishment (cse) and the canadian institute for cybersecurity (cic)”, URL `https://www.unb.ca/cic/datasets/ids-2018.html` (2018).

Cunningham, R. K., R. P. Lippmann, D. J. Fried, S. L. Garfinkel, I. Graf, K. R. Kendall, S. E. Webster, D. Wyschogrod and M. A. Zissman, “Evaluating intrusion detection systems without attacking your friends: The 1998 darpa intrusion detection evaluation”, Tech. rep., MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB (1999).

Davis, J. and M. Goadrich, “The relationship between precision-recall and roc curves”, in “Proceedings of the 23rd international conference on Machine learning”, pp. 233–240 (2006).

Debroy, S., P. Calyam, M. Nguyen, A. Stage and V. Georgiev, “Frequency-minimal moving target defense using software-defined networking”, in “Computing, Networking and Communications (ICNC), 2016 International Conference on”, pp. 1–6 (IEEE, 2016).

Devine, K. D., E. G. Boman, R. T. Heaphy, R. H. Bisseling and U. V. Catalyurek, “Parallel hypergraph partitioning for scientific computing”, in “Proceedings 20th IEEE International Parallel & Distributed Processing Symposium”, pp. 10–pp (IEEE, 2006).

Dhanabal, L. and S. Shantharajah, “A study on nsl-kdd dataset for intrusion detection system based on classification algorithms”, International Journal of Advanced Research in Computer and Communication Engineering **4**, 6, 446–452 (2015).

Ding, C. and X. He, “K-means clustering via principal component analysis”, in “Proceedings of the twenty-first international conference on Machine learning”, p. 29 (2004).

Dragon, D., “Double dragon: Apt41, a dual espionage and cyber crime operation”, `https://content.fireeye.com/apt-41/rpt-apt41`, (Accessed on 07/29/2020) (2020).

Du, M., F. Li, G. Zheng and V. Srikumar, “Deeplog: Anomaly detection and diagnosis from system logs through deep learning”, in “Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security”, pp. 1285–1298 (2017).

DVWA, U., “Damn vulnerable web application”, URL `http://www.dvwa.co.uk/` (2020).

El Mir, I., A. Chowdhary, D. Huang, S. Pisharody, D. S. Kim and A. Haqiq, “Software defined stochastic model for moving target defense”, in “International Afro-European Conference for Industrial Advancement”, pp. 188–197 (Springer, 2016).

Emmott, A. F., S. Das, T. Dietterich, A. Fern and W.-K. Wong, “Systematic construction of anomaly detection benchmarks from real data”, in “Proceedings of the ACM SIGKDD workshop on outlier detection and description”, pp. 16–21 (2013).

Eppstein, D. and S. Muthukrishnan, “Internet packet filter management and rectangle geometry”, in “Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms”, pp. 827–835 (Society for Industrial and Applied Mathematics, 2001).

Evans, A., R. France, K. Lano and B. Rumpe, “The uml as a formal modeling notation”, in “International Conference on the Unified Modeling Language”, pp. 336–348 (Springer, 1998).

Feamster, N., J. Rexford and E. Zegura, “The road to sdn: an intellectual history of programmable networks”, ACM SIGCOMM Computer Communication Review **44**, 2, 87–98 (2014).

Fontugne, R., P. Borgnat, P. Abry and K. Fukuda, “Mawilab: combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking”, in “Proceedings of the 6th International COnference”, p. 8 (ACM, 2010).

Frahim, J., O. Santos and A. Ossipov, *Cisco ASA: All-in-one Next-Generation Firewall, IPS, and VPN Services* (Cisco Press, 2014).

Friedberg, I., F. Skopik, G. Settanni and R. Fiedler, “Combating advanced persistent threats: From network event correlation to incident detection”, Computers & Security **48**, 35–57 (2015).

Ghaznavi, M., N. Shahriar, R. Ahmed and R. Boutaba, “Service function chaining simplified”, arXiv preprint arXiv:1601.00751 (2016).

Goodfellow, I., “Nips 2016 tutorial: Generative adversarial networks”, arXiv preprint arXiv:1701.00160 (2016).

Guerrero, D., A. A. Carsteanu, R. Huerta and J. B. Clempner, “An iterative method for solving stackelberg security games: A markov games approach”, in “Electrical Engineering, Computing Science and Automatic Control (CCE), 2017 14th International Conference on”, pp. 1–6 (IEEE, 2017).

Han, B., V. Gopalakrishnan, L. Ji and S. Lee, “Network function virtualization: Challenges and opportunities for innovations”, IEEE Communications Magazine **53**, 2, 90–97 (2015).

Han, Y., J. Li, D. Hoang, J.-H. Yoo and J. W.-K. Hong, “An intent-based network virtualization platform for sdn”, in “2016 12th International Conference on Network and Service Management (CNSM)”, pp. 353–358 (IEEE, 2016).

Hayes, M. A. and M. A. Capretz, “Contextual anomaly detection framework for big sensor data”, Journal of Big Data **2**, 1, 2 (2015).

Hochreiter, S. and J. Schmidhuber, “Long short-term memory”, Neural computation **9**, 8, 1735–1780 (1997).

Hodge, V. and J. Austin, “A survey of outlier detection methodologies”, Artificial intelligence review **22**, 2, 85–126 (2004).

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).

Houmb, S. H., V. N. Franqueira and E. A. Engum, “Quantifying security risk level from cvss estimates of frequency and impact”, JSS **83**, 9, 1622–1634 (2010).

House, W., “Trustworthy cyberspace: Strategic plan for the federal cyber security research and development program”, Report of the National Science and Technology Council, Executive Office of the President (2011).

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, 2014).

Huang, D., A. Chowdhary and S. Pisharody, *Software-Defined networking and security: from theory to practice* (CRC Press, 2018).

Huang, H., S. Zhang, X. Ou, A. Prakash and K. Sakallah, “Distilling critical attack graph surface iteratively through minimum-cost sat solving”, in “Proceedings of the 27th Annual Computer Security Applications Conference”, pp. 31–40 (ACM, 2011).

Hunt, P., M. Konar, F. P. Junqueira and B. Reed, “Zookeeper: Wait-free coordination for internet-scale systems.”, (2017).

Ingols, K., R. Lippmann and K. Piwowarski, “Practical Attack Graph Generation for Network Defense”, in “null”, pp. 121–130 (IEEE, 2006).

Jacobs, A. S., R. J. Pfitscher, R. A. Ferreira and L. Z. Granville, “Refining network intents for self-driving networks”, in “Proceedings of the Afternoon Workshop on Self-Driving Networks”, pp. 15–21 (2018).

Jafarian, J. H., E. Al-Shaer and Q. Duan, “Openflow random host mutation: Transparent moving target defense using software defined networking”, in “Proceedings of the first workshop on Hot topics in software defined networks”, pp. 127–132 (ACM, 2012).

Jain, S., A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, “B4: Experience with a globally-deployed software defined wan”, in “ACM SIGCOMM Computer Communication Review”, vol. 43, pp. 3–14 (ACM, 2013).

Jajodia, S., N. Park, E. Serra and V. Subrahmanian, “Share: A stackelberg honey-based adversarial reasoning engine”, ACM Transactions on Internet Technology (TOIT) **18**, 3, 30 (2018).

Jajodia, S., V. Subrahmanian, V. Swarup and C. Wang, *Cyber deception*, vol. 6 (Springer, 2016).

Jang, Y., S. P. Chung, B. D. Payne and W. Lee, “Gyrus: A framework for user-intent monitoring of text-based networked applications.”, in “NDSS”, (2014).

Jha, S., O. Sheyner and J. Wing, “Two formal analyses of attack graphs”, in “Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW-15”, pp. 49–63 (IEEE, 2002).

Jiang, Y., C. Zeng, J. Xu and T. Li, “Real time contextual collective anomaly detection over multiple data streams”, Proceedings of the ODD pp. 23–30 (2014).

Jonathan Hart, “Basic ONOS Tutorial”, `https://wiki.onosproject.org/display/ONOS/SDN-IP` (Accessed: June, 5,2020).

Kampanakis, P., H. Perros and T. Beyene, “Sdn-based solutions for moving target defense network protection”, in “World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a”, pp. 1–6 (IEEE, 2014).

Karypis, G. and K. Schloegel, “Parallel graph partitioning and sparse matrix ordering”, University of Minnesota, Department of Computer Science and Engineering (2013).

Kaynar, K. and F. Sivrikaya, “Distributed attack graph generation”, IEEE Transactions on Dependable and Secure Computing **13**, 5, 519–532 (2015).

Kazemian, P., M. Chang, H. Zeng, G. Varghese, N. McKeown and S. Whyte, “Real time network policy checking using header space analysis”, in “Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)”, pp. 99–111 (2013).

Kazemian, P., G. Varghese and N. McKeown, “Header space analysis: Static checking for networks.”, in “NSDI”, vol. 12, pp. 113–126 (2012).

Khurshid, A., X. Zou, W. Zhou, M. Caesar and P. B. Godfrey, “Veriflow: Verifying network-wide invariants in real time”, in “Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)”, pp. 15–27 (2013).

Klir, G. and B. Yuan, *Fuzzy sets and fuzzy logic*, vol. 4 (Prentice hall New Jersey, 1995).

Koponen, T., M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, “Onix: A distributed control platform for large-scale production networks.”, (2010).

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

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 (2014).

Langner, R., “Stuxnet: Dissecting a cyberwarfare weapon”, IEEE Security & Privacy **9**, 3, 49–51 (2011).

Lei, C., D.-H. Ma and H.-Q. Zhang, “Optimal strategy selection for moving target defense based on markov game”, IEEE Access **5**, 156–169 (2017).

Lin, P., J. Hart, U. Krishnaswamy, T. Murakami, M. Kobayashi, A. Al-Shabibi, K.-C. Wang and J. Bi, “Seamless interworking of sdn and ip”, in “Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM”, pp. 475–476 (2013).

Littman, M. L., “Markov games as a framework for multi-agent reinforcement learning”, in “Eleventh International Conference on Machine Learning”, (1994).

Machiraju, R. and R. Yagel, “Reconstruction error characterization and control: A sampling theory approach”, IEEE Transactions on Visualization and Computer Graphics **2**, 4, 364–378 (1996).

Maleki, H., S. Valizadeh, W. Koch, A. Bestavros and M. van Dijk, “Markov modeling of moving target defense games”, in “Proceedings of the 2016 ACM Workshop on Moving Target Defense”, pp. 81–92 (ACM, 2016).

Mämmelä, O., J. Hiltunen, J. Suomalainen, K. Ahola, P. Mannersalo and J. Vehkaperä, “Towards micro-segmentation in 5g network security”, in “European Conference on Networks and Communications (EuCNC 2016) Workshop on Network Management, Quality of Service and Security for 5G Networks”, (2016).

Manadhata, P. K., “Game theoretic approaches to attack surface shifting”, in “Moving Target Defense II”, pp. 1–13 (Springer, 2013).

McCumber, J., “Information systems security: A comprehensive model”, in “Proceedings of the 14th National Computer Security Conference”, (1991).

McWhorter, D., “Apt1: exposing one of china’s cyber espionage units”, Mandiant. com **18** (2013).

Medved, J., R. Varga, A. Tkacik and K. Gray, “Opendaylight: Towards a model-driven sdn controller architecture”, in “2014 IEEE 15th International Symposium on”, pp. 1–6 (IEEE, 2014).

Mehmood, Y., M. A. Shibli, U. Habiba and R. Masood, “Intrusion detection system in cloud computing: Challenges and opportunities”, in “2013 2nd National Conference on Information Assurance (NCIA)”, pp. 59–66 (IEEE, 2013).

Merkel, D., “Docker: lightweight linux containers for consistent development and deployment”, Linux journal **2014**, 239, 2 (2014).

Microsoft, “One-class support vector machine”, URL `https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/one-class-support-vector-machine` (2019).

Mininet, “Mininet virtual network”, URL `http://mininet.org/` (2015).

Mjihil, O., D. Huang and A. Haqiq, “Improving attack graph scalability for the cloud through sdn-based decomposition and parallel processing”, in “International Symposium on Ubiquitous Networking”, pp. 193–205 (Springer, 2017).

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning”, nature **518**, 7540, 529–533 (2015).

Moustafa, N. and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)", in "2015 military communications and information systems conference (MilCIS)", pp. 1–6 (IEEE, 2015).

Myneni, S., A. Chowdhary, A. Sabur, S. Sengupta, G. Agrawal, D. Huang and M. Kang, "Dapt 2020-constructing a benchmark dataset for advanced persistent threats", (2020).

Noble, W. S., "What is a support vector machine?", Nature biotechnology **24**, 12, 1565–1567 (2006).

Ou, X., S. Govindavajhala and A. W. Appel, "Mulval: A logic-based network security analyzer.", in "USENIX security symposium", vol. 8, pp. 113–128 (Baltimore, MD, 2005).

OWASP, "Owasp mutillidae 2 project", URL `https://wiki.owasp.org/index.php/` (2020).

Padmanabhan, V. N. and D. R. Simon, "Secure traceroute to detect faulty or malicious routing", ACM SIGCOMM Computer Communication Review **33**, 1, 77–82 (2003).

Pedrosa, L., R. Iyer, A. Zaostrovnykh, J. Fietz and K. Argyraki, "Automated synthesis of adversarial workloads for network functions", in "Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication", pp. 372–385 (2018).

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

Pham, M. and D. B. Hoang, "Sdn applications-the intent-based northbound interface realisation for extended applications", in "2016 IEEE NetSoft Conference and Workshops (NetSoft)", pp. 372–377 (IEEE, 2016).

Pisharody, S., A. Chowdhary and D. Huang, "Security policy checking in distributed sdn based clouds", in "2016 IEEE Conference on Communications and Network Security (CNS)", pp. 19–27 (IEEE, 2016).

Pisharody, S., J. Natarajan, A. Chowdhary, A. Alshalan and D. Huang, "Brew: A security policy analysis framework for distributed sdn-based cloud environments", IEEE Transactions on Dependable and Secure Computing (2017).

Prakash, C., J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma and Y. Zhang, "Pga: Using graphs to express and automatically reconcile network policies", in "ACM SIGCOMM Computer Communication Review", vol. 45, pp. 29–42 (ACM, 2015).

Qu, Z., L. Su, X. Wang, S. Zheng, X. Song and X. Song, "A unsupervised learning method of anomaly detection using gru", in "2018 IEEE International Conference on Big Data and Smart Computing (BigComp)", pp. 685–688 (IEEE, 2018).

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

Ross, R. S., “Managing information security risk: Organization, mission, and information system view”, Special Publication (NIST SP)-800-39 (2011).

Russell, R. and H. Welte, “Linux netfilter hacking howto”, Disponıvel em http://www. netfilter. org/documentation/HOWTO//netfilter-hacking-HOWTO. letter. ps (Junho de 2005) (2002).

Sabur, A., A. Chowdhary, D. Huang, M. Kang, A. Kim and A. Velazquez, “S3: A dfw-based scalable security state analysis framework for large-scale data center networks”, in “22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019)”, pp. 473–485 (2019).

Saha, R. and A. Agarwal, “SDN approach to large scale global data centers”, Proceedings of Open Networking Summit, Santa Clara, California, USA (2012).

Sahhaf, S., W. Tavernier, M. Rost, S. Schmid, D. Colle, M. Pickavet and P. Demeester, “Network service chaining with optimized network function embedding supporting service decompositions”, Computer Networks **93**, 492–505 (2015).

Sawilla, R. E. and X. Ou, “Identifying critical attack assets in dependency attack graphs”, in “European Symposium on Research in Computer Security”, pp. 18–34 (Springer, 2008).

Schlenker, A., O. Thakoor, H. Xu, F. Fang, M. Tambe, L. Tran-Thanh, P. Vayanos and Y. Vorobeychik, “Deceiving cyber adversaries: A game theoretic approach”, in “Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems”, pp. 892–900 (International Foundation for Autonomous Agents and Multiagent Systems, 2018).

Schmerl, B., J. Cámara, G. A. Moreno, D. Garlan and A. Mellinger, “Architecture-based self-adaptation for moving target defense”, Tech. rep., Technical Report CMU-ISR-14-109. Carnegie Mellon University (2014).

Security, O., “Metasploitable unleashed”, URL `https://www.offensive-security.com/metasploit-unleashed/requirements/` (2020).

Sefraoui, O., M. Aissaoui and M. Eleuldj, “Openstack: toward an open-source solution for cloud computing”, International Journal of Computer Applications **55**, 3, 38–42 (2012).

Sengupta, S., A. Chowdhary, D. Huang and S. Kambhampati, “Moving target defense for the placement of intrusion detection systems in the cloud”, in “International Conference on Decision and Game Theory for Security”, pp. 326–345 (Springer, 2018).

Sengupta, S., A. Chowdhary, D. Huang and S. Kambhampati, “General sum markov games for strategic detection of advanced persistent threats using moving target defense in cloud networks”, in “International Conference on Decision and Game Theory for Security”, pp. 492–512 (Springer, 2019).

Sengupta, S., S. G. Vadlamudi, S. Kambhampati, A. Doupé, Z. Zhao, M. Taguinod and G.-J. Ahn, “A game theoretic approach to strategy generation for moving target defense in web applications”, (International Foundation for Autonomous Agents and Multiagent Systems, 2017).

Shapley, L. S., “Stochastic games”, Proceedings of the national academy of sciences **39**, 10, 1095–1100 (1953).

Sharafaldin, I., A. H. Lashkari and A. A. Ghorbani, “A detailed analysis of the cicids2017 data set”, in “International Conference on Information Systems Security and Privacy”, pp. 172–188 (Springer, 2018).

Sherry, J., S. Ratnasamy and J. S. At, “A survey of enterprise middlebox deployments”, (2012).

Sheyner, O. and J. Wing, “Tools for generating and analyzing attack graphs”, in “Proceedings of Vol. 3188 Lecture Notes in Computer Science pp 344-371”, pp. 344–371 (Springer, 2003).

Sheyner, O. M., “Scenario graphs and attack graphs”, PhD Thesis, CMU (2004).

Shi, Y., H. Zhang, J. Wang, F. Xiao, J. Huang, D. Zha, H. Hu, F. Yan and B. Zhao, “Chaos: An sdn-based moving target defense system”, Security and Communication Networks **2017** (2017).

Shiravi, A., H. Shiravi, M. Tavallaee and A. A. Ghorbani, “Toward developing a systematic approach to generate benchmark datasets for intrusion detection”, computers & security **31**, 3, 357–374 (2012).

Shu, Z. and G. Yan, “Ensuring deception consistency for ftp services hardened against advanced persistent threats”, in “Proceedings of the 5th ACM Workshop on Moving Target Defense”, pp. 69–79 (ACM, 2018).

Singh, M., S. Kumar, T. Garg and N. Pandey, “Penetration testing on metasploitable 2”, International Journal of Engineering and Computer Science **9**, 05, 25014–25022 (2020).

Taylor, J., K. Zaffarano, B. Koller, C. Bancroft and J. Syversen, “Automated effectiveness evaluation of moving target defenses: metrics for missions and attacks”, in “Proceedings of the 2016 ACM Workshop on Moving Target Defense”, pp. 129–134 (ACM, 2016).

TheRegister, “Rsa explains how attackers breached its systems • the register”, `https://www.theregister.com/2011/04/04/rsa_hack_howdunnit/`, (Accessed on 07/29/2020) (2020).

Tian, B., X. Zhang, E. Zhai, H. H. Liu, Q. Ye, C. Wang, X. Wu, Z. Ji, Y. Sang, M. Zhang *et al.*, “Safely and automatically updating in-network acl configurations with intent language”, in “Proceedings of the ACM Special Interest Group on Data Communication”, pp. 214–226 (2019).

Trifunovic, A., *Parallel algorithms for hypergraph partitioning* (University of London, 2006).

Ussath, M., D. Jaeger, F. Cheng and C. Meinel, “Advanced persistent threats: Behind the scenes”, in “Information Science and Systems (CISS), 2016 Annual Conference on”, pp. 181–186 (IEEE, 2016).

Vadlamudi, S. G., S. Sengupta, M. Taguinod, Z. Zhao, A. Doupé, G.-J. Ahn and S. Kambhampati, “Moving target defense for web applications using bayesian stackelberg games”, in “Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems”, pp. 1377–1378 (International Foundation for Autonomous Agents and Multiagent Systems, 2016).

Van Dijk, M., A. Juels, A. Oprea and R. L. Rivest, “Flipit: The game of “stealthy takeover””, Journal of Cryptology **26**, 4, 655–713 (2013).

Varadharajan, V., K. Karmakar, U. Tupakula and M. Hitchens, “A policy-based security architecture for software-defined networks”, IEEE Transactions on Information Forensics and Security **14**, 4, 897–912 (2018).

Venkatesan, S., M. Albanese, G. Cybenko and S. Jajodia, “A moving target defense approach to disrupting stealthy botnets”, in “Proceedings of the 2016 ACM Workshop on Moving Target Defense”, pp. 37–46 (ACM, 2016).

Vizel, Y., G. Weissenbacher and S. Malik, “Boolean satisfiability solvers and their applications in model checking”, Proceedings of the IEEE **103**, 11, 2021–2035 (2015).

Vorobeychik, Y. and S. Singh, “Computing stackelberg equilibria in discounted stochastic games (corrected version)”, (2012).

Vulnhub, “Vulnhub badstore”, URL `https://www.vulnhub.com/entry/badstore-123,41/` (2020).

Wagner, R., M. Fredrikson and D. Garlan, “An advanced persistent threat exemplar”, MONTH (2017).

Wang, C. and Z. Lu, “Cyber deception: Overview and the road ahead”, IEEE Security & Privacy **16**, 2, 80–85 (2018).

Wang, H., X. Li, Y. Zhao, Y. Yu, H. Yang and C. Qian, “Sics: Secure in-cloud service function chaining”, arXiv preprint arXiv:1606.07079 (2016a).

Wang, Y., W.-d. Cai and P.-c. Wei, “A deep learning approach for detecting malicious javascript code”, security and communication networks **9**, 11, 1520–1534 (2016b).

Wuyangjack, "wuyangjack/standford-backbone", URL `https://github.com/wuyangjack/standford-backbone` (2018).

Xu, Z., W. Liang, M. Huang, M. Jia, S. Guo and A. Galis, "Approximation and online algorithms for nfv-enabled multicasting in sdns", in "2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)", pp. 625–634 (IEEE, 2017).

Yegnanarayana, B., *Artificial neural networks* (PHI Learning Pvt. Ltd., 2009).

Yuan, Y., S.-J. Moon, S. Uppal, L. Jia and V. Sekar, "Netsmc: A custom symbolic model checker for stateful network verification", in "17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)", pp. 181–200 (2020).

Zaharia, M., M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing", in "Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation", pp. 2–2 (USENIX Association, 2012).

Zhao, Z., F. Liu and D. Gong, "An sdn-based fingerprint hopping method to prevent fingerprinting attacks", Security and Communication Networks **2017** (2017).

Zheng, Y., H. Zhang and Y. Yu, "Detecting collective anomalies from multiple spatio-temporal datasets across different domains", in "Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems", pp. 1–10 (2015).

Zhu, Q. and T. Başar, "Game-theoretic approach to feedback-driven multi-stage moving target defense", in "International Conference on Decision and Game Theory for Security", pp. 246–263 (Springer, 2013).

Zhuang, R., S. A. DeLoach and X. Ou, "Towards a theory of moving target defense", in "Proceedings of the First ACM Workshop on Moving Target Defense", pp. 31–40 (ACM, 2014).

APPENDIX

APPENDIX A

DETAILS ON APT DATASET CONSTRUCTION

APPENDIX

A.1 Environment Setup Description

The environment consists of a public network with services comprising known vulnerabilities, scanned and exploited by a remote attacker. The Public VM is connected to external network via Gateway Router (another VM). The storage and memory configurations of all VMs has been specified below.

A.1.1 Public Services

The VM (Public VM, Private VM, and Log Server) was connected over the internal network (192.168.3.0/24). Each VM has multiple docker services exposed to the physical ports of the host VM. For instance *DVWA* is a docker instance of Damn Vulnerable Web Application. The application is running as a container on Public VM. The application has a web server, database server, etc. If the IP of the docker instance is *172.16.0.2* and service runs on port *80*, i.e., 172.16.0.2:80, I mapped it to an open port of Public VM, *172.16.0.2:80* $\rightarrow$ *192.168.3.29:9000.*

The docker images were created using publicly available vulnerable services used in academia and industry for penetration testing. I utilized the Damn Vulnerable Web App (DVWA), Bad Store, Meta exploitable, and OWASP Mutillidae containers. Additionally, I used *TCP Dump* utility and *Snort IDS* to capture the live traffic and attack signatures of known attacks, e.g., SYN Flood, Illegal Access Attempt, etc. I use *filebeat* agent to capture the host logs - *Audit Logs*, *IDS Logs*, *Access Logs*, etc. and send them to Log Server VM over internal IP address of the Elastic Search (192.168.3.31:9200). The details of services hosts on the public network has been provided below:

APPENDIX

1. Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to help security professionals test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications, and aid teachers/students to teach/learn web application security in a classroom environment.

2. BadStore.net presents a typical three-tier web storefront application. This self-contained application was built from the ground up with typical security mistakes to serve as a platform for demonstration, security training, evaluation, and testing purposes.

3. OWASP Mutillidae II is a free, open-source, deliberately vulnerable web-application providing a target for web-security enthusiasts. Mutillidae can be installed on Linux and Windows using LAMP, WAMP, and XAMMP. It is pre-installed on SamuraiWTF and OWASP BWA. The existing version can be updated on these platforms.

4. Metasploitable is a VM built from the ground up with a large number of security vulnerabilities. It is intended to be used as a target for testing exploits with Metasploit.

5. CICFlowMeter CSE-CIC-IDS2018 (2018) is a network traffic flow generator written in Java and offers more flexibility in choosing the features you want to calculate, adding new ones, and having better control of the duration of the flow timeout. It generates Bidirectional Flows (Biflow), where the first packet determines the forward (source to destination) and backward (destination to source) directions. Hence the 83 statistical features such as Duration, Number of packets, Number of bytes, Length of packets, etc. are also calculated separately.

6. Snort Roesch *et al.* (1999) is an open-source network intrusion prevention system capable of performing real-time traffic analysis and packet logging on IP networks. It can perform protocol analysis and content searching/matching and can be used to detect various attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, and OS fingerprinting attempts, and much more.

A.1.2 Private Services

Private VM has a list of services with known vulnerabilities; however, the VM (192.168.3.30) is connected to only an internal network. This VM has a list of services relatively harder to exploit and require more in-depth application security and service protocol understanding by the penetration tester/ attacker. I utilized *VulnHub*, a collection of Open Source vulnerable docker environments for composing docker containers on this VM. Current set up hosts vulnerable services: WordPress website, FTP 2.3.4, DNS. MySQL, vulnerable Web application, Nessus, and SAMBA service, exposed to the ports of VM. A prerequisite on this machine is “Docker Compose” apart from the docker application. I used *filebeat* agent to capture the host logs - *Audit Logs*, *Access Logs*, etc. and send them to Log Server VM similar to Public VM. The details of services hosts on the private network has been provided below:

1. Nexus Repository Manager 3 - missing access controls and Remote Code Execution vulnerabilities.

2. Mysql consists of CVE-2012-2122. Details on how to exploit the vulnerability are present in the README file inside the vulnhub source directory.

3. WordPress 4.6 - Remote code execution vulnerability CVE-2016-10033.

4. Samba: Consists of CVE-2017-7494, can be exploited using metasploitable.

5. FTP 2.3.4 consists of Backdoor Command Execution Vulnerabilities.
6. DNS service consists of BIND zone-transfer vulnerability. Details on how to exploit the vulnerability are present in the README file inside the vulnhub source directory.
7. Web-app Django $<$ 2.0.8 consists of Open Redirect Vulnerability, CVE-2018-14574.

A.1.3 APT Attack Phases

1. Reconnaissance
   - Scan Applications - Nessus, Web Scarab, Burp Suite. Find vulnerabilities such as XSS, XSRF, SQL Injection, etc.
   - Scan Network - NMap, Portsweep, Mscan, Satan, Ipsweep, Saint. Find systems' fingerprints, network architecture information, etc. The firewall should log deny the event. If multiple denies are seen against unique destination ports from the same origin host within a small window of time, it is safe to assume that some sort of port scanning activity is taking place.
2. Establish Foothold
   - Download or Install Malware - Scanbox, Backdoor Sogu, PoisonIvy, Key-Loggers.
   - R2L - Guess_Password, Ftp_Write, Imap, Phf, Multihop, Warezmaster, Warezclient, SpyXlock, Xsnoop, Snmpguess, Snmpgetattack, Httptunnel, Sendmail, Named.

- C&C Communication - Send the communication to an external server that the malware has been installed. Monitor network traffic originating from a system to an external server after a download of a file or similar network activity.

3. Lateral Movement

   - Credential Compromise - Key Loggers, Hash retrieval, LDAP, Metasploit.
   - Privilege Escalation (U2R) - Buffer_Overflow, Loadmodule, Rootkit, Perl, Sqlattack, Xterm, PS.

4. Internal Reconnaissance

   Same as Reconnaissance above, just from a different source in search of data. IP range might be probed for port 1433 in the case of enumerating SQL servers. Ports 135-139 are usually probed by attackers when in search of network shares.

5. Data Exfiltration

   This phase's data was uploaded to Google Drive, Dropbox, AWS, or any such cloud. Need to baseline against the normal activity of a system.

6. Cover Up

   Deletion of log files, modification of log files etc. Needs host based intrusion detection agent. OUT OF SCOPE for current research.

A.1.4 APT Feature Description

As a part of APT dataset construction, I collected the following features from the network and host logs. The details of features extracted extracted from the data collected are present in the Table A.1.

Table A.1: APT20 Feature Description

fl_dur	Flow duration
tot_fw_pk	Total packets in the forward direction
tot_bw_pk	Total packets in the backward direction
tot_l_fw_pkt	Total size of packet in forward direction
fw_pkt_l_max	Maximum size of packet in forward direction
fw_pkt_l_min	Minimum size of packet in forward direction
fw_pkt_l_avg	Average size of packet in forward direction
fw_pkt_l_std	Standard deviation size of packet in forward direction
Bw_pkt_l_max	Maximum size of packet in backward direction
Bw_pkt_l_min	Minimum size of packet in backward direction
Bw_pkt_l_avg	Mean size of packet in backward direction
Bw_pkt_l_std	Standard deviation size of packet in backward direction
fl_byt_s	Flow byte rate that is number of packets transferred per second
fl_pkt_s	Flow packets rate that is number of packets transferred per second
fl_iat_avg	Average time between two flows
fl_iat_std	Standard deviation time two flows
fl_iat_max	Maximum time between two flows
fl_iat_min	Minimum time between two flows
fw_iat_tot	Total time between two packets sent in the forward direction
fw_iat_avg	Mean time between two packets sent in the forward direction
fw_iat_std	Standard deviation time between two packets sent in the forward direction

fw_iat_max	Maximum time between two packets sent in the forward direction
fw_iat_min	Minimum time between two packets sent in the forward direction
bw_iat_tot	Total time between two packets sent in the backward direction
bw_iat_avg	Mean time between two packets sent in the backward direction
bw_iat_std	Standard deviation time between two packets sent in the backward direction
bw_iat_max	Maximum time between two packets sent in the backward direction
bw_iat_min	Minimum time between two packets sent in the backward direction
fw_psh_flag	Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
bw_psh_flag	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
fw_urg_flag	Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
bw_urg_flag	Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
fw_hdr_len	Total bytes used for headers in the forward direction
bw_hdr_len	Total bytes used for headers in the forward direction
fw_pkt_s	Number of forward packets per second
bw_pkt_s	Number of backward packets per second
pkt_len_min	Minimum length of a flow

pkt_len_max	Maximum length of a flow
pkt_len_avg	Mean length of a flow
pkt_len_std	Standard deviation length of a flow
pkt_len_va	Minimum inter-arrival time of packet
fin_cnt	Number of packets with FIN
syn_cnt	Number of packets with SYN
rst_cnt	Number of packets with RST
pst_cnt	Number of packets with PUSH
ack_cnt	Number of packets with ACK
urg_cnt	Number of packets with URG
cwe_cnt	Number of packets with CWE
ece_cnt	Number of packets with ECE
down_up_ratio	Download and upload ratio
pkt_size_avg	Average size of packet
fw_seg_avg	Average size observed in the forward direction
bw_seg_avg	Average size observed in the backward direction
fw_byt_blk_avg	Average number of bytes bulk rate in the forward direction
fw_pkt_blk_avg	Average number of packets bulk rate in the forward direction
fw_blk_rate_avg	Average number of bulk rate in the forward direction
bw_byt_blk_avg	Average number of bytes bulk rate in the backward direction
bw_pkt_blk_avg	Average number of packets bulk rate in the backward direction
bw_blk_rate_avg	Average number of bulk rate in the backward direction
subfl_fw_pk	The average number of packets in a sub flow in the forward direction

subfl_fw_byt	The average number of bytes in a sub flow in the forward direction
subfl_bw_pkt	The average number of packets in a sub flow in the backward direction
subfl_bw_byt	The average number of bytes in a sub flow in the backward direction
fw_win_byt	Number of bytes sent in initial window in the forward direction
bw_win_byt	Number of bytes sent in initial window in the backward direction
fw_act_pk	Number of packets with at least 1 byte of TCP data payload in the forward direction
fw_seg_min	Minimum segment size observed in the forward direction
atv_avg	Mean time a flow was active before becoming idle
atv_std	Standard deviation time a flow was active before becoming idle
atv_max	Maximum time a flow was active before becoming idle
atv_min	Minimum time a flow was active before becoming idle
idl_avg	Mean time a flow was idle before becoming active
idl_std	Standard deviation time a flow was idle before becoming active
idl_max	Maximum time a flow was idle before becoming active
idl_min	Minimum time a flow was idle before becoming active

APPENDIX B

FORMAL ANALYSIS OF OBJECT-ORIENTED POLICY CHECKING (OOPC) FRAMEWORK

In this appendix, I describe the formal relations between virtual network functions (VNFs) and the Universal Modeling Language (UML) based design principles that I considered for analyzing the dependencies between VNFs and their rules.

B.1 Inheritance

Theorem B.1.1 - The classes C_1, C_2, are related via inheritance relation. The values in the range of objects of the classes can also be related using inheritance relation.

Proof: The proof builds upon the formal proofs of UML Berardi *et al.* (2005); Evans *et al.* (1998) to create a mapping between the VNF class described for network functions and identify rule dependencies in a fast and efficient manner. Consider two VNF classes C_1, C_2, where C_1 is subset of C_2. Consider the objects 'o' and values of the objects.

$$\boxed{C_1 \subseteq C_2 \quad - \quad ①}$$

$$\boxed{\begin{array}{l} \text{dom} \quad \text{objects} \subseteq o.Values \\ \forall o : Value \times o \in dom \\ \text{objects} \rightarrow (objects(o)).self = o \quad - \quad ② \end{array}}$$

I can use the relations ①, ② to define the semantic mapping between class inheritance and object range as a proof for the theorem,

$$\boxed{\begin{array}{l} \quad \forall\ C_1, C_2, \quad iff \quad C_1 \subseteq C_2 \\ \{\mathrm{C}_1\{o : Value \quad | \quad (objects(o))\}\} \subseteq \\ \{\mathrm{C}_2\{o : Value \quad | \quad (objects(o))\}\} \quad - \quad ③ \end{array}}$$

Moreover, the knowledge base K of security rules, if we have pair of classes (C_1, C_2),

$$\boxed{\begin{array}{l} \exists D \in K, s.t. \\ \mathrm{Is}(\mathrm{C}_1, D) \cap Is(C_2, D) \\ \forall x \neg Is(x, C_1) \cup \neg Is(x, C_2) \quad - \quad \textcircled{4} \end{array}}$$

The relation $\textcircled{4}$ holds, i.e., subclasses of the same class, are mutually exclusive. Using relations $\textcircled{3}$ and $\textcircled{4}$ if the parent classes inherit from each other, the domain of object values (rules of VNF in case of described examples) also hold the inheritance relationship.

Corollary B.1.1 - I can use the inheritance relation to associate the rules of virtual network functions if the classes of VNF are associated with inheritance relation.

$$\boxed{(\ \mathrm{h}_i \subseteq h_j, a_i = a_j) \quad \exists \quad Inheritance(r_i, r_j)}$$

Consider headers (h_i, h_j) and actions (a_i, a_j) of the *rule* i and *rule* j respectively. Clearly, $rule \quad i \subseteq rule \quad j$, thus I can associate both rules via inheritance relation, as shown in the illustrative example in Figure 7.2. Rules with conflicts of this nature can be resolve by merging both rules into one single rule, i.e., *Encapsulation*

B.2 Polymorphism

Theorem B.2.1 - If there are two classes such that, attributes of class C_1 and C_2, such that $C_1 \subseteq C_2$, and the classes differ by at least one attribute then polymorphism relation exists between classes. The rules of the classes C_1 and C_2 can also be associated with polymorphism relationship.

Proof: Consider that there is one attribute of class C_2 different from C_1.

$$\boxed{\mathrm{C}_2.attrib \quad \setminus \quad C_1.attrib \geq 1 \quad - \quad \textcircled{5}}$$

Consider the range of attribute values of class C_1, and C_2, for class, based on equation $\textcircled{1}$,

$$\boxed{\begin{array}{l}\mathrm{C}_1,\ range(C_1.attrib) \in range(C_2.attrib) \\ \mathrm{C}_1\{o: Values\} \cap C_2\{o.Values\} \neq 0 \quad - \quad \textcircled{6}\end{array}}$$

Using equation $\textcircled{5}$, and $\textcircled{6}$, it can be established that domain of object values (VNF rules), of the classes with non-equal attributes, can be related with polymorphism relation, provided the property holds for the classes in consideration.

Corollary B.2.1 - Rules r_i is instantiated using object of class C_1, and r_j using object of class C_2, $C_1 \subseteq C_2$,

$$\boxed{(\ \mathrm{h}_i \subseteq h_j, a_i \neq a_j) \quad \exists \quad Polymorphism(r_i, r_j)}$$

Consider rules (i,j), *rule i* from stateless firewall, and *rule j* from an IPS can have conflicting actions,

```
rule i: {table=0, in_port=2, dl_dst=00:00:00:00:00:01,actions=output:1}
rule j: {table=0, in_port=2, dl_dst=00:00:00:00:00:01,actions=DROP}.
```

Clearly, header h_i, of *rule i* is subset of header h_j of *rule j*, but actions $\{a_i = output : 1\} \neq \{\mathrm{a}_j = DROP\}$. I cateogorized these conflicts using *Polymorphism* relation, and utilize *Pre-defined Security Priority*, a policy conflict resolution mechanism to assign higher priority to a rule, which is broader in scope, i.e., *rule j* in this particular example.

B.3 Aggregation

Theorem B.3.1 - If the classes C_1, C_2 are associated using aggregation relation, the rules of the classes can also be associated using aggregation relation.

Proof: Consider classes C_1, and C_2, such that $C_1 \cap C_2 \neq 0$. Formally, in this model I can define objects of a class C_1 have n subparts P of class C_2, for n=1 by default. Considering x as an attribute of class C_1, and *P(x)* as subpart of x.

$$\boxed{\begin{array}{l}\text{has }(\mathrm{C}_1, C_2, P, n)\\ \forall x \forall i \quad Is(x, C_1) \rightarrow Is(P_i(x), C_2)\\ 1 \leq i \leq n; P, n \quad \textit{optional} \quad - \quad \textcircled{7}\end{array}}$$

I can define rules in the form of parameterized classes, like *C(C',n,p)*, as shortened version of *class* C with n-subparts, C_i, where C_i is C with i parts of *C'*, and $P(C_i|C)$ is binomial probability value of i success cases in n trials. For simplicity I kept number of class constructs to minimum. Consider the range of objects that have attribute x, and part-of x, i.e., $y = P_i(x)$, it can be noted that,

$$\boxed{\begin{array}{l}\forall C_1, C_2 \quad iff \quad C_1 \cap C_2 \neq 0\\ \exists x, y = P_i(x), \quad s.t. \;\; \{C_1(x)\{o : Value|(x.object(o)\}\}\\ \cap \quad C_2(y)\{o : Value|(y.object(o)\}\} \neq 0 \quad - \quad \textcircled{8}\end{array}}$$

In this case, the attribute under consideration is *action*, hence using $\textcircled{7}$, and $\textcircled{8}$, if classes are associated with an aggregation relation, I can relate the rules created using an object from those classes using aggregation relation.

Corollary B.3.1 - Rules r_i is instantiated using object of class C_1, and r_j using object of class C_2, $C_1 \cap C_2 \neq 0$,

$$\boxed{(\ \mathrm{h}_i \cap h_j \neq 0, a_i = a_j) \quad \exists \quad \textit{Aggregation}(r_i, r_j)}$$

An example of aggregation in the context of the OOPC model is values of classes C_1, C_2, i.e., if the header space of the rules of two classes is non-overlapping, but actions are similar. Consider *rule i*, and *rule j* from two VNFs that have similar actions,

```
   rule i:  {table=0, in_port=2, ip_src=192.168.1.0/24, ip_dst=192.168.2.12,
actions=output:1}
rule j:  {table=0, in_port=2, ip_src=192.168.1.10, ip_dst=192.168.2.0/28,
```

`actions=output:1`}.

Here the header h_i, of *rule i* intersects with h_j of *rule j*, and the actions $\{a_i = \mathit{output} : 1\} = \{a_j = \mathit{output} : 1\}$. I can identify these rules under the conflict class *Aggregation*. A policy conflict resolution mechanism that combines both rules into one class using *Encapsulation*, can be used for conflict resolution in this case, i.e., *rule k* = *rule i* ∪ *rule j*.

B.4 Composition

Theorem B.4.1 - If the classes C_1, C_2 are associated using composition relation, the rules of the classes can also be associated using composition relation.

Proof: I can represent the relation between subparts P_1, P_2 of the objects of classes C_1, C_2. If no relationship exists between the parts, but there is a partial overlap in the attributes of the class, such that $C_1(P_1) \cap C_2(P_2) \neq 0$. The rule R (C) which does not have any part assignment can be represented using $Is(x, C) \rightarrow R(x)$.

It follows from the relation ⑧, that the domain of object values of the parts P_1, and P_2 have non-empty intersection. The only difference is that range of values associated with the action is different for these objects. Hence, it can be established that the rules created using the object instance of these classes using composition relation as well.

Corollary B.4.1 - Rules r_i is instantiated using object of class C_1, and r_j using object of class C_2, $C_1 \cap C_2 \neq 0$,

$$\boxed{(\ \mathrm{h}_i \cap h_j \neq 0, a_i \neq a_j) \quad \exists \quad Composition(r_i, r_j)}$$

Consider *rule i*, and *rule j* from two VNFs that have non-overlapping header space, $h_i \cap h_j \neq \emptyset$, and $a_i \neq a_j$, as can be seen in the example below,

```
   rule i:  {table=0, in_port=2, ip_src=192.168.1.0/24, ip_dst=192.168.2.0/28,
actions=DROP}
rule j:  {table=0, in_port=2, ip_src=192.168.1.10, ip_dst=192.168.2.0/24,
actions=output:ALL}.
```

I categorized these conflicts as *Composition* since the conflicting rules are composed of overlapping rule sets. Thus, the framework needs a conflict resolution mechanism that decomposes these rules into non-overlapping rules. I call the conflict resolution *Decomposition*.

APPENDIX C

RESEARCH WORKS

C.1 Research Publications

Book

- Huang, Dijiang, Ankur Chowdhary, and Sandeep Pisharody. Software-Defined Networking and Security: From Theory to Practice. CRC Press, 2018.

Conference and Workshop Papers

- Ankur Chowdhary, Dijiang Huang, Jayasurya Sevalur Mahendran, Daniel Romo, Yuli Deng, Abdulhakim Sabur "Autonomous Security Analysis and Penetration Testing", The 16th International Conference on Mobility, Sensing and Networking (IEEE MSN 2020), 2020.
- Ankur Chowdhary*, Sowmya Myneni*, Abdulhakim Sabur, Sailik Sengupta, Garima Agrawal, Dijiang Huang, Myong Kang, "Generating Benchmark Dataset for Advanced Persistent Threat", ACM MLHat 2020.
- Sabur, Abdulhakim, Ankur Chowdhary, Dijiang Huang, Myong Kang, Anya Kim, and Alexander Velazquez. "S3: A DFW-based Scalable Security State Analysis Framework for Large-Scale Data Center Networks. In 22nd International Symposium on Research in Attacks, Intrusions and Defenses" (RAID 2019). 2019.
- Sengupta, Sailik, Ankur Chowdhary, Dijiang Huang, and Subbarao Kambhampati. "General Sum Markov Games for Strategic Detection of Advanced Persistent Threats using Moving Target Defense in Cloud Network." In International Conference on Decision and Game Theory for Security, Springer, Cham, 2019.
- Chowdhary, Ankur, Adel Alshamrani, Dijiang Huang, Myong Kang, Anya Kim, and Alexander Velazquez. "TRUFL: Distributed Trust Management Framework in SDN." IEEE International Conference on Communications (ICC) (2019).

- <u>Chowdhary, Ankur</u>, and Dijiang Huang. "SDN based Network Function Parallelism in Cloud." In 2019 International Conference on Computing, Networking and Communications (ICNC), pp. 486-490. IEEE, 2019.
- <u>Chowdhary, Ankur</u>, Sailik Sengupta, Adel Alshamrani, Dijiang Huang, and Abdulhakim Sabur. "Adaptive MTD Security using Markov Game Modeling." In 2019 International Conference on Computing, Networking and Communications (ICNC), pp. 577-581. IEEE, 2019.
- <u>Chowdhary, Ankur</u>, Adel Alshamrani, and Dijiang Huang. "SUPC: SDN enabled Universal Policy Checking in Cloud Network." In 2019 International Conference on Computing, Networking and Communications (ICNC), pp. 572-576. IEEE, 2019.
- Alshamrani, Adel, <u>Ankur Chowdhary</u>, Oussama Mjihil, Sowmya Myneni, and Dijiang Huang. "Combining Dynamic and Static Attack Information for Attack Tracing and Event Correlation." In 2018 IEEE Global Communications Conference (GLOBECOM), pp. 1-7. IEEE, 2018.
- Sengupta, Sailik, <u>Ankur Chowdhary</u>, Dijiang Huang, and Subbarao Kambhampati. "Moving Target Defense for the Placement of Intrusion Detection Systems in the Cloud." In International Conference on Decision and Game Theory for Security, pp. 326-345. Springer, Cham, 2018.
- Alshamrani, Adel, Sayantan Guha, Sandeep Pisharody, <u>Ankur Chowdhary</u>, and Dijiang Huang. "Fault Tolerant Controller Placement in Distributed SDN Environments." In 2018 IEEE International Conference on Communications (ICC), pp. 1-7. IEEE, 2018.
- <u>Chowdhary, Ankur</u>, Vaibhav Hemant Dixit, Naveen Tiwari, Sukhwa Kyung, Dijiang Huang, and Gail-Joon Ahn. "Science DMZ: SDN based Secured Cloud

Testbed." In 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pp. 1-2. IEEE, 2017.

- Alshamrani, Adel, Ankur Chowdhary, Sandeep Pisharody, Duo Lu, and Dijiang Huang. "A Defense System for Defeating DDoS Attacks in SDN based Networks." In Proceedings of the 15th ACM International Symposium on Mobility Management and Wireless Access, pp. 83-92. ACM, 2017.
- Pisharody, Sandeep, Ankur Chowdhary, and Dijiang Huang. "Security Policy Checking in Distributed SDN based Clouds." In 2016 IEEE Conference on Communications and Network Security (CNS), pp. 19-27. IEEE, 2016.
- El Mir, Iman, Ankur Chowdhary, Dijiang Huang, Sandeep Pisharody, Dong Seong Kim, and Abdelkrim Haqiq. "Software defined Stochastic Model for Moving Target Defense." In International Afro-European Conference for Industrial Advancement, pp. 188-197. Springer, Cham, 2016.
- Myneni Sowmya*, Chowdhary Ankur*, Sabur Abdulhakim, Sengupta Sailik, Agrawal Garima, Dijiang Huang, Kang Myong. "DAPT 2020-Constructing a Benchmark Dataset for Advanced Persistent Threats." In MLHat: The First International Workshop on Deployable Machine Learning for Security Defense. 2020.
- Chowdhary, Ankur, Dijiang Huang, Gail-Joon Ahn, Myong Kang, Anya Kim, and Alexander Velazquez. "SDNSOC: Object Oriented SDN Framework." In Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, pp. 7-12. ACM, 2019.
- Chowdhary, Ankur, Sailik Sengupta, Dijiang Huang, and Subbarao Kambhampati. "Markov Game Modeling of Moving Target Defense for Strategic Detection of Threats in Cloud Networks." In Proceedings of Workshop on Artificial

Intelligence for Cybersecurity (AICS), AAAI (2018).

- Chowdhary, Ankur, Dijiang Huang, Adel Alshamrani, and Hongbin Liang. "MTD Analysis and Evaluation Framework in Software Defined Network (MASON)." In 2018 ACM International Workshop on Security in Software Defined Networks and Network Function Virtualization, SDN-NFVSec 2018, pp. 43-48. Association for Computing Machinery, Inc, 2018.
- Chowdhary, Ankur, Sandeep Pisharody, Adel Alshamrani, and Dijiang Huang. "Dynamic Game based Security Framework in SDN-enabled Cloud Networking Environments." In Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, pp. 53-58. ACM, 2017.
- Chowdhary, Ankur, Sandeep Pisharody, and Dijiang 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.
- Lu, Duo, Zhichao Li, Dijiang Huang, Xianglong Lu, Yuli Deng, Ankur Chowdhary, and Bing Li. "VC-Bots: A Vehicular Cloud Computing Testbed with Mobile Robots." In proceedings of the first international workshop on internet of vehicles and vehicles of internet, pp. 31-36. ACM, 2016.

Journals and Survey Papers

- Sengupta, Sailik*, Ankur Chowdhary*, Abdulhakim Sabur, Dijiang Huang, Adel Alshamrani, and Subbarao Kambhampati. "A Survey of Moving Target Defenses for Network Security, IEEE Communications Surveys & Tutorials", 2020.
- Ankur Chowdhary, Sabur, Abdulhakim, Dijiang Huang, James Kirby, Myong Kang , "Object-Oriented Policy Conflict checking in Cloud Environment

(OOPC)", IEEE Transactions on Dependable and Secure Computing (TDSC), [under-review]

- Alshamrani, Adel, Sowmya Myneni, Ankur Chowdhary, and Dijiang Huang. "A Survey on Advanced Persistent Threats: Techniques, Solutions, Challenges, and Research Opportunities." IEEE Communications Surveys & Tutorials 21, no. 2 (2019): 1851-1877.
- Pisharody, Sandeep, Janakarajan Natarajan, Ankur Chowdhary, Abdullah Alshalan, and Dijiang Huang. "Brew: A Security Policy Analysis Framework for Distributed SDN-based Cloud Environments." IEEE Transactions on Dependable and Secure Computing (2017).

Pending Patents and Preprints

- Pisharody, Sandeep, Ankur Chowdhary, and Dijiang Huang, "US Patent 15795036: Security Policy Analysis Framework for Distributed Software Defined Networking (SDN) based Cloud Environments"
- Ankur Chowdhary, Dijiang Huang, Adel Alshamrani, Abdulhakim Sabur, Myong Kang, Anya Kim, and Alexander Velazquez. "SDFW: sdn-based stateful distributed firewall." arXiv preprint arXiv:1811.00634 (2018).

C.2 Research Works Not Included in Thesis

The topics covered in this thesis include efficient attack modeling techniques - Scalable Attack Graph and Programmable Network-based countermeasure selection. I have been able to work on many other topics involving artificial intelligence and programmable network based security.

Distributed Trust Management in SDN - SDN supports different protocols, third-party applications, and controllers. SDN can help an admin centralize command and control of the cloud environment. A big concern is the trust between various components in SDN itself. There can be several ways in which an attacker can violate trust. A rogue insider can add a fake switch, additional host nodes that are not part of the SDN environment to achieve desired communication. It can be quite hard to detect trust violations in the SDN framework. It is important to ensure that flow rules across the infrastructure are compliant with high Service-Level Agreements (SLAs). If the existing public key-based trust management systems are used in SDN, scalability will be a major concern for large multi-tenant cloud networks. TRUFL Chowdhary *et al.* (2019c) provides a scalable *distributed trust management* framework to shield SDN-infrastructure against rogue-insiders and loss of availability attacks.

Policy and Network Function Management in SDN - A preliminary investigation into issues associated with network policies and unified framework to manage policies from different network functions such as Firewall, IDS, IPS has been discussed in Chowdhary *et al.* (2019a) and Chowdhary *et al.* (2019b). Network Function Parallelism (NFP) SFC-NFP for OpenFlow network improves SFC performance in the cloud network by analyzing the opportunities of parallel implementation of virtual network functions Chowdhary and Huang (2019). The use of microsegmentation Mämmelä *et al.* (2016) and design of stateful distributed firewall managed by SDN has been discussed in Chowdhary *et al.* (2018b). The research work Alshamrani *et al.* (2018) identifies the optimal placement of multiple network controllers in a distributed SDN environment. The placement ensures fault tolerance and optimal network performance. The distributed firewall was utilized to create scalable attack graphs using the microsegementation approach by Sabur *et al.* (2019). I explored the use of SDN for securing a mobile environment in Chowdhary (2015). Educational institutions, like many other industries, face a lot of security threats. I established an SDN enabled Demilitarized Zone (DMZ) - Science DMZ to serve as a testbed for securing the ASU Internet2 environment. Science DMZ Chowdhary *et al.* (2017a) allows researchers to conduct an in-depth analysis of security attacks and take necessary countermeasures using SDN based command and control (C&C) center.

Artificial Intelligence-based Pentesting - Security Assessment of large networks is a challenging task. Penetration testing (pentesting) analyzes a network's attack surface to find security vulnerabilities. I proposed an autonomous security analysis and penetration testing framework (ASAP) Chowdary *et al.* (2020) that creates a map of security threats and possible attack paths. ASAP utilizes Deep-Q Network (DQN) Mnih *et al.* (2015) to identify optimal policy for performing pentesting and incorporates domain-specific transition matrix and reward modeling for capturing the importance of security vulnerabilities and difficulty inherent in exploiting them. ASAP generates autonomous attack plans and validates them against real-world networks. The attack plans are generalizable to the complex enterprise network, and the framework scales well on a large network.