DYNAMIC SHIFTING OF VIRTUAL NETWORK TOPOLOGIES FOR NETWORK ATTACK PREVENTION

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Lenoy Avidan

May 2019

COMMITTEE MEMBERSHIP

TITLE: Dynamic Shifting of Virtual Network Topologies for Network Attack Prevention

AUTHOR: Lenoy Avidan

DATE SUBMITTED: May 2019

COMMITTEE CHAIR: Bruce DeBruhl, Ph.D.
Associate Professor of Computer Science

COMMITTEE MEMBER: John Bellardo, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Aaron Keen, Ph.D.
Professor of Computer Science

ABSTRACT

Dynamic Shifting of Virtual Network Topologies for Network Attack Prevention

Lenoy Avidan

Computer networks were not designed with security in mind, making research into the subject of network security vital. Virtual Networks are similar to computer networks, except the components of a Virtual Network are in software rather than hardware. With the constant threat of attacks on networks, security is always a big concern, and Virtual Networks are no different. Virtual Networks have many potential attack vectors similar to physical networks, making research into Virtual Network security of great importance. Virtual Networks, since they are composed of virtualized network components, have the ability to dynamically change topologies. In this paper, we explore Virtual Networks and their ability to quickly shift their network topology. We investigate the potential use of this flexibility to protect network resources and defend against malicious activities.

To show the ability of reactively shifting a Virtual Network's topology to secure a network, we create a set of four experiments, each with a different dynamic topology shift, or "dynamic defense". These four groups of experiments are called the Server Protection, Isolated Subnet, Distributed Port Group, and Standard Port Group experiments. The Server Protection experiments involve detecting an attack against a server and shifting the server behind a protected subnet. The other three sets of experiments, called Attacker Prevention experiments, involve detecting a malicious node in the internal network and initiating a dynamic defense to move the attacker behind a protected subnet. Each Attacker Prevention experiment utilizes a different dynamic defense to prevent the malicious node from attacking the rest of the Virtual Network. For each experiment, we run 6 different network attacks to validate the effectiveness of the dynamic defenses. The network attacks utilized for each experiment are ICMP Flooding, TCP Syn Flooding, Smurf attack, ARP

Spoofing, DNS Spoofing, and NMAP Scanning. Our validation shows that our dynamic defenses, outside of the standard port group, are very effective in stopping each attack, consistently lowering the attacks' success rate significantly. The Standard Port Group was the one dynamic defense that is ineffective, though there are also a couple of experiments that could benefit from being run with more attackers and with different situations to fully understand the effectiveness of the defenses. We believe that, as Virtual Networks become more common and utilized outside of data centers, the ability to dynamically shift topology can be used for network security purposes.

ACKNOWLEDGMENTS

Thanks to:

- Tissa, for guiding me through this project and for giving me all the help and resources I needed to complete my thesis. I could not have accomplished this without your guidance.
- Dr. DeBruhl, for advising me on my project and pushing me in the right direction when I came to you for help. You were always helpful and enthusiastic in your assistance.
- Dr. Bellardo, for all you taught me on the thesis process and all your help with all aspects of my masters degree.
- Dr. Keen for being on my committee and for finding interest in my thesis.
- My loving family, Aba, Ema, Shawn, Ashley, and Milli, for being so supportive and giving me encouragement to complete my work.
- Jess, for always being there for me and getting me through the ups and downs of my thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Networks are an integral part of our society today. We rely on them to exchange data around the world in the blink of an eye. However, networks were not originally designed with much security in mind. Many network protocols relied on people having good intentions and following the rules. We now know that this is not realistic, as there are many malicious users throughout the world's largest network, the Internet. Security is a huge challenge in networks because there are many protocols that were designed with no security. With the vast resources many malicious hackers have and the many possible points of entry, it is nearly impossible to stop all possible attacks. Even with the difficulty of preventing every threat, proper security for a network can make it very hard on an attacker. While traditional networks were thought out and built many years ago, Virtual Networks are a newer concept. Virtual Networks are similar to Virtual Machines (VMs), in that they are built purely in software, but still rely on hardware. Virtual Networks perform the functionality of hardware routers and switches inside VMs, allowing for multiple virtual routers or switches on the same physical host [45]. Because there can be multiple virtual routers/switches on a physical machine, Virtual Networks can physically overlap while being logically separate. Virtual Networks also allow for a logical overlapping of physically separate networks. Virtual Networks also have the ability to change network topology on the fly, allowing for greater flexibility than physical networks.

In some ways Virtual Networks improve security and in other ways they open up the possibility of new threats. Being in a sandboxed VM is one advantage that potentially comes with a virtualized network [5]. This lowers the possibility of a virus breaking out of a VM and spreading to other network nodes, since the VM's memory is isolated from the rest of the machine, as we explain in the background

section. At the same time, the communication between the Virtual Network manager and the nodes opens up a new attack space. However, despite the new security advantages and disadvantages that come with Virtual Networks, the majority of their vulnerabilities are similar to those of physical networks [62]. There are countless numbers of possible network attacks, and that is no different for Virtual Networks. So while Virtual Networks do open up new opportunities for attacks, the majority of vulnerabilities for traditional networks still apply [5].

To the best of the author's knowledge, the ability for Virtual Networks to change their topology is currently not used with security in mind. It is more utilized for its flexibility in changing to a topology that fits the networks needs at a given moment and for a break in case of emergency when a node or link fails. However, with proper identification of which topologies are more vulnerable and which are more secure against certain network attacks, a Virtual Network's topology could be changed to defend against these attacks. One could argue that this is unnecessary, as patches could be used to protect against attacks. However, an attack might exploit a vulnerability without a patch or is not leveraging any vulnerabilities, such as denial of service. Changing topology could prevent or slow down the target while certain components are being patched.

A system which would allow for the transformation of a network from one topology to another could greatly improve the security of that network. Virtual networks allow for this transformation, unlocking the ability to always be in the best topology to defend against an impending attack or an attack that is already underway. Traditional networks make it very hard to reconfigure the network topology, since it would entail physically changing and moving connections. The routers and switches would also have to relearn the topology through their MAC and IP tables.

There have been numerous studies, experiments, and evaluations on the different security aspects of traditional networks and even Software-Defined Networks. However, there has not been as much research into the security aspects of Virtual Networks. Virtual networks allow for many things that traditional networks don't, such as creating multiple overlapping (but logically separate) networks and dynamic topology changes. My contribution to the field of Virtual Networks will be a set of experiments that measures the impact of using dynamic security defenses in Virtual Networks to defend against network attacks. To do this, we design a set of dynamic defenses and network attacks. Dynamic defenses involve shifting the topology of the network by moving a VM to a different subnet or physical machine. There are four types of dynamic defenses: Server Protection, Isolated Subnet, Distributed Port Group, and Standard Port Group. These defenses use virtual network security functions such as a distributed firewall, spoof guard, or an isolated/protected subnet (black hole). To test the effectiveness of these topology shifts, we use 6 network attacks: TCP Syn DOS, ICMP DOS, Smurf attack, ARP Poisoning/Spoofing, DNS Poisoning/Spoofing, and NMAP Scanning. We test the success of the attacks before and after the dynamic defenses have been initiated to determine the effectiveness of each. Our experiments show that the success rate of each attack drops dramatically once the dynamic defenses take place and either the victim is more heavily protected or the attacker is more isolated. With that knowledge, ideally future security mechanisms for Virtual Networks can utilize dynamic topology shifts in response to and to prevent network attacks. While the defenses may not seem necessary, they could allow protection while waiting for an admin to take action

The rest of this paper is structured as follows. Chapter 2 gives background on network attacks, Virtual Networks, and Virtual Network Architectures. Chapter 3 discuss our system design, describing how our Virtual Network environment is

created and set up. Chapter 4 discuss our experimental design, giving details on how we test the validity of our system. Chapter 5 looks at the results of the experiments and analyzes them. Chapter 6 is related works. Chapter 7 concludes the paper with a brief recap of our system and results.

Chapter 2

BACKGROUND

To fully understand the set of experiments run, some background into network security and network virtualization is needed. In this chapter, we go over the concepts used throughout this paper and explain how they work. In our experiments, we run network attacks to test our dynamic defenses. To understand these attacks, we give background in section 2.1 on network attacks, specifically denial of service and man in the middle attacks. We also utilize virtualization of network and security components, and nested virtualization, which we explain in section 2.2. Finally, in section 2.3, we describe the different Virtual Network Architectures that exist and how they work. This helps explain the system we use for our experimentation and why we selected it. We begin with a background of network attacks.

2.1 Network Attacks

There are many different ways to attack a network. Many of the existing network protocols were designed without much security in mind, making for many network vulnerabilities. The vast scale of devices communicating with each other and the vast number of subnets make it impossible to secure every area of a network. Limited resources generally force admins, especially in large companies, to choose how to most effectively disperse their resources. Even with the right tools to secure a network, it is up to the network admins to use them properly. Any improper use, whether it be accidental or due to lack of knowledge, opens the door for attackers to take advantage of some vulnerability. Even with everything in the network set up perfectly, there is the possibility of some existing vulnerability in a device or even of a zero-day attack. On top of that, network admins must balance security with usability and ease of use. It would be most secure for a network to allow no

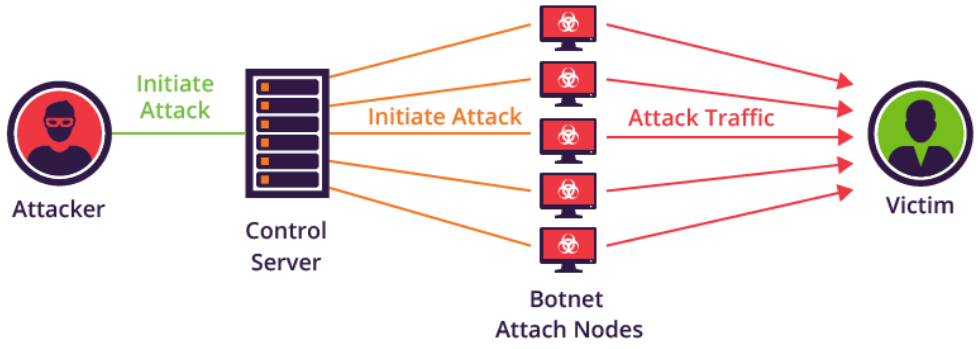

Figure 1: Example of a Simple Distributed Denial of Service System

communication outside of its internal network, but this is infeasible as devices need to communicate with the outside world. In this section, we provide background on different network attacks related to the ones used in our experiments. We divide this section into two parts: Denial of Service/Distributed Denial of Service and Man in the Middle attacks. For each category, we provide some background on the different attacks, discuss how to detect them and what countermeasures exist, and list the tools that can be used to carry them out. We start by looking at Denial of Service and Distributed Denial of Service attacks, or DOS/DDOS attacks.

2.1.1 Denial of Service (DOS) and Distributed Denial of Service (DDOS)

Denial of Service is an attempt to make network resources, services, or hosts unavailable to its intended users, such as temporarily or indefinitely suspending services to users [23]. DOS attacks generally target servers or machines servicing multiple users, but could be used to target and prevent an individual user from accessing some resource. DOS attacks are one of the most common types of attacks due to the large number of different ways to carry them out [23]. They also are generally very effective, especially against systems that aren't properly secured.

A variation of Denial of Service is Distributed Denial of Service. DDOS is similar to DOS, but utilizes multiple hosts. Both DOS and DDOS have the same

end goal, but DDOS uses multiple hosts to accomplish denial of service rather than launching the attack from one host [10]. Figure 1 shows an example of a DDOS attack. DDOS attacks involve a master and multiple zombies. The master is the controller that communicates with the zombies and gives them commands, such as when and who to attack [12]. Zombies are infected hosts who communicate with the master and carry out attacks [12]. These zombies are infected with malware that searches for other vulnerable hosts to infect and waits to be given commands by its master. DDOS attacks have the potential for much greater disruption due to its distributed nature [10]. Having many more hosts carrying out attacks means more power and more ability to use up a resource to the point of denial of service. It also makes it much harder to find the original source of the attacks because the workload is divided between potentially many hosts, and the source of the attacks is abstracted by its zombies [10]. Detection of these attacks is becoming harder as these DDOS systems are increasingly reliant on shorter bursts, with more packets per second spread across different sources [10].

In this section we look at the types and subtypes of DOS and DDOS attacks, how to detect them, possible countermeasures, and what tools can be used. We start by describing the different DOS/DDOS attack types.

2.1.1.1 Types

There are many different types of DOS and DDOS attacks, and even the different subtypes of DOS attacks can be divided up into more attack subtypes. Because of this we won't list every single existing attack or go into too much detail on each attack, but rather give a general overview of each attack and its subtypes.

1. Flooding:

   Flooding is one of the most common ways to perform a DOS/DDOS attack. A flooding attack is when an attacker sends the target so much data that it either

slows down or stops the target's services completely [43]. This can come either through saturating the target's links, using up some resource on the host, such as memory, or through overtaxing the target by giving it more data than it can process [43]. There are many types of flooding attacks, all taking advantage of different, common network protocols.

(a) UDP Flooding:

This is a simple flooding attack that involves sending many, very large UDP packets, either to use up some processing resource or to use up the victim links' bandwidth [43]. As with most flooding attacks, the source can be abstracted by spoofing the source IP address or by distributing the work among infected hosts [43]. This is one of the less resource intensive flooding attacks to run for an attacker.

(b) Ping/ICMP Flooding:

Ping flooding is similar to UDP flooding, except with ping (ICMP) packets [43]. Like UDP flooding, the goal is to use up the bandwidth of the victim's links.

(c) TCP Syn:

Rather than use up links' bandwidth, this attack uses up a victim's TCP resources [43]. The attacker floods the victim server with TCP Syn requests to create new connections. This could lead to outcomes such as preventing new TCP connections or dropping current, valid TCP connections [43]. Either outcome would result in partial or full Denial of Service for many users to whatever service the victim is providing.

(d) Reflection Attack:

A Reflection attack, or Reflective flooding, is a way to perform a DDOS attack without having to infect other hosts. Reflective attacks involve

sending network protocol packets that require a response from other hosts. The attacker spoofs the source address of these packets as the victim's IP address and then sends many packets to many different hosts [43]. These hosts then receive the packets and all send reply packets to the victim, since its IP address is given as the source. This leads to a flood of packets directed at the victim host, potentially causing a Denial of Service [43]. The source of this type of attack is better abstracted than other types of flooding attacks due to its distributed nature and the spoofed source IP address. There are many different types of Reflective Flooding attacks. The most common protocols used in Reflection attacks are NTP, DNS, CharGen, SSDP and RIPv1 [19]. However, one of the most famous examples of a Reflection Attack is the Smurf attack.

i. Smurf Attack:

A Smurf Attack is when an attacker performs a Reflective attack with ICMP echo requests [43]. It was given the name Smurf in reference to the cartoon 'The Smurfs' [43]. Smurfs are small, blue people who individually are small and weak, but collectively could overwhelm a stronger opponent. The attacker sends many ICMP echo packets with the source IP address spoofed as the victim's IP address, causing the victim to receive large amounts of ICMP responses [43]. This potentially saturates the victim's links and causes a Denial of Service.

2. Application/Layer 7 Attack:

This form of DOS involves any use of application layer protocols such as DNS or HTTP to cause a Denial of Service [23]. This attack may use flooding techniques or could exhaust a host's resources such as sockets, memory, etc. [23]. Application attacks are more dangerous than certain flooding attacks as they are

more difficult to detect. This is because requests coming from an attacker are almost indistinguishable from requests coming from authenticated users [23]. The attack is more likely to pass through firewalls undetected, because defense mechanisms based on TCP/IP authentication won't catch application level attacks, since they use valid TCP/IP headers [23].

(a) DNS Amplification Attack:

A DNS Amplification Attack is an attack that uses DNS queries to perform a Reflective Flooding attack, similar to the Smurf Attack [51]. The attacker sends many DNS queries with the source IP address spoofed with the victim's IP address. This causes DNS servers to send many DNS responses to the victim host, potentially causing a Denial of Service [51].

3. Denial of Sleep Attack:

A Denial of Sleep attack involves draining a host's energy to cause a Denial of Service [31]. When the victim host's receiver is in inactive mode (conserving energy) the attacker starts bombarding the host with packets to drain its energy [31]. The attacker only sends packets when the victim's receiver is in inactive mode. This attack requires detailed knowledge of the victim host's MAC layer protocol [31].

2.1.1.2 Detection and Countermeasures

There are many different ways to detect and react to DOS/DDOS attacks. Here we list the different detection methods and countermeasures. We give basic explanations of how each method works without explaining the math behind it, as this would become too complex. We start with the detection methods.

1. Detection Methods

(a) Anomaly-Based Detection:

Anomaly-Based detection involves monitoring the behavior of live traffic [23]. Traffic data is compared to a baseline of how normal, non-malicious traffic is expected to behave. To do this, a model is built with training and testing data of what is considered normal and what is considered malicious. A large majority of current DOS detection is done this way and many IDS systems use this type of detection [23]. Some examples include Fast Entropy Detection, Fuzzy Estimator and Multivariate Correlation Analysis (MCA). This technique is beneficial because after it is trained it doesn't require large amounts of memory. However, it does have the potential for inaccuracies, especially if the process of training the model is done incorrectly. It also requires a large amount of time to train the model initially [23].

(b) Data Mining Detection

Data Mining Detection is a signature-based form of DOS Detection [23]. A number of attack signatures are stored in a database and traffic signatures are compared to those in the database [23]. This requires no time spent on training a model and thus can be used for immediate detection. However, depending on the number of signatures in the database, it could use a large amount of memory. It also could take a large amount of time to match the signature in the database, depending on the number of signatures stored and the method for searching [23].

2. Countermeasures

(a) Access Control Lists (ACLs):

An Access Control List is a list of rules that say what network behavior is allowed and not allowed [43]. ACLs can accept, drop, or deny traffic based

on a pre-determined list of patterns [43]. For example, one pattern could say that any host from the subnet 192.168.144.0/24 cannot send HTTP traffic. ACLs allows for potentially malicious data to be dropped, preventing an attack from occuring. An example of an ACL is the command line tool "iptables".

(b) Rate Limiting:

Rate Limiting is a countermeasure that uses a threshold to determine when to start dropping packets [43]. If some pre-determined data rate threshold is reached by a host, group of hosts, or subnet, their packets are dropped. Rate limiting, in combination with ACLs, are used by firewalls to prevent a wide range of attacks, not just DOS/DDOS attacks [43].

(c) Amplification Honeypot:

An Amplification Honeypot is a honeypot specifically designed to prevent reflection and amplification DOS attacks [19]. To prevent these attacks, the honeypot emulates protocols that are generally abused by those DOS attacks, making it an appealing target to attackers [19]. Attackers are then likely to target the honeypot because they detect it as vulnerable, exposing themselves as malicious and wasting their resources on a trap.

2.1.1.3 Tools

There are a few tools that can be used to carry out DOS attacks. DOS tools generally aren't overly complex, as many of the types of Denial of Service attacks perform similar actions and don't require large amounts of code. We list here a few of these tools and give a brief summary of each.

1. Hping3:

Hping3 is a tool in the Kali Linux OS [43]. It allows for the user to send different types of flooding attacks such as TCP Syn flooding, UDP flooding, etc. It also allows for further modification through setting sending intervals, spoofing of source IP address, and many other options [43]. Hping3 is simple and quick, but lacks the power needed for Distributed DOS attacks.

2. Hyenae packet generator:

   Hyenae is a tool that allows for various DOS and Man in the Middle attacks [43]. Hyenae is more sophisticated than Hping3, allowing for more complex forging of packets, running attacks through multiple zombie hosts (DDOS), and triggering remote attacks from one master host [43].

3. Netflow/IPtraf:

   Netflow and IPtraf are Cisco tools used to detect traffic flows [43]. They can be used for the detection of DOS based attacks or for gathering information to be used in launching DOS attacks.

4. Netstat:

   Netstat is a tool that is used to list all the network connections of a machine [43]. It can be used to see the ingoing and outgoing connection, their types, and more detailed information [43]. Netstat can be used to gather information to determine whether a DOS/DDOS attack is being launched or to gather information to be used in launching a DOS attack.

2.1.2 Man in the Middle (MITM)

A Man in the Middle (MITM) attack is an attack where a "malicious third party secretly takes control of the communication channel between two or more endpoints" [11]. The goal for the attack is to be able to either eavesdrop or

manipulate information being shared between two parties. This could have many end goals, such as stealing login info or even denial of service. There is also the possibility of intercepting encryption keys if they are sent insecurely [11]. The keys could then be used to make a connection seem secure for both parties, even though the attacker would have access to all the data being exchanged. Once an attacker has access to a channel of communication, they can perform many different attacks, such as intercepting, dropping, modifying, or replacing packets being sent between two parties [11]. In this section, we describe the different types of MITM attacks, discuss possible countermeasures, and list the tools that can be used for it. We start with describing different types of MITM attacks.

2.1.2.1 Types

There are many different Man in the Middle attack types and subtypes. We don't have space to list them all here, so we don't describe every type. For each, we give a brief description of the attack and list the attack subtypes. We start with spoofing and poisoning attacks.

1. Spoofing and Poisoning:

   A Spoofing Man in the Middle attack is when an attacker intercepts communication through the use of spoofing [11]. This spoofing allows the attacker to control any data transferred through the intercepted communication, without the other parties having any knowledge. Spoofing can come in a few different forms, such as an attacker pretending to be a device between the victims or pretending to be the another party and spoofing the victim directly [11]. There are many different forms of spoofing, each accomplishing a similar goal in different ways.

   (a) ARP Spoofing/Poisoning:

ARP Spoofing is when an attacker modifies a victim's local ARP cache [11]. The attacker associates their MAC address with some other device's IP address, causing messages sent from the victim to the spoofed device to be sent to the attacker instead [11]. Generally, the device who's IP address is associated with the attacker's MAC is a subnet gateway, allowing the attacker to gain all information the victim is sending outside their subnet. To make this a true Man in the Middle attack, the attacker must also compromise the spoofed device's ARP cache to allow for all data flowing to and from the victim to be intercepted [11]. This attack can also be utilized to cause Denial of Service.

(b) DNS Spoofing/Poisoning:

DNS Spoofing is when an attacker tries to send a victim a malicious DNS packet containing the wrong IP address for a queried url [39]. The goal is to give the victim an IP address controlled by the attacker in reply to a DNS query. This causes the victim to connect to the attacker's IP address when visiting the queried url. The attack is then completed by forwarding the victim's data to the real website, allowing for the attacker to intercept all traffic going between the two [11]. There are two ways for the attacker to accomplish this attack, DNS cache poisoning and DNS hijacking.

i. DNS Cache Poisoning:

There are two types of DNS cache poisoning. The first is when an attacker inserts a malicious DNS server into a network, causing the victim to send DNS queries to the malicious DNS server [11]. The malicious DNS server can then reply with whatever IP address that attacker wants associated with different urls. This attack is possible because DNS doesn't use any authentication to confirm that a DNS server can be trusted. The second is accomplished through sending a

fake DNS reply before the real DNS server can respond to the victim's query [39]. This is possible because DNS packets are transmitted in plaintext, allowing an attacker to sniff and read any DNS query. This attack is riskier, because it relies on sniffing the query and sending a fake DNS response before the real DNS server does, which is not guaranteed to happen.

ii. DNS Hijacking:

This attack involves sniffing or intercepting DNS packets being sent between the authoritative and recursive DNS servers [11]. The attacker can do this through either of the two DNS cache poisoning attack types to try to feed the authoritative server incorrect DNS information.

(c) DHCP Spoofing:

DHCP Spoofing is an attack where the DHCP protocol is used to achieve a man in the middle through sending false DHCP information. Like DNS, DHCP doesn't have any authentication to create trust in a DHCP server. Also like DNS, messages are transmitted in plaintext, allowing an attacker to sniff and read any DHCP query [11]. There are two ways to accomplish a DHCP Spoofing attack. The first is to try to respond to a DHCP request faster than the real DHCP server [39]. The second is to set up a fake DHCP server to send fake information to the victims. These both allow the attacker to send incorrect information on the gateway address, the DNS server, or even the victim's own IP address [39]. Any of this information would then allow the attacker to intercept the victim's data and perform a MITM attack.

(d) IP Spoofing:

IP spoofing is when an attacker intercepts communication between two parties [11]. This can then be used to view all data sent through this

communication. There are a few ways to intercept communications: blind and non-blind spoofing, ICMP spoofing, and TCP hijacking.

i. Blind/Non-Blind Spoofing:

   Both blind and non-blind spoofing can be used for data mining to intercept communication, but can also be used for DDOS attacks [11]. In blind spoofing, the attacker has to send requests to the victim's network and can then analyze the transmission sequence [11]. In non-blind spoofing, the attacker is in the same subnet as the victim, allowing the attacker to sniff data such as sequence and acknowledgement numbers [11].

ii. ICMP Spoofing:

   Since ICMP has no authentication mechanism, the ICMP redirect packet can be used to route a victim's messages through a malicious router [39]. The malicious router can then eavesdrop, modify, or drop any packets.

iii. TCP Hijacking:

   For TCP hijacking, the attacker either guesses or uses knowledge of a server's psuedo-number generator to predict a sequence number of an existing TCP connection [11]. If one of the sequence numbers is guessed correctly, the attacker takes control of the connection from the victim.

2. SSL/TLS MITM

   SSL/TLS allows authentication of users and servers to prevent malicious machines from spoofing [11]. It encrypts data to prevent any sniffing of information or modification of packets. It also ensures data integrity so that even if an attacker is able to modify data, it will be obvious that a change was made and the packet will be dropped. All of that allows SSL/TLS to secure

communication between two parties [11]. To intercept communication secured by SSL/TLS, more complex attacks are necessary.

(a) Forge Certificate:

This attack involves using a forged certificate to create a man in the middle between two parties using an SSL/TLS connection [11]. If the attacker forges a certificate, they create an invalid cert and intercept the initial exchange between the client and the server. The attacker then sends the invalid certificate to the client and hopes the user ignores the cert warning [11]. If the user accepts, the attacker can create a second connection with the server and intercept data between the client and the server.

(b) Hijack Certificate

This attack involves using a hijacked certificate to create a Man in the Middle between two parties using an SSL/TLS connection [11]. In this attack, an attacker somehow obtains a valid certificate to a server or a private key through malicious means [11]. The attacker could then create a connection with the client with no cert warnings, since the certificate is valid. The attacker could then create a second connection between itself and server, acting as a man in the middle between the client and server.

2.1.2.2 Countermeasures

Detection and prevention of Man in the Middle attacks are much more specific to each type of attack than DOS/DDOS detection and prevention. There are many more specific solutions rather than general ones, so we don't list all the possible methods but rather list a few brief examples for each attack. In this section, we list the different detection methods and countermeasures for each MITM attack.

We give a brief description of each countermeasure without going into too much detail. We start with measures for detection and countermeasures for ARP spoofing.

1. ARP Spoofing/Poisoning:

   There are a few solutions to detect ARP spoofing. ARP-Guard and ARP-Defender use network architectures utilizing sensors to detect the attack [11]. ARPwatch tracks MAC-IP pairings to be able to detect if any pairings are changed [11]. However, it only alerts admins if a suspicious change has been made and doesn't take any action itself. There are also cryptographic solutions, such as S-ARP which authenticates ARP replies using public key cryptography [11]. There is the voting based solution MR-ARP, which queries neighboring machines to vote on new IP-MAC pairings [11]. There are hardware solutions such as Dynamic ARP Inspection, which ensures the validity of ARP requests and responses [11]. There is Antidote, a server-based solution, which gives priority for an IP-MAC pairing to old IP addresses [11].

2. DNS Spoofing/Poisoning:

   Cache Poisoning detection system is a system that uses machine learning to detect DNS spoofing in real-time [11]. There is a P2P system that, similar to the MR-ARP, uses a voting system to authenticate a DNS response [11]. There are cryptographic solutions similar to ARP crypto defenses, such as DNSSEC which uses cryptographic signatures to achieve data integrity and origin authentication [11]. There is also an Artificial Neural Network solution that can predict the reliability of DNS response packets [11].

3. DHCP Spoofing:

   DHCP detection is mostly cryptographically based, such as SDSS and SDDC [11]. There is also a hardware solution called DHCP snooping that acts as a firewall between hosts and DHCP servers [11].

4. IP Spoofing:

   IPSec and Ingress Filtering are the two main methods that have been used to prevent IP spoofing [11]. Ingress filtering filters packets at the border of an Autonomous System and can use either Access Control Lists (ACLs) or unicast Reverse Packet Forwarding [11]. There are router based solutions such as Distributed Packet Filtering, Source Address Validation Enforcement, and Spoofing Prevention Method. Distributed Packet Filtering filters based on packet flow and interface used [11]. Spoofing Prevention Method inserts a tag into packets that validates AS/prefix pairings [11]. Hop Count Filtering, a host based solution, validates prefix and hop count pairings [11]. There is also the cryptographic based Accountable Internet Protocol that validates correctness of addresses [11].

5. SSL/TLS MITM:

   There are many different ways to detect this attack. Detection of forged certificates can be used, with Cert Transparency, ICSI, and Crossbear all doing this. Crossbear actively scans the Internet to detect forged certificates [11]. Cert pinning can also be used, where hosts are associated with certificates or public keys [11]. Multi-path probing uses a distributed voting approach to authenticate certificates [11]. Also, forcing SSL/TLS connection can be done to prevent downgrade attacks, where the attacker makes the client or server not use SSL/TLS [11]. There are also friendly MITM defenses that are used to secure every new connection created [11].

2.1.2.3 Tools

Unlike DOS/DDOS attacks, MITM tools are generally more catered to a specific MITM attack. There are many available MITM tools for each attack. We list a few tools and give brief descriptions of each.

1. Ettercap

   Ettercap is a suite of tools that have the capability to do many different Man in the Middle attacks. It can be used for ARP poisoning/spoofing and DNS poisoning/spoofing [39]. It also has tools for hijacking connections, filtering traffic, and sniffing SSH connections [39].

2. Dsniff

   Dsniff, like Ettercap, is a suite of tools that can run different Man in the Middle attacks. Like Ettercap, it can be used for ARP spoofing, DNS spoofing, and sniffing [39].

3. Zodiac

   Zodiac is a tool specifically used for DNS spoofing [39].

4. IRPAS icmp_redirect and icmp_redir

   Both these tools are used specifically to pull off an ICMP redirect attack to redirect a victim's traffic through a malicious machine [39]. IRPAS can also be used for route mangling, where an attacker presents the best route as one that goes through a malicious machine [39].

5. Nemesis

   Like IRPAS, this tool can be used for route mangling [39].

2.2 Virtual Networks

Physical networks have the problem of lack of flexibility, both in the difficulty of adding new services, such as firewalls and IDSs, to existing networks and in moving those services around the network. The idea of Virtual Networks were suggested as a solution to this lack of flexibility [17]. Virtual Networks are the decoupling of network hardware and software, so that network services run purely in software on top of the physical network [20]. When it was first suggested, the idea was Virtual Networks could potentially allow for the ability to seamlessly add new services, hosts, routers, etc., while also being able to move them around at will. Today, we see that this is true. The decoupling of network hardware and software allows applications and VMs to be connected to each other by a Virtual Network as if it were a physical one [59]. Virtual Networks are also sometimes referred to as Network Function Virtualization (NFV), which is the virtualization of network components [29]. Virtual Networking is not to be confused with Software-Defined Networking (SDN), which is the decoupling of the data and control planes of a network [29]. In this section we give background on how Virtual Networks work and how they are used. In section 2.2.1, we begin with a brief discussion on virtualization. Then, in section 2.2.2, we discuss how virtualization is applied to networks and how Virtual Networks work. We then discuss nested virtualization in section 2.2.3, as it is utilized in our system design. Finally, in section 2.2.4, we discuss the state of security in virtual networks.

2.2.1 Virtualization

Virtualization is the simulation of hardware function in software to allow for the creation of a virtual computer system or virtual machines [59]. This allows for the hardware resources to be divided up among virtual machines so that multiple

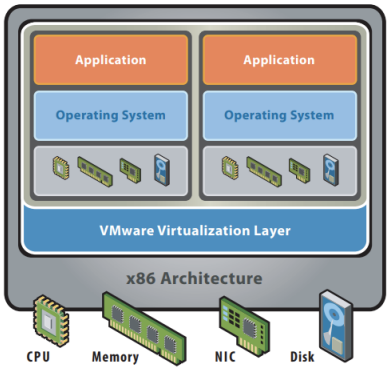

Figure 2: After Virtualization [54]

computers or servers can be running on top of one physical machine. This creates a benefit of flexibility and efficiency [59]. In this subsection we first describe conceptually what virtualization is and then we go into basic details on how virtualization is achieved.

Virtualization has many properties. Virtualization allows for different operating systems to run on the same machine, even if they are not compatible with each other. This is called partitioning and also includes the ability to divide the physical machine's resources among the VMs [59]. Virtualization also provides isolation, where despite being on the same hardware, a VM's memory can't be accessed by anyone other than the VM itself. Isolation also means "preserving the performance of a VM with advanced resource control tools" [59]. Virtualization also utilizes encapsulation, where the entire VM is saved as files and can be moved and cloned very easily, similar to moving files [59]. Lastly, virtualization brings hardware independence, where any OS can be used on any type of hardware, allowing for great flexibility [59].

So how does this division of resources actually work? To allow for VMs to share resources on a system, a special OS is needed to communicate with the host machine to get access to the hardware resources and divide them among the VMs. This OS is typically called a hypervisor. On top of the hypervisor are the VMs'

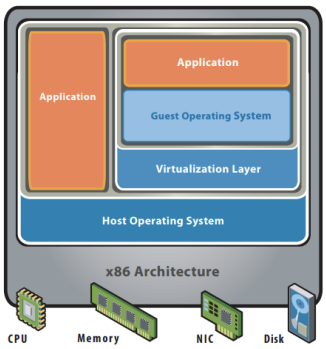

Figure 3: Hosted Virtualization [54]

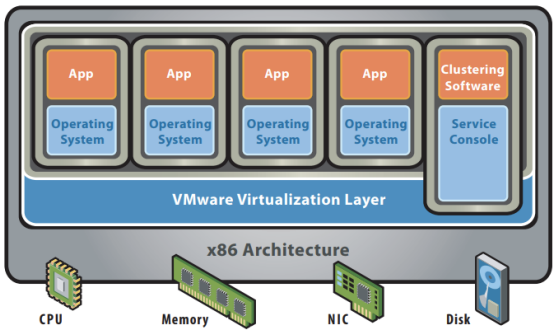

Figure 4: Hypervisor Virtualization [54]

operating systems, which communicate with the hypervisor to gain access to resources [54]. When a VM's OS requests a resource from the hypervisor, the hypervisor communicates with the host system to get access to the resource. Figure 2 shows an example host with virtualized machines on top of it. There are two main approaches to where the hypervisor is placed, hosted and hypervisor [54]. They are both very similar, with the key difference being that in the hosted approach the physical machine's OS remains, while in the hypervisor (also called bare metal) approach, the host OS is replaced with a hypervisor [54]. The hypervisor approach is more efficient, since the abstraction layer (the hypervisor) has direct access to hardware resources, but the hosted approach allows the most hardware configurations [54]. Figure 3 shows a hosted virtualization and figure 4 shows a hypervisor virtualization. Because a hosted approach leaves the original OS, the machine can have both VMs and other applications running simultaneously. Due to abstraction, VMs see their resources as physical, not knowing that they are communicating with a hypervisor to get them [54]. This allows for VMs to function like normal machines.

2.2.2 Network Virtualization

Network Virtualization works similar to machine virtualization, utilizing hypervisors to abstract hardware and implementing hardware components in software. Think of Virtual Networks as similar to a physical network but in software, just like how a VM is similar to a computer but in software. In this subsection we describe how Virtual Networks work and how they are used.

Virtual Networks generally are divided into two parts, the overlay and the underlay networks. The underlay network is made up of the underlying physical network components that communicate with each other physically, while the overlay network is the Virtual Network with the virtualized network components and

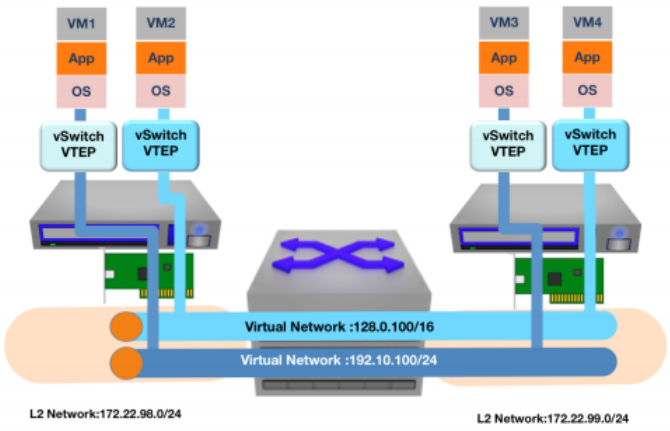

Figure 5: Simple Virtual Network [45]

VMs [21]. Figure 5 shows an example, basic Virtual Network. Network components such as routers and switches are emulated in software to provide the layer 2 and layer 3 functionality of physical networks. For example, a virtual switch (vSwitch) has the same packet forwarding of a physical switch, but in software [45]. Similarly, NICs are also emulated in software as vNICs, which are used for VMs. In figure 5, the two machines each host 2 VMs and two vSwitches, with one of each belonging to separate networks. Similar to a VM's ability to preserve isolation and security on the same physical machine, Virtual networks allow for VMs to be on separate networks despite being on the same physical host or subnet [45]. They also allow for VMs to be on the same network despite being physically far from each other.

To allow for VMs to be logically on different networks, something must be done to prevent VMs and physical machines from looking at packets that aren't a part of their network. The solution is VXLAN, which acts similarly to VPN. VXLAN is a MAC over IP encapsulation [21]. It is used to create virtual overlays (or tunnels as it is called in VPN) across physical networks [45], preventing components, both physical and virtual, from viewing packets that dont belong to their virtual network. The vSwitches connected to the VMs in the virtual network

act as virtual tunnel endpoints (VTEP) between NICs [45]. These VTEPs allow for both physical and virtual components to create routes to send data between VMs. VXLAN reduces the complexity of forwarding, allowing for routes to be created based on the overlay network and having the physical network components simply forward VXLAN packets to the correct VTEP [45]. With VXLAN, whole datacenters can be virtualized [45].

So how are Virtual Networks used? Generally Virtual Networks are utilized in data centers and service provider networks [45]. This allows them to create networks based on certain application needs and then change the network topology on the fly when the needs change. For now, those are the main use cases for Virtual Networks. However, there have been other proposed applications such as in mobile and home networks [17]. While Virtual Networks aren't currently being used for those applications, it is possible that in the future they will be.

2.2.3 Nested Virtualization

Creating a Virtual Network environment requires a full, underlying physical network. But what happens if there is a lack of physical resources? For our system, we need to create a Virtual Network environment that is contained within one physical host. Despite the lack of physical network devices, this can be accomplished with nested virtualization. Nested virtualization is when a hypervisor runs one or more other hypervisors, each of which can run their own VMs [7]. Nested virtualization can have many uses, such as the ability of users to select their own hypervisor and VM combination, or to allow hypervisors and VMs to be migrated as a unit for load balancing or disaster recovery [7].

Hardware-assisted nested virtualization is a common way to achieve this. Figure 6 shows an example of hardware-assisted virtualization and the three levels of nested virtualization. In this figure, VMM is the hypervisor and L0 stands for the

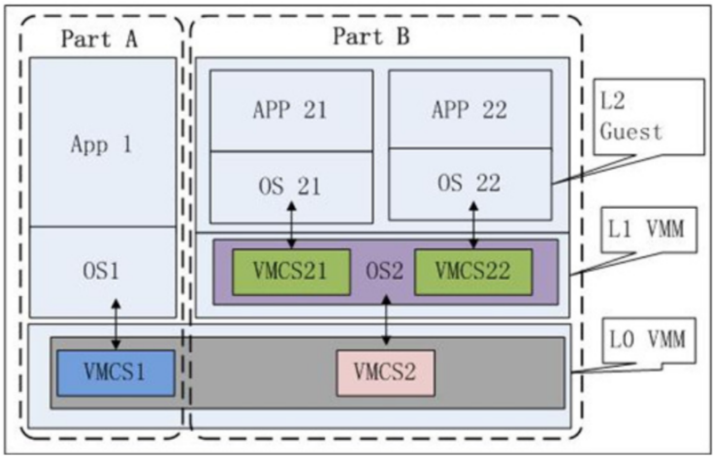

Figure 6: Hardware-Assisted Nested Virtualization Diagram from Nestcloud [40]

first level and so on. In normal virtualization the hypervisor runs in host mode, having direct access to hardware resources, while the VMs run in guest mode [64]. When a VM tries to execute a privileged instruction that accesses hardware resources, a control transition takes place from guest mode (VM) to host mode (hypervisor), with the hypervisor executing the VM's command for them. Once the command is completed, the hypervisor relinquishes control back to the VM [64]. In hardware-assisted virtualization, there is a second layer of virtualization. This second layer makes both the nested hypervisor and the VMs operate in guest mode (in separate contexts), while the underlying hypervisor is in host mode [64]. The operations work similarly to normal virtualization, with the underlying hypervisor acting as a middle man between the nested hypervisor and the VM. When a VM requests a privileged hardware resource, a control transition takes place from the VM to the underlying hypervisor. The underlying hypervisor then sends the request directly to the nested hypervisor to process. Once the nested hypervisor has processed the request, it sends the event to the underlying hypervisor to execute. Once the event is complete, the underlying hypervisor relinquishes control back to the VM [64]. This process allows for nested hypervisors to behave logically like a

normal hypervisor on top of a physical host, when in reality the nested hypervisor is dependent on another, underlying hypervisor. Using this technology, we are able to use the nested hypervisors to act as physical hosts in our system.

2.2.4 Virtual Network Security

Virtualization changes the vulnerabilities of Virtual Networks as compared to physical networks. While virtualization adds some protection, it also adds new attack vectors, such as multiple Virtual Networks being able to share memory and link resources [5]. It is important to realize what is gained and lost security wise in respect to virtualization and Virtual Networks. However, overall, most of the vulnerabilities of physical networks apply to Virtual Networks as well. In this section, we look at security related to virtualization and Virtual Networks. Specifically, we look at how security is changed, both positively and negatively, with virtualization and how that affects Virtual Networks. We start with describing virtualization security.

2.2.4.1 Virtualization Security

Virtualization allows for confidentiality and integrity of data, separating VMs from each other despite being on the same hardware. However, there have been cases of malware either breaking out of or in to a VM. There are two main ways for an attacker to attack a VM: getting access to the VM from the host or getting access to the host from the VM [62]. The first one is done through infecting the host machine that the victim VMs are on. The attacker can then attempt to breach VMs or create their own malicious VMs and launch them [62]. An example of this is the W32.Crisis. This was an attack where malware infected a host machine as a java file, which made its way onto the machine through social engineering. It then utilized the ability to manipulate VMs through their stored files, adding itself

to the VM so that the next time the VM was powered on, it contained the malware [62]. The second threat is breaking out of a VM and getting access to the underlying host. This is more dangerous, as breaking out of a VM could allow malware to spread rapidly. It is also of great importance to prevent, as VMs are sometimes used to inspect malware [62]. Cloudburst was an attack where malware in VMs was able to execute commands on the underlying host [62]. This was accomplished by using invalid instructions, which created exceptions. These exceptions were then hijacked and then cached on the host machine, allowing code to be created by the malware and executed on the host [62]. There is also the possibility of shared folders being used to get from a VM to a host [62]. The best way to prevent these attacks is to use hardening, malware protection, access control, and other common computer protections [62].

2.2.4.2 Network Virtualization Security

Virtual Networks secure some of the vulnerabilities of physical networks, and at the same time add some new vulnerabilities as well. However, they are just as vulnerable as physical networks. The majority of network attacks work for both physical and virtual networks. In this section, we look at the threats to Virtual Networks and some general countermeasures. We list the categories of each and describe some of the specific attacks and countermeasures briefly. We start with Virtual Network threats and vulnerabilities.

For Network Virtualization, there are four broad categories of threats and vulnerabilities: disclosure, deception, disruption, and usurpation [5]. Disclosure is "unauthorized access to protected information" [5]. Deception is convincing other devices that false information is actually true. Disruption is causing some part of the system or communication to fail. Usurpation is gaining unauthorized control of some part of the network [5]. Each category has further subcategories of

vulnerabilities and threats for Virtual Networks. For each category, we list the different subcategories and their examples, and give a brief description of each.

1. Disclosure

   (a) Information Leakage

   Information leakage is when some information from one Virtual Network is able to be viewed by another, separate Virtual Network [5]. This occurs because some part of the Virtual Network is leaking private information. This attack may be done through ARP Poisoning [9]. If the attacker spoofs an IP address in another Virtual Network, it may be able to gain information that is supposed to be private to that network. Information leakage could also be done by escalating privilege and getting root access to a VM in the other Virtual Network [5].

   (b) Information Interception

   This is also called eavesdropping, where data being sent between two parties is intercepted by a malicious device [5]. Again, ARP poisoning can be used to achieve this attack. The attacker could also trick a physical switch or router into sending data through a path that allows the attacker to sniff all the packets being exchanged [5]. According to [14], some Virtual Network environments may not correctly isolate data between different Virtual Networks, allowing for sniffing of other Virtual Network information. Even if that data is encrypted, the packets can be analyzed to derive some sensitive information [5]. One could also use Virtual Network requests to discover the physical topology of the network and certain routing information [5].

   (c) Introspection Exploitation

Introspection is used to confirm the states of VMs [5]. This can be exploited to disclose information from VMs, since introspection allows external access to information in the Virtual Network.

2. Deception

   (a) Spoofing

   This involves an attacker sending data with false information to make another device believe that the attacker is someone else [5]. Virtual Network environments complicate spoofing prevention because of the need to keep track of multiple, overlapping Virtual Networks and the potential for incompatible security policies over them [5]. An attacker could also potentially remove a node and re-add it to a Virtual Network to get around security mechanisms [5]. The ability to migrate and duplicate nodes further complicates the prevention of spoofing and opens up more attack vectors.

   (b) Replay Attacks

   This is similar to a replay attack that takes place in a physical network. Virtual routers may replay control messages to corrupt certain data [5].

3. Disruption

   (a) Physical Resource Overloading/Failure

   This can lead to virtual nodes crashing, congestion in Virtual Networks, and prevention of tasks in Virtual Networks. An attacker could coordinate so that one area of a Virtual Network requires too many physical resources to split among the rest of the network [5]. It is also possible for one Virtual Network to hog resources, depriving other Virtual Networks [2]. Because of this, enforcing of minimum/maximum resource consumption is vital [5].

DOS attacks on physical resources would also cause denial of service to the parts of the Virtual Network located on the physical machine [5].

4. Usurpation

   (a) Spoofing

   Similar to spoofing under deception, except it can also be used to gain escalated privileges [5].

   (b) Software Vulnerabilities

   As discussed in section 2.2.4.1, an attacker can find vulnerabilities in the hypervisor and use those vulnerabilities to escape a VM and gain access to the underlying physical machine [2]. This could give the attacker the ability to disrupt one or more Virtual Networks as a result [5].

While there are a lot of threats to Virtual Networks, there are also ways to protect the environment from those threats. There are 6 groups of countermeasures, or security services: access control, authentication, data confidentiality, data integrity, nonrepudiation, and availability [50]. Access Control is the same as an Access Control List (ACL) or firewall, which can be used to determine under what circumstances a node can access a resource. Authentication is used to confirm the identity of a device [50]. Data Confidentiality makes sure information is only seen by authorized devices [50]. Data Integrity ensures that any data in the network isn't unknowingly manipulated [50]. Nonrepudiation connects malicious actions with the perpetrator(s) [5]. Availability means that devices always have access to information that they are authorized to see [50]. For each category, we describe the countermeasures that can be used and give a brief description of each. We don't list every countermeasure, as that would be too expansive, but rather only list a few examples for each category.

1. Access Control

(a) Trusted Virtual Domains

[8] is a framework that creates trusted virtual domains between virtual machines. Each virtual domain is isolated from one another, providing security in communications. A domain only allows a VM to join a group if it meets certain criteria.

(b) Sandbox

Sandboxes can be used to prevent malicious (or any) VMs from accessing resources outside of their own [5]. IPsec can be used to create secure, cryptographic communications and X.509 can be use for authentication of VMs [5]. The combination of the 3 can be used to secure a Virtual Network

2. Authentication

(a) Certificate based Authentication

Virtual Networks can use certificates to authenticate nodes joining the network [5]. As stated before, X.509 can be used for authentication, requiring new nodes to request a certificate from the CA. The certificate given to the new node has its IP address in the certificate to prevent spoofing and reuse [5].

(b) Key based Authentication

Keys can also be used to authenticate nodes in Virtual Networks. Similar to physical networks, two devices can exchange keys to verify each other and then use those keys to create encrypted communication [5].

3. Data Confidentiality

(a) Tunneling and Cryptography

As we discussed in section 2.2.2, tunneling protocols such as VPN or VXLAN can be used to create isolated communication between nodes in a

Virtual Network. They also utilize encryption to protect the data portion of a packet from being snooped.

(b) Firewalls and Subnets

Firewalls can be used to prevent communication between Virtual Networks or filter those communications [5]. Each individual Virtual Network can also be bound to a subnet to provide some further security. Firewalls have large use in physical networks and are useful and flexible in Virtual Networks. We discuss the use of distributed virtual firewalls in our system design in section 3.2.

(c) Path Splitting

Path splitting is where multiple paths are used to send information between two nodes in a network [5]. This can be used to partially evade malicious eavesdropping. If encryption is used, this can be helpful in preventing an attacker from deriving meaningful information by analyzing flows.

(d) Limit Introspection

This would limit what the hypervisor could see in each VM [5]. This would potentially protect against exploitations.

4. Data Integrity

(a) Encryption

Encryption can be used, as with pretty much every other category, to keep data integrity [5]. Like with physical networks, cryptography is vital to securing communications in Virtual Networks.

(b) Timestamps

Putting encrypted timestamps with packets prevents replay attacks from taking place [5]. This is also commonly utilized in physical networks.

5. Nonrepudiation

   This is used to tie attacks to the malicious nodes that did them [5]. There is not much currently in this area for Virtual Networks.

6. Availability

   (a) Physical Resource Isolation

   Isolating physical resources would prevent attacks where resources are maliciously used up [2]. If malicious nodes can be prevented from exhausting resources and the system divides the resources fairly, then availability in the Virtual Network will be preserved.

   (b) Virtual Network Resilience

   Virtual Networks also have to be able to work around physical machine or link failures [5]. Backups can be created for certain VMs to ensure availability. Virtual Networks can also work around link failures, quickly sending data through other paths, since the topology is flexible [5].

2.3 Virtual Network *Architectures*

With the storage needs of the world increasing, especially with the growth of cloud computing, data centers are being utilized more than ever. The increasing need of storage has led to more use of virtualization to help intra and inter data center communications. The virtualization of physical networks is used to help achieve this goal, but with the creation of Virtual Networks, systems must be designed to create, manage, and secure them. A Virtual Network Architecture is the system design for creating and maintaining virtual network components and the resulting networks they create. Different companies design different Virtual Network Architectures, with each having potentially different use cases. In designing a Virtual Network Architecture, there are many questions about how different aspects

of the system work. Questions such as how do network nodes communicate with the management system, how are the data and control planes implemented, etc. Most companies creating Virtual Network Architectures have multiple architectures that they create based on different factors. One factor in creating different architectures is the use of different virtual components, creating architectures that operate the same except for their selection of virtual components. For example, VMware has different architectures based on the types of virtual switches and routing mechanisms utilized [55]. Another potential factor in differing architectures is the potential needs and uses of the customer. For example, OpenStack has many different architectures, ranging from classic implementations to high availability implementations, based on the customer's virtual network needs [34]. In this section, we summarize and compare the main architectures of each company, as covering every single architecture is repetitive. These architectures are used for creating and managing Virtual Networks, some with different use cases, but most with the purpose of creating and managing virtualized networks in large data centers. The rest of the section is organized as follows. In subsection 2.3.1, we discuss the Virtual Network Architectures, and how they are built and used. Then in subsection 2.3.2, we compare the architectures based on their strengths and weaknesses in different categories.

2.3.1 Enterprise Architectures

A Virtual Network Architecture dictates how a virtualized network is managed, created, and functions. In this subsection, we look at different architectures, how they work, how they are used, and their trade-offs. The architectures include VMware vSphere architectures [57], including one that utilizes Palo Alto Networks components [60], Juniper Networks' Contrail Architecture [22], OpenStack's Neutron Architectures [34], Apache's CloudStack Architecture [3], and

the VirtualBox "Architecture" which isn't a true Virtual Network Architecture [30]. We start first with the VMware Architectures.

2.3.1.1 VMware Architecture

VMware has two slightly different main architectures, with a third architecture being based off the second architecture. Their difference is the use of standard switches versus distributed switches, with the rest of the architecture being based on this difference. Because of this we will describe both the standard and distributed switch architectures (and the Palo Alto Networks variant of the distributed switch architecture).

Standard Switch Architecture The vSphere Standard Switch Architecture allows for the creation of an internal virtual network within each ESXi host. A standard virtual switch is placed on top of each host and connects VMs to each other and the external network outside of the host machine. In this section we describe the different components of the architecture and how they work. Then we look at some use cases and trade-offs.

The standard virtual switch (vSwitch) is more similar to a physical switch than the distributed vSwitch, as it maintains both the data and management planes [57]. Figure 7 shows an example of the Standard Switch Architecture and how the components are connected. This architecture differs from the distributed architecture as each standard vSwitch operates on a single host, independently of one another. Because of this, each standard vSwitch implements both the data and management planes of the network [57]. Standard vSwitches are configured and maintained individually and are hosted on an ESXi host [57]. A vSwitch's port groups connect to a VM's virtual NIC, providing connectivity to the other VMs on the host [57]. These port groups can be assigned to any VMs on the same host as

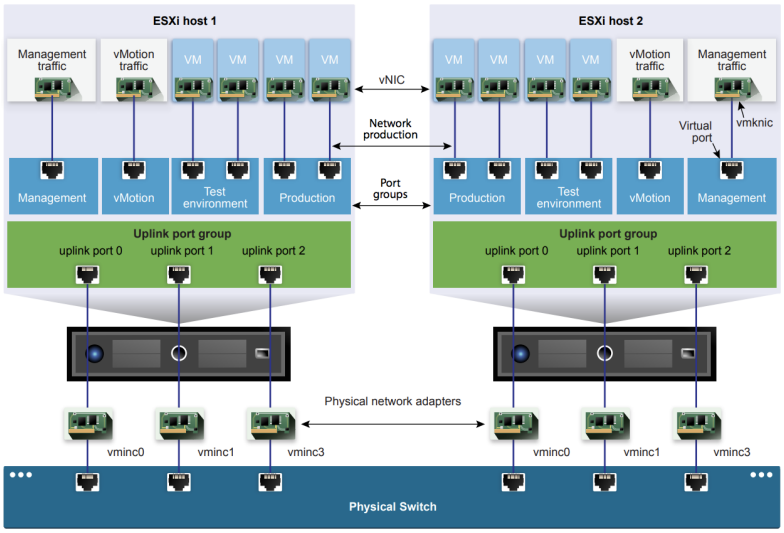

Figure 7: VMware vSphere Standard Switch Architecture [57]

the vSwitch, allowing for groups of VM's to be managed together through a port group's specifications. All VMs connected to the same port group on a standard vSwitch are in the same "network" or subnet, even if they aren't on the same host [55]. Figure 7 shows an example of different port groups, such as "Test Environment" and "Production", connecting to the vNICs of the VMs. The port group labeled "Management Traffic" is connected to a component called the VMKernel, with it's own Virtual NIC called the vmknic. The VMKernel component is "an adapter that is used to provide network connectivity to hosts and to accommodate system traffic for features of the network such as fault tolerance and IP storage" [57].

While vSwitches provide connectivity between VMs, they also can connect a host and its VMs to the external network. They do this through a set of ports, called uplink ports, which connect the ESXi host to some physical network device [57]. The physical switch's NICs are assigned to the uplink ports of the standard vSwitch. An example of this is shown in Figure 7. A vSwitch provides external network connectivity to the VMs on the host by connecting their port

groups to the uplink port groups [57]. Figure 9 shows an example. However, if an uplink port is not connected to a physical NIC, all VMs in the port group connected to that uplink can only communicate with each other and not the external network [57]. Each uplink port group on the standard vSwitch has the capability to connect to one or more physical NICs. This is called NIC teaming and allows for load balancing and a failsafe if a physical NIC goes down [57].

The vSphere Standard Switch Architecture is mainly used when a less complex Virtual Network is needed, such as having ESXi hosts and VMs that need to connect to each other in a way similar to a physical network. Other than basic cases, there isn't much reason to use the Standard Switch Architecture over the distributed version. The Standard Switch Architecture is not as flexible as the vSphere Distributed Switch Architecture and has less features as well. This is because each standard vSwitch can only manage one host and must maintain that host's data and management planes for the internal network. Managing a VM as it moves across hosts is not possible with a standard vSwitch and once a VM is moved to a new host, any management or network preferences have to be recreated [57].

Distributed Switch Architecture The vSphere Distributed Switch Architecture is more robust than the Standard Switch Architecture. The Distributed Switch Architecture centralizes the management plane of the vSwitch so that the vSwitch can be logically connected to multiple ESXi hosts, allowing for a more flexible Virtual Network to be created [57]. In this section we describe the components that differ from the Standard Switch Architecture and how they work. As with the Standard Switch Architecture, we then look at the potential use cases and trade-offs.

The Distributed Switch Architecture has more functionality than the standard switch version due to the distributed nature of the vSwitches. Figure 8 shows an example of the Distributed Switch Architecture. There are many

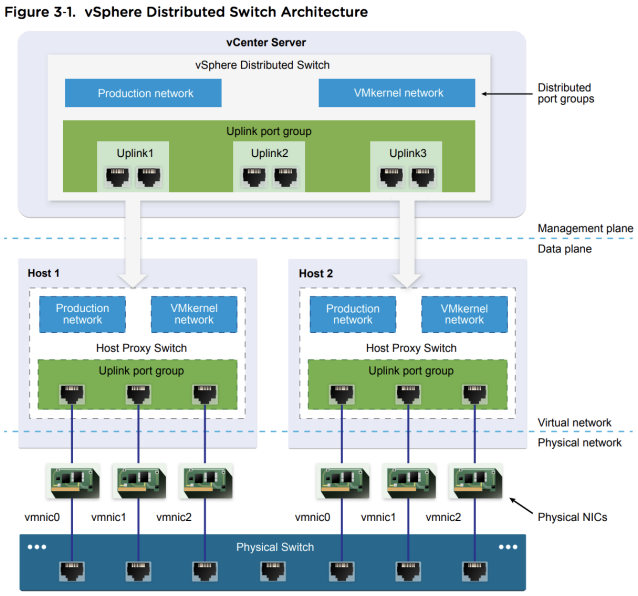

Figure 8: VMware vSphere Distributed Switch Architecture [57]

differences between the standard and distributed vSwitches. Because the distributed vSwitch is divided among potentially multiple hosts, it allows for the separation of the data and management plane, unlike in the Standard Switch Architecture [57]. The data plane acts much like the data plane of a physical network and is responsible for package switching, filtering, tagging packets, etc. In the Distributed Switch Architecture, the data plane is still controlled at the host level, same as the Standard Switch Architecture. The data plane component of a host in the Distributed Switch Architecture is called the "host proxy switch" and is the part of each host that receives the network configurations from the management plane [57]. The management plane is responsible for configuring the data plane, i.e. the virtual switches and the hosts where the data plane resides [57]. However, in the Distributed Switch Architecture, the management plane is now controlled by a new component called vCenter, a virtual server. The configurations for a distributed vSwitch are made in vCenter and then are stored locally in all the hosts that are "connected" to the distributed vSwitch [57]. "By putting the management plane in

a vCenter server, functionality of the network can be controlled at the highest level" [57]. This is especially useful in data centers, where the configuration of all the switches throughout the network can be controlled in one location [57].

The distributed vSwitch has a few differences in the data plane compared to the standard vSwitch. Like the Standard Switch Architecture, the Distributed Switch Architecture contains uplink ports on the hosts that connect to the NICs of the physical switch. The difference between the uplink ports for each vSwitch is that with the separation of the management and data planes, network configurations can now be made for each uplink port group [57]. These configurations then propagate down to all the host proxy switches that contain the uplink port group being configured. Another difference in the data plane level is the addition of the distributed port group. The distributed port group is similar to the port group used in the standard vSwitches to connect the vSwitch to the VMs on the host [57]. As with the standard vSwitches, the distributed port groups are identified with labels such as "Production Network". The difference between a standard port group and a distributed port group is the latter can be connected to VMs on multiple hosts and configured from vCenter to have different network settings. This allows certain VMs, even if they are on different hosts, to share network settings by being connected to the same distributed port group [57]. Different policies such as load balancing, security, etc. for a group of VMs can be sent to the host proxy switch and applied differently depending on the distributed port groups [57].

We have discussed the different distributed vSwitch components, but not yet how they interact. Figure 9 shows a very helpful example of how the components of an example distributed switch system are connected. The VMs and VMKernels on each host are each connected to a port in their respective distributed port group. In this example there is only one port group for the VMs and one for the VMKernels, but there can be multiple port groups for each. Also, each VM, no matter which

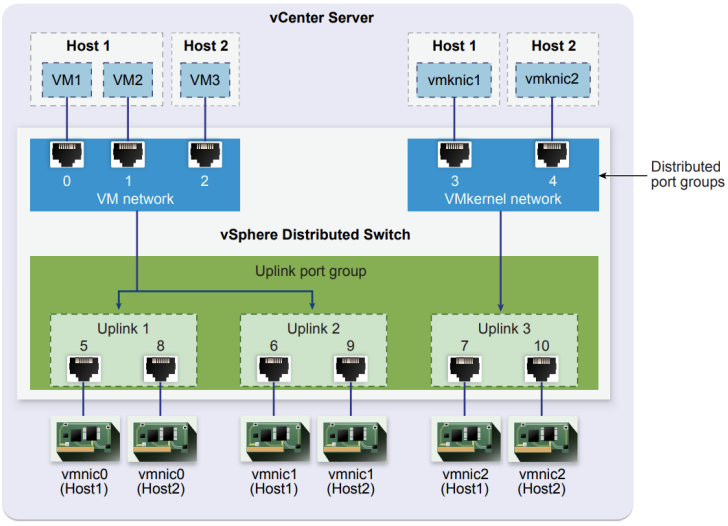

Figure 9: VMware vSphere Distributed Switch Architecture Data Flow Example [57]

physical host it is on, can belong to any of the distributed port groups. Each distributed port group is connected to one or more uplink port groups to allow for external network connectivity. In figure 9, the "VM Network" group is connected to two uplink port groups. The distributed vSwitch allocates an uplink port for each physical NIC on the host. These uplink ports are then connected to their respective physical NIC. In the figure, each uplink port group has two ports, one to connect to Host1's physical NICs and one to connect to Host2's physical NICs. If Host1 had another physical NIC, a 4th uplink port for Host1 would be created.

Data flow in the Distributed Switch Architecture is designed to allow for redundancy to ensure that VM's can communicate, even when a component is failing. This is made possible by the ability to connect distributed port groups to multiple uplink port groups, allowing for the use of multiple physical NICs to transmit a VM's packet. Figure 10 shows how a packet travels from a host out to the external network. In the figure, the details of Host2 are omitted. For example, if VM1 wants to send a packet out of the physical host, the packet would first go to Port 0 in the port group "VM Network". The packet then goes through either

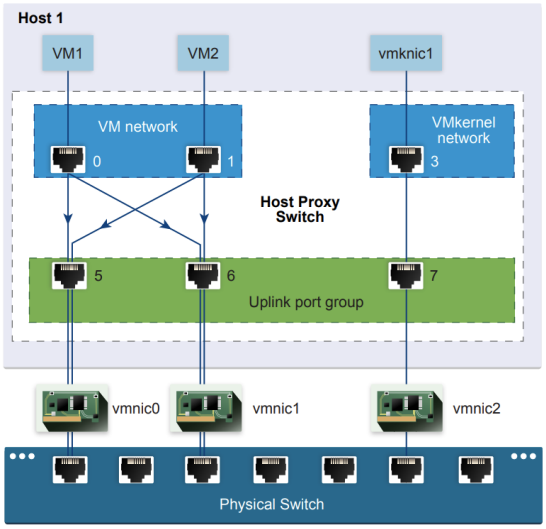

Figure 10: VMware vSphere Distributed Switch Architecture Packet Flow Example [57]

uplink port 5 or 6, because both ports are connected to "VM Network". If the packet goes through uplink port 5, it then goes through the "vmnic0" physical NIC of Host1; otherwise it goes through "vmnic1". Through either of those Host1 physical NICs, the packet then goes to the physical switch, which can then forward the packet to its next hop on the way to its destination. [57]

The vSphere Distributed Switch Architecture has more potential use cases than the Standard Switch version, such as in data centers and cloud computing that require more robust and available virtual networks [55]. The use of distributed vSwitches and a central management server allow for more flexibility and more available services. The ability to control the data plane of each host from a central component allows for much more flexibility over a Standard Switch Architecture, which requires configuring each host's vSwitch individually. The ability to have VMs on different hosts be a part of the same logical subnet is very beneficial, because it allows for the creation of any Virtual Network Topology, which wouldn't be possible on a physical network. The distributed vSwitch allows for VMs to maintain consistent network configurations even as they are moved across the

network to different hosts [57]. One downside to the architecture is that once a host is connected to a distributed vSwitch, it must always be connected to some distributed vSwitch. Another downside is the need to send the configurations of each host over the network, which creates some overhead and has the potential for failure. However, the overhead is small and done in increments [57]. Despite this, the Distributed Switch Architecture allows for more flexibility and more features than the Standard Switch Architecture. Compared to other architectures, both VMware Architectures contain more options for use of virtual network, virtual security, and distributed components. The virtual network and virtual security components are created and managed by VMware's NSX. NSX is the system for creating and managing the distributed vSwitch, complex virtual networking components, and virtual security components such as virtual firewalls [58]. NSX allows for more complex network function virtualization. VMware is at an advantage because they specialize in virtualization, making it easier to integrate more robust virtual components. However, VMware's architectures are limited to managing the virtual components of a network and don't allow for management of physical network components. This means that other Architectures must be used with VMware to manage a full Virtual Network system.

Palo Alto Networks + VMware Architecture The Palo Alto Networks (PAN) VM-Series Architecture builds on top of the VMware vSphere Distributed Switch Architecture to add more security features [60]. The PAN VM Series Architecture is different from the other VMware architectures because of the integration of Palo Alto Networks security components. The additions are few enough that the architecture is similar to the Distributed Switch Architecture talked about above, but the security components added are important enough that it deserves its own

section. We first discuss the PAN components and then discuss use cases and trade-offs.

VMware has its own virtual network security called NSX, but this architecture allows for the additional use of 3 Palo Alto Networks components: VM-Series Firewall, Dynamic Address Groups and Panorama Centralized Management. "The VM-Series Firewall is a state of the art firewall that has the functionality of a hardware firewall but is distributed and virtualized" [60]. The firewall allows for the testing of unknown malware in a sandbox environment and many other IDS type features which are absent in the previous VMware architectures [60]. Dynamic Address Groups allow for the tagging of VMs. Because VMs can be moved from one physical host to another, this allows for security policies to be applied to VMs even as they travel across the network [60]. The Panorama Centralized Management is a central management system for the VM-Series Firewalls that allows for configuring devices, deploying security policies, and performing analysis across all the virtual firewalls in the network [60].

These three additional Palo Alto Networks components added on to VMware's Distributed Switch Architecture creates a new, combined architecture. Some of the things that could be improved about the Distributed Switch Architecture are certain security aspects, such as the lack of visibility into intra VM traffic and slowness [60]. This architecture greatly increases security for any Virtual Networks created. The potential downside is the system will be made slower due to another centrally managed component and the added overhead of all the security utilized. However, the system isn't slowed enough to harm the network and is outweighed by the benefits of having the extra security added. The use cases for this architecture are the same as the Distributed Switch Architecture, since the network components of the architecture operate in the same way. The difference is

simply the added security, which is beneficial in data centers, cloud computing, and really any use case.

2.3.1.2 Juniper Contrail Architecture

Juniper Networks is a company that specializes in network solutions, both physical and software-defined. A software-defined network is a network which separates the control and data plane. A central controller makes the forwarding decisions and propagates those decisions to the switches, which become simple forwarding devices [25]. Juniper's Virtual Network Architecture is called Contrail. "Juniper Network's Contrail is an open-source software-defined networking (SDN) solution that automates and orchestrates the creation of highly scalable virtual networks" [22]. Potential use cases are as a cloud networking platform and for network function virtualization [22]. Contrail allows for the creation of overlay virtual networks to allow for a separated distribution of physical and virtual resources [22]. An overlay network is when the physical network is used to provide connectivity, while the virtual networking components lay on top of the physical servers routing the data [22]. In this section, we describe Contrail's components and how they work, followed by use cases and trade-offs.

Figure 11 shows the details of the Juniper Contrail architecture and how the components are connected. While there are many components in the system, the two main Juniper created components of the architecture are the SDN Controller and its subnodes, and the vRouters. Contrail's software-defined network controller is used for management, analytics, and control mechanisms of the system [22]. It is logically central, but is physically separate, meaning that the controller is hosted on multiple physical servers, but is viewed by the rest of the network as one controller when communicating with it. VRouters are similar to the VMware vSwitch that we described earlier, except the vRouter also provides routing services [22]. The

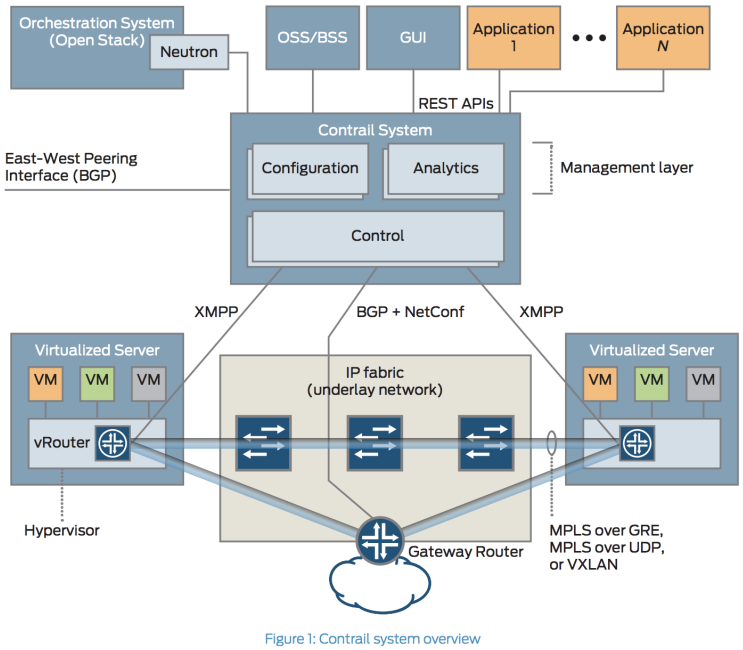

Figure 11: Juniper Contrail Architecture [22]

vRouter acts as a forwarding plane that lies in the hypervisor of a virtualized server [22].

Contrail's SDN Controller is divided into three main services: Configuration Nodes, Control Nodes, and Analytics Nodes [22]. The configuration nodes are in charge of the management plane and communicate using REST APIs. They translate a high level data model into a low level model that can properly interact with network elements [22]. The control nodes implement the logically centralized portion of the control plane and are in charge of applying the changes made by the configuration nodes. They propagate the low level model created by the configuration nodes to and from network elements and peer systems [22]. This communication is done using XMPP to control the vRouters and a combination of BGP and NETCONF protocols to control the physical routers [22]. The analytics nodes are in charge of collecting statistics for troubleshooting and network usage stats. They capture real time data and turn it into a presentable form [22]. Contrail

utilizes what is called an overlay virtual network, as described earlier. The vRouters create tunnels between each other, so when a VM on one physical host sends a packet to another, it can be routed without interacting with another overlapping virtual network and without having its data being viewed by any unauthorized routers/hosts on the way to its destination [22]. VMware's Virtual Network Architecture with NSX allows for similar functionality.

Other than the two Juniper components, there is also an orchestration system for the virtual networking, which can utilize two other Virtual Network Architectures discussed later, called OpenStack and CloudStack [22]. The orchestration system is in charge of communicating with the virtualized network functions such as virtual firewalls and virtual routing components. OpenStack and CloudStack are Virtual Network Architectures like Contrail, but can be utilized to simply manage a Virtual Network's components, which is why they are used in Contrail's Architecture. Contrail also utilizes 3 main interfaces. The first is the Northbound REST API that is used to talk to apps and the orchestration system [22]. The second is the Southbound interfaces used to communicate with the vRouters and the physical network elements [22]. Lastly, the East-West interface is utilized for peer to peer communication, such as vRouters communicating with one another [22].

Juniper's Architecture utilizes many different forms of communication. The overlay network's messaging format is based on MPLS L3VPNs (for layer 3 overlay networks) and MPLS EVPNs (for layer 2 overlays) [22]. Contrail's virtual overlay network data plane can utilize either MPLS GRE/UDP or VXLAN to create tunnels of communication between vRouters [22]. The control plane utilizes BGP for communication [22]. The protocol for communication between the SDN Controller and the vRouters is very similar to and based off XMPP, and is semantically very

similar to the BGP protocol [22]. We wont go into detail on how these protocols work, as knowing how they work isn't important to understanding the architecture.

Use cases for Contrail include as a Private Cloud for companies, as an Infrastructure as a Service, as a Virtual Private Cloud for Service Providers and for managing Network Function Virtualization for Service Provider Networks [22]. Contrail, like other Virtual Network Architectures, is utilized in data centers, allowing for the creation of multi-tenant virtualized data centers [22]. Contrail allows each tenant to share physical resources while being assigned their own virtual resources, such as their own virtual networks, virtual storage, and VMs [22]. For Network Function Virtualization, Contrail allows for management and orchestration of virtualized functions such as firewalls, IDSs, caching, etc. Contrail's strengths are that it allows for multiple options in choice of Virtual Network Orchestrator and type of virtual network components. It also utilizes overlay networks and tunneling, allowing for logical separation of networks while keeping data protected. However, it doesn't have as much flexibility as the VMware Distributed Switch Architecture, because its vRouters have similar shortcomings to the VMware Standard vSwitch, such as only being able to connect to VMs on one host. Contrail also lacks in security compared to the vSphere Architectures. While VMware has both NSX and PAN distributed security features that can be managed from a central location, Contrail's only allows for the use of external security components. While other security features such as IDS's and Firewalls can be useful in the architecture, they cant be distributed or centrally managed to more easily secure large parts of the Virtual Networks created.

2.3.1.3 OpenStack Architecture

OpenStack is a company that created a Cloud Management System that can control "large pools of compute, storage, and networking resources throughout a

data center" [38]. The most important part of that system is virtual network management and virtual components, which allow for the creation of virtual and overlay networks. OpenStack Neutron has different architectures based on the needs of the user, such as classic, high availability, and provider (overlay) networks [34]. Architectures also vary based on different components added or substituted, such as architectures with and without a linux bridge. The primary use case for Neutron is in data centers, where it is used to create and manage Virtual Networks similar to the other architectures we have described [38]. In this section, we look at the Classic Architecture with Open vSwitch, as it is the "classic" architecture type for Neutron and shows the base case for OpenStack Virtual Networking. We then briefly discuss the High Availability with Distributed Virtual Routing Architecture and the Provider Network Architecture to show the full capabilities of Neutron. We then look at use cases and trade-offs.

OpenStack's Neutron Virtual Network Architecture is used for controlling the networking components of a Virtual Network. "It contributes the networking portion of self-service virtual data center infrastructure by providing a method for regular (non-privileged) users to manage virtual networks within a project" [34]. An overview of the basic network layout can be seen in figure 12. The infrastructure is made up of 3 main parts: a controller node, a network node, and one or more compute nodes [34]. The controller node has one network interface and manages the network. The network node has four network interfaces and acts as a router or a gateway to the network. The compute nodes contain three network interfaces and hosts the VMs. There isn't much to say about the controller node, other than it is used to manage the whole network, but there is more to the network and compute nodes.

Figure 13 shows the overall architecture and components for the network node. The network node is the most complex of the three nodes and is used as a

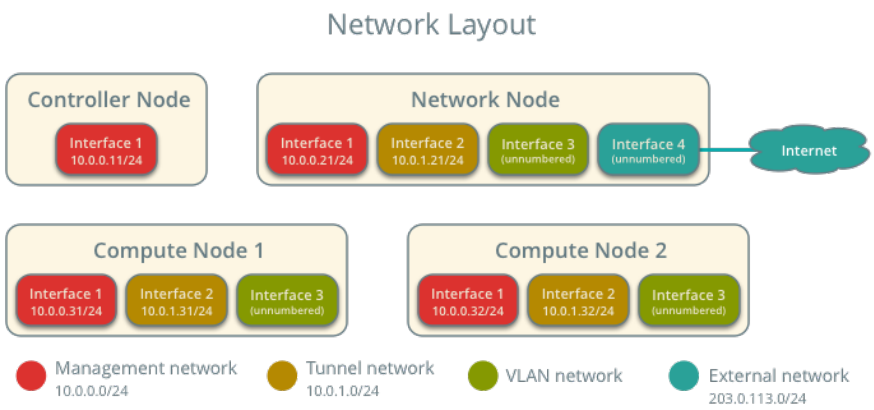

Figure 12: OpenStack Virtual Network Basic Layout for a Classic Open vSwitch Architecture [34]

sort of gateway router for the network, connecting subnets together and connecting the network to the Internet [34]. It contains four main components: an Open vSwitch agent, a DHCP agent, an L3 agent, and a Metadata agent [34]. The Open vSwitch agent is in charge of managing all the virtual switches. It manages the virtual switch's connectivity and their interactions with other components through its virtual ports. The Virtual switches interact with other components such as namespaces, Linux bridges, and underlying interfaces [34]. The DHCP agent is in charge of managing the qhdcp namespaces, which provide DHCP service to VMs of the system using an internal network [34]. The L3 agent manages the qrouter namespaces, providing routing between internal and external networks [34]. It also provides routing between internal networks, allowing these networks to communicate among each other. It also routes metadata between instances of the network and the Metadata agent. The Metadata agent handles all the metadata operations for all VMs of the network [34].

Figure 14 shows the overall architecture and components for the compute node(s). The compute node is a host, similar to the VMware ESXi host, that can host multiple VMs [34]. Multiple compute nodes can be connected as a subnet or

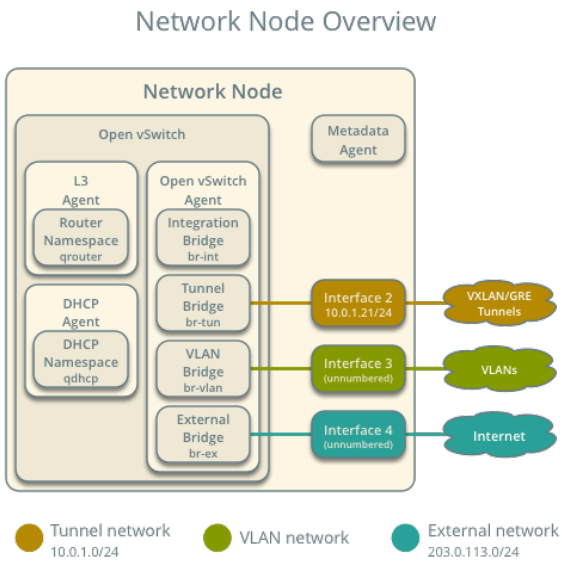

Figure 13: OpenStack Virtual Network's Network Node Design [34]

even one compute node can be considered a subnet itself. The compute node contains a virtual switch for its hosted VMs to connect to and has two components. The first component is another Open vSwitch agent that performs the same tasks as described in the network node [34]. The compute node agent manages Open vSwitches on the compute node, while the Open vSwitch agent in the network node manages all the switches in the network. The other component is the linux bridge, which handles security groups and their applications to the different VMs [34]. Due to limitations with Open vSwitch and iptables, the networking service uses a linux bridge to manage security groups for VMs [34].

Figure 15 shows an example of how the compute, controller, and network nodes are all connected. In the figure, the network node is acting as the gateway router, routing between compute nodes and from compute nodes to the external network. The compute nodes act as subnets, with each having a switch to route between instances. The figure also shows 2 overlapping networks that don't know about the other: the Tunnel network and the VLAN network. This is a good

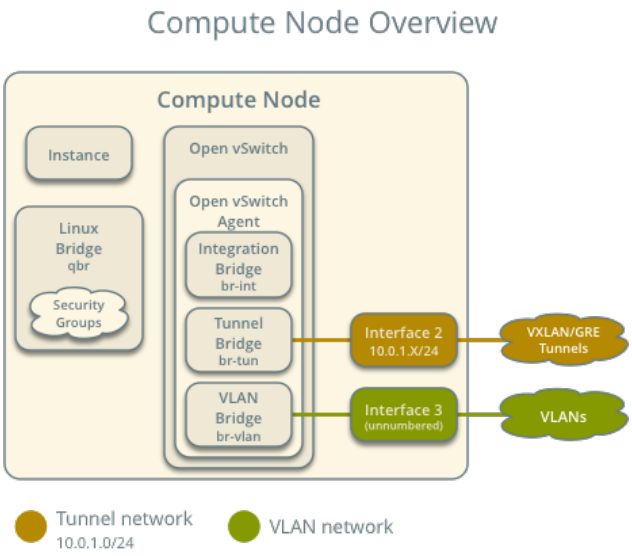

Figure 14: OpenStack Virtual Network's Compute Node Design [34]

example of how virtual networks can be separate, even though they are on the same physical machine.

A High Availability with Distributed Virtual Routing Architecture adds to the Classic Architecture by adding distributed virtual routers [35]. Figure 16 shows the overall architecture with the added distributed virtual router. The distributed virtual router is added to every compute node, and compute nodes now have an interface connecting them directly to the external network. This is done to allow for direct connection to the external network and other compute nodes, bypassing the network node [37]. The distributed virtual router also allows for address translation for the Virtual Network on the compute node [37]. This is normally done by the network node, but is now also done by the distributed router. The point of the distributed router is to allow for quicker connection to other nodes and the external network. The distributed router is also important because it makes Virtual Networks more scalable by taking a lot of the load off of the network node and distributing it [37].

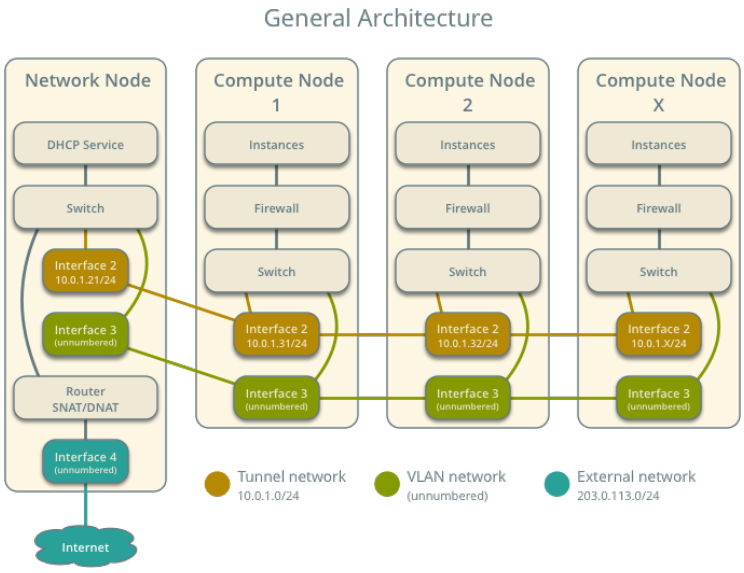

Figure 15: OpenStack's Virtual Network Architecture and Components [34]

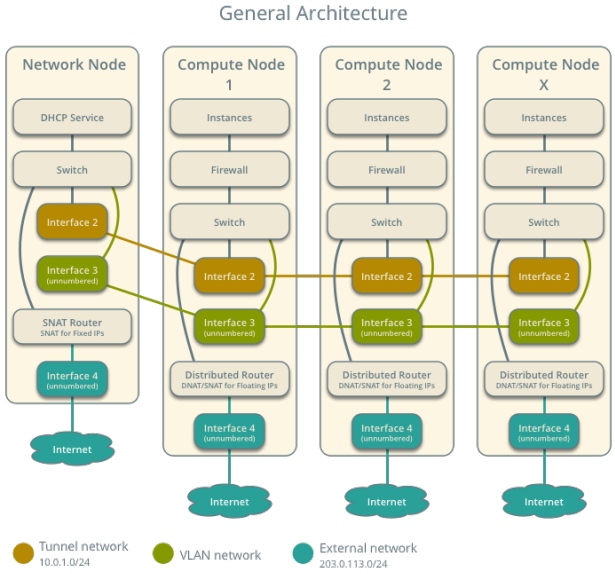

Figure 16: OpenStack's High Availability using Distributed Virtual Router Architecture [35]

A Provider Network Architecture is similar to the Classic Switch Architecture, but with the virtual components being connected to an underlying physical network rather than directly to the Internet. The main difference is that the Provider Network has the compute and control nodes connected to physical network components such as routers, switches, etc. [36]. The controller and compute nodes have interfaces that directly connect to ports on a physical network component to allow for data to move through the overlay network [36]. Another difference is that the network node and controller node are now combined so that the controller node is now also partly responsible for network management [36]. The Provider Network Architecture allows for the use of overlay networks and communication inside and outside a Virtual Network, while the Classic Architecture is more utilized for intra-data center communication between nodes, though communication with external networks is still common [36].

The OpenStack Neutron Architectures allow for many different ways to create and manage Virtual Networks and their components. This is shown in the Juniper Contrail Architecture that we discussed, where OpenStack's Architecture can be used to control all the virtualized network components in the system. Like other Virtual Network Architectures, the Openstack Neutron Architecture is mainly used in creating virtual data center networks [36]. Neutron is useful because it has many architecture types depending on the type of Virtual Network system that is needed for a solution. The downside is that once one is selected, it is harder to switch to a different architecture and one can't reap the benefits of the different architectures at once. Neutron also has an architecture that utilizes distributed routers, which allows for more efficient intra-data center communication [37]. It also makes the system scalable and efficient due to distributing some of the work done by the network node, such as address translation [37]. Another downside is that similar to Juniper's Contrail Architecture, there aren't as many security features as

VMware's Architectures outside of using normal network IDSs and virtual firewalls. This is seen with OpenStack's vSwitches having limitations that force the network node to use a Linux Bridge to manage the security of VM groups.

2.3.1.4 Apache CloudStack Architecture

The Apache Software Foundation is a company that focuses on developing open-source software. One of their projects is Apache CloudStack. "Apache CloudStack is an open source cloud computing software, which is used to build private, public and hybrid Infrastructure as a Service (IaaS) clouds by pooling computing resources" [27]. Like other architectures, the Apache CloudStack Architecture is used in virtualizing datacenter communication. Apache CloudStack, like OpenStack, has many architectures based on need and is used as an option for managing network components in the Juniper Contrail Architecture. However, CloudStack differs from OpenStack in that the different architectures it employs are based on the size of the desired network and whether certain components, such as storage, should be separated from the rest of the network. In this section, we first describe the CloudStack Architecture and its components, followed by use cases and trade-offs.

The Apache CloudStack Architecture allows for the virtualization and management of data center networks. Figure 17 shows an overview of the general CloudStack Architecture. CloudStack "supports a large set of hypervisors, scalable architectures, multi node installation and load balancing, making it a high availability system" [27]. However, the CloudStack Architecture has fewer components and less functionality than the other Architectures we have covered, with the basic architecture consisting mainly of a management server, storage, and computing nodes. The management server is what manages the cloud resources such as provisioning storage, hosts, and IP addresses [27]. Admins can use the

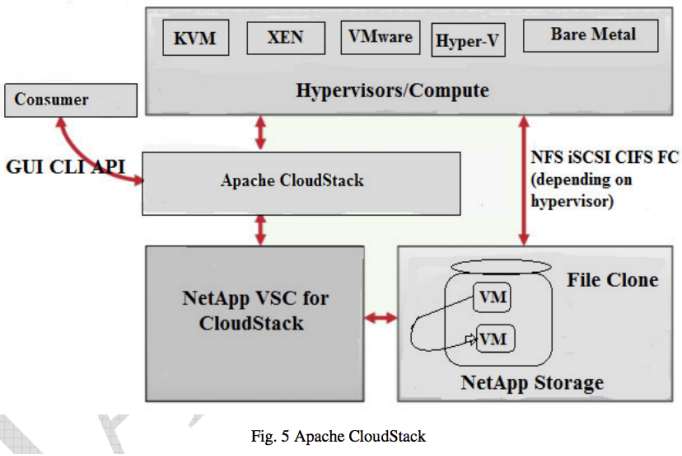

Figure 17: Apache CloudStack's Virtual Network Architecture and Components [27]

management server through either a user interface or through its API. Storage has the option of being located on a server within the same part of the network as the other components or being separated into its own storage subnet [27]. The computing nodes are the machines hosting the VMs and other virtual network components [27].

CloudStack has a range of architectures from a very simple, small network to more complex, more divided networks. The CloudStack Architecture has both basic networking, similar to an AWS type network, and more advanced networking for more complex network topologies [27]. Figure 18 shows the Small-Scale Deployment Architecture, the most basic CloudStack Architecture [3]. The architecture has one subnet with a gateway router/firewall, a layer 2 switch, a management server, an NFS storage server, and the compute nodes holding the virtual resources [3]. It also contains a VMware vCenter server if VMware virtual network components are being used [3]. CloudStack has other architectures with more flexible setups, such as architectures that divide certain components between different subnets.

The CloudStack Architecture allows for various Virtual Network setups ranging from smaller to larger networks. CloudStack is mainly used in data centers

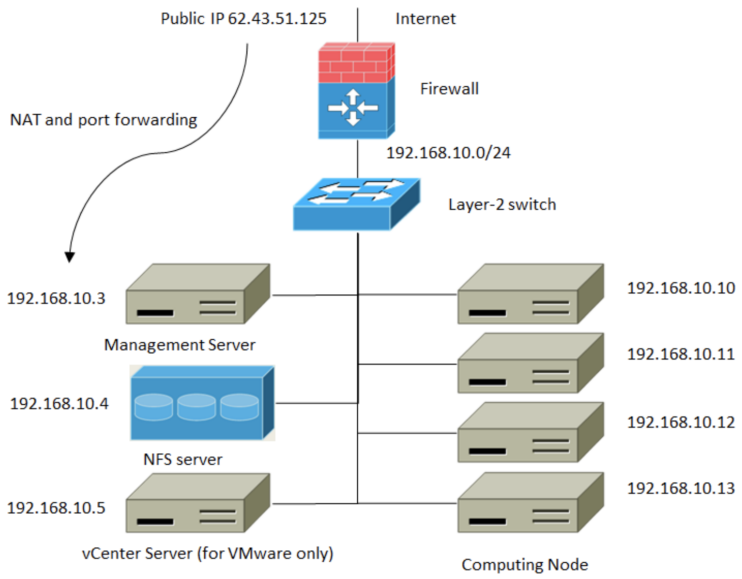

Figure 18: Apache CloudStack's Virtual Network Small-Scale Deployment [3]

and cloud computing to create and manage Virtual Networks, making intra data center communication easier. A big benefit of the CloudStack Architecture is its flexibility in component use. It allows the use of many different VM types, virtual component types, and even the use of VMware's vCenter to manage Virtual Network resources (if the Virtual Network components are VMware's). The CloudStack Architecture also allows for more functional network setups, which allow the architecture to handle bigger networks and/or the separation of storage nodes, which helps lower the load of the management network. However, despite the increased functionality of some of the Architectures, it still lacks in flexibility of its implementations. The different architectures of CloudStack are based more on division of storage than in creating more architectures of varying uses, unlike OpenStack which allows for plenty of architecture options. The CloudStack Architecture also lacks in use and management of virtualized security functions from a central location. It doesn't have any dedicated management for security and instead relies on the use of outside virtual security components and management.

This makes it more difficult to manage network-wide security and protect any Virtual Networks created.

2.3.1.5 Oracle VirtualBox "Architecture"

Of all the Virtual Network Architectures, Oracle's VirtualBox is the easiest to describe. This is because it doesn't really have a Virtual Network Architecture, hence the quotations surrounding architecture in the heading. To create a Virtual Network in VirtualBox, it is necessary to use regular VMs as virtual routers, rather than using virtual switches that better emulate a physical switch. It also requires personally creating a management network through VirtualBox's network setting and by enabling port forwarding [30]. Other steps to get the network properly configured are needed as well. So while a Virtual Network can be created in VirtualBox, it is more of a workaround than a true Virtual Network.

2.3.2 Architecture Comparisons

In this section we look at the overarching view of all the architectures and shortly describe how they compare to each other in certain aspects. Each architecture has similar use cases, mainly in data centers and cloud computing, and each has its own trade-offs. No architecture is necessarily better overall than any other, but rather each is better in different areas such as flexibility or management. Table 1 shows an overview.

For flexibility of implementation, the VMware vSphere Distributed Switch Architecture, the PAN VM-Series Architecture, and the OpenStack Architectures allow for diverse choice in architectures based on different virtual network components. However, OpenStack's architectures vary and once one is picked, the whole system is set. VMware and the PAN architectures, however, allow for flexibility in choice of virtual switch, virtual security components, etc while also

Architecture	Implementation Flexibility	Management	Security and Management	Virtual Component Flexibility
VMware vSphere + PAN	Good, multiple options for vSwitch/security	Good management of virtual network, lacks physical network management	Good, has many distributed/centrally managed security features	None, must use VMware virtual components
Juniper Contrail	Some, has only one architecture, must use vRouter	Manages both virtual and physical, Others manage virtual better	Relies on outside virtual security components	Very, allows different hypervisors and virtual components
OpenStack	Good, has multiple architecture and virtual component options	Same as VMware, virtual management slightly worse	Relies on outside virtual security components	None, must use OpenStack virtual components
CloudStack	Some, has different architectures based on network division	Same as Juniper	Relies on outside virtual security components	Very, can use different hypervisors and VMware vCenter

Table 1: Architecture Comparison Overview

allowing for them to be implemented at any point in time. CloudStack allows for the use of VMware network virtualization, so through that capability it also allows for some flexibility of architecture. The Contrail Architecture has some flexibility, but not as much as the other architectures, being limited by its lack of options for virtual switch (vRouter must be used) and virtual security components.

For management of a network, each architecture has its strengths and weaknesses. Juniper Contrail and CloudStack allow for management of both virtual and physical resources, while VMware and OpenStack don't. However, VMware and OpenStack allow for better management of virtual components and both allow for the distribution and central management of switches and subnets. This allows for the management of a VM's network configurations even as it moves around the physical network. For integration of security and easy management of it, VMware has the clear edge with its Distributed Switch Architecture, along with NSX and the PAN architecture. The vast options for virtual security through NSX and PAN puts the VMware and PAN architectures above the rest. The other architectures all rely on outside security components and don't provide centrally managed Virtual Network security.

CloudStack and Contrail allow for the most flexibility in type of hypervisors and virtual components used, with CloudStack being a little more flexible in this regard due to the Contrail Architecture needing to use its own virtual router. CloudStack is also flexible in that it allows for the use of VMware's vCenter to manage the VMware virtual components. Juniper also allows for the choice between CloudStack and OpenStack for managing virtual resources. VMware/PAN and OpenStack are not flexible in this category as they require use of their own virtual components.

Chapter 3

SYSTEM DESIGN

To test the idea that dynamic topology changes can be used to secure a Virtual Network, we must set up a system to show the potential effects. An important part of that system is creating a Virtual Network environment for testing. To do this we use VMware's virtual technology, which we explain in detail in section 2.3.1.1. VMware allows for a vast amount of options for virtualized network and security components, making the creation of a viable environment easier. It is flexible and, since we won't need to manage physical devices, it makes the most sense for creating our Virtual Network environment. Besides setting up a virtual environment, it is also necessary to simulate situations in which our dynamic defenses could be used as protection. We accomplish this through the use of Kali Linux and its many attack tools, which we use to run network attacks. We also set up a detection system, using the open source Snort IDS, to alert when an attack is happening. The alert is used to initiate our dynamic defenses. Finally, we implement virtual security components that are used in the dynamic defenses for securing the Virtual Network, which leverages VMware's NSX. In this chapter, we detail how our system is built by describing our Virtual Network environment, the security components utilized for our dynamic defenses and how they are set up, how we run the network attacks, and how we detect those attacks. In section 3.1, we describe how we created all the hypervisors, VMs, and other virtual components in our environment and how they are set up. In section 3.2, we discuss the creation and setup of NSX and our virtual security components, which we utilize for our dynamic defenses. In section 3.3, we discuss our network attacks and how they are run using Kali Linux. Finally, in section 3.4, we look at the setup of the Snort IDS, which is used to detect the network attacks in our experiments.

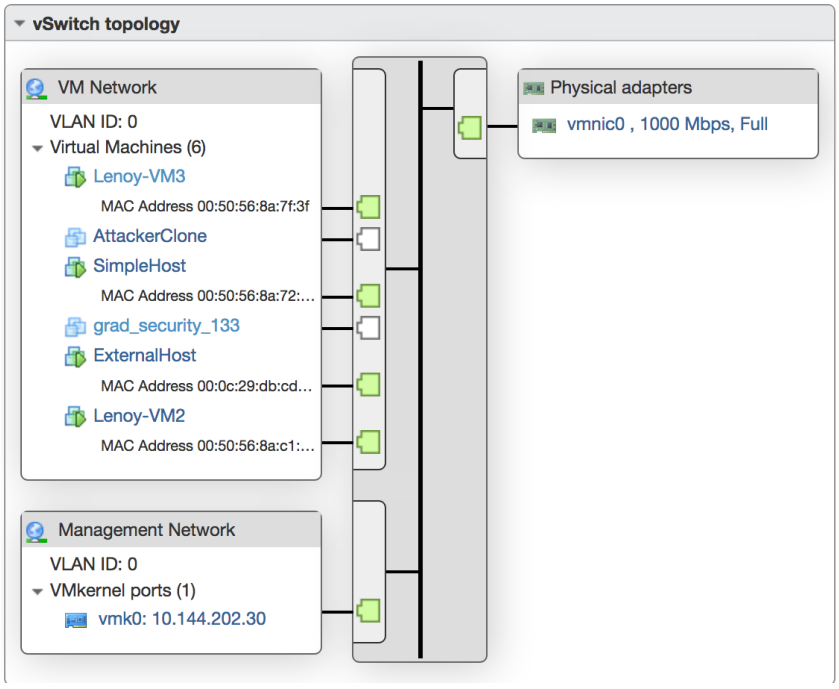

Figure 19: Topology of Host Virtual Network

3.1 Virtual Network Setup

In this section, we describe the setup of the Virtual Network environment from start to finish. We detail the creation of VMs, hypervisors, networks, and other components in our virtual environment.

To create our Virtual Network environment, we use VMware's vSphere and vCenter technologies, which are used for the creation of Virtual Networks. As described in section 2.3.1.1, vSphere is used to create Virtual Network components on top of an individual host and vCenter is used to centrally manage all the Virtual Network components for all the hosts. To store our system, we use a physical DELL machine with 12 CPUs, hyperthreading capabilities, approximately 1 TB hard disk, and 100 GBs of memory. The machine has a VMware ESXi hypervisor installed on it to allow for the creation and management of VMs. Figure 19 shows the topology of the Virtual Network environment on top of the physical host. The physical host

also has 4 NICs and is connected to a physical switch, which is connected to the gateway of the subnet. Only one NIC is utilized for our system for network connectivity, with a second NIC being used as a backup in case of failure. The host uses an internal DHCP server in a separate subnet, and an external cal poly DNS server called "larry" and the google DNS servers. The host also contains a standard virtual switch to allow external network connectivity for all the VMs. All created VMs are connected to the "VM Network" standard port group, while the VMKernel is connected to its own "Management Network" standard port group.

On top of the physical host we install a vCenter Manager, which is used to manage our Virtual Network environment. It is given an initial 40 GB hard disk and 10 GBs of memory, all of which is thin provisioned. Thin provisioning is when only the amount of memory needed at the time of creation is allocated for a VM [53]. More storage can then be added afterwards as needed. So for our system, even though the VM is given a 40 GB hard disk, only the part of the 40 GBs that is currently needed is allocated. All VMs created in our system use thin provisioning, since it allows for more flexibility in memory usage. Once the vCenter Manager is created, the web GUI can be accessed and used to create other VMs on top of the physical host. The vCenter management GUI can be accessed through a web browser by entering the IP address of the vCenter Manager. The vCenter Manager is the main tool we use to create and manipulate our Virtual Network environment.

On the vCenter Manager, we create 3 different "clusters" of machines. Figure 20 shows the vCenter layout with the 3 clusters. Clusters are used to abstract multiple machines as one entity, combining their resources into one cluster [56]. This is generally done for datacenters. However, since our environment wont be dealing with datacenter-like activity, we will use them to organize our VMs rather then to group resources. We create 3 clusters: a "mgmt-cluster" which holds the physical host and all the vms created on it, a "compute-cluster" which we use as

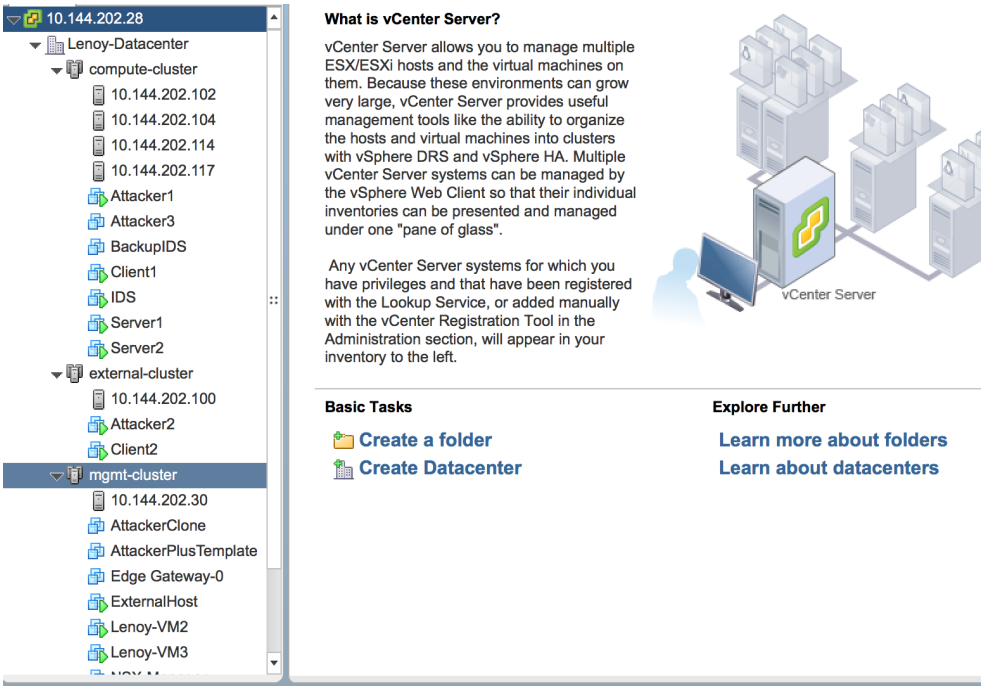

Figure 20: VCenter GUI and Clusters Layout

the internal network for our experiments, and an "external-cluster" which is used to hold other VMs that are not a part of the "internal network".

In a real Virtual Network, VMs and Virtual Networking components are created on top of many hosts and physical network components. However, this is very time intensive and requires a lot of physical resources that we do not have. Instead, we use nested virtualization to emulate a Virtual Network environment on a single physical machine. In section 2.2.3 we explain how nested virtualization works. With VMware nested virtualization, there is an outer guest VM and an inner guest VM. The outer guest, or the guest hypervisor, is an ESXi hypervisor VM which runs on the physical host, on top of the main ESXi hypervisor [32]. The inner guest is the nested VM that runs on top of another VM (the hypervisor VM) [32]. The nested hypervisors can be managed in the exact same way as a physical host's hypervisor, through vSphere's web GUI. This web GUI is accessed by entering the IP address of the ESXi hypervisor in a web browser. A nested ESXi

hypervisor VM can be added to a vCenter cluster as a host and acts as a physical host when interacted with through vCenter. After it is added as a host, VMs can be added to and run on the nested hypervisor.

To use nested virtualization to emulate a real Virtual Network system, we must first create the nested ESXi hypervisor VMs that will act as physical hosts. Each hypervisor VM is created on top of the real physical host and put in the standard port group "VM Network" to provide external network connectivity. In the vCenter view, the hypervisor VMs are placed in the "mgmt-cluster" with the main ESXi host (the physical machine) and the vCenter Manager. They can then be added to other clusters as "physical" hosts, which is what allows for nested virtualization. Once the hypervisors are added as hosts to the clusters, VMs can be created and "stored" on them. 5 ESXi hypervisor VMs are created in the "mgmt-cluster", with 4 being added to the "compute-cluster" as hosts and 1 being added to the "external-cluster" as a host. Each host created from the hypervisor VMs is given a 40 GB, thin provisioned hard disk, 8 GBs of memory, and 2 CPUs.

Of the 4 hosts added to the "compute-cluster", 2 utilize a standard vSwitch and standard port groups. The host on the "external-cluster" also utilizes a standard vSwitch. The other two hosts on the "compute-cluster" are connected to a distributed vSwitch called "DSwitch1". Figure 21 shows an example of the distributed switch's network topology. The distributed vSwitch allows for all VMs that are on a host connected to the distributed vSwitch to be in the same distributed port group. It also has the ability to utilize a distributed firewall, which we describe in section 3.2.

After creating the hypervisor VMs and adding them as hosts to the clusters, the VMs that act as the clients, servers, etc. are created on top of the emulated hosts. To allow for nested virtualization, the underlying physical host must have hardware-assisted virtualization enabled. However, if the physical host doesn't have

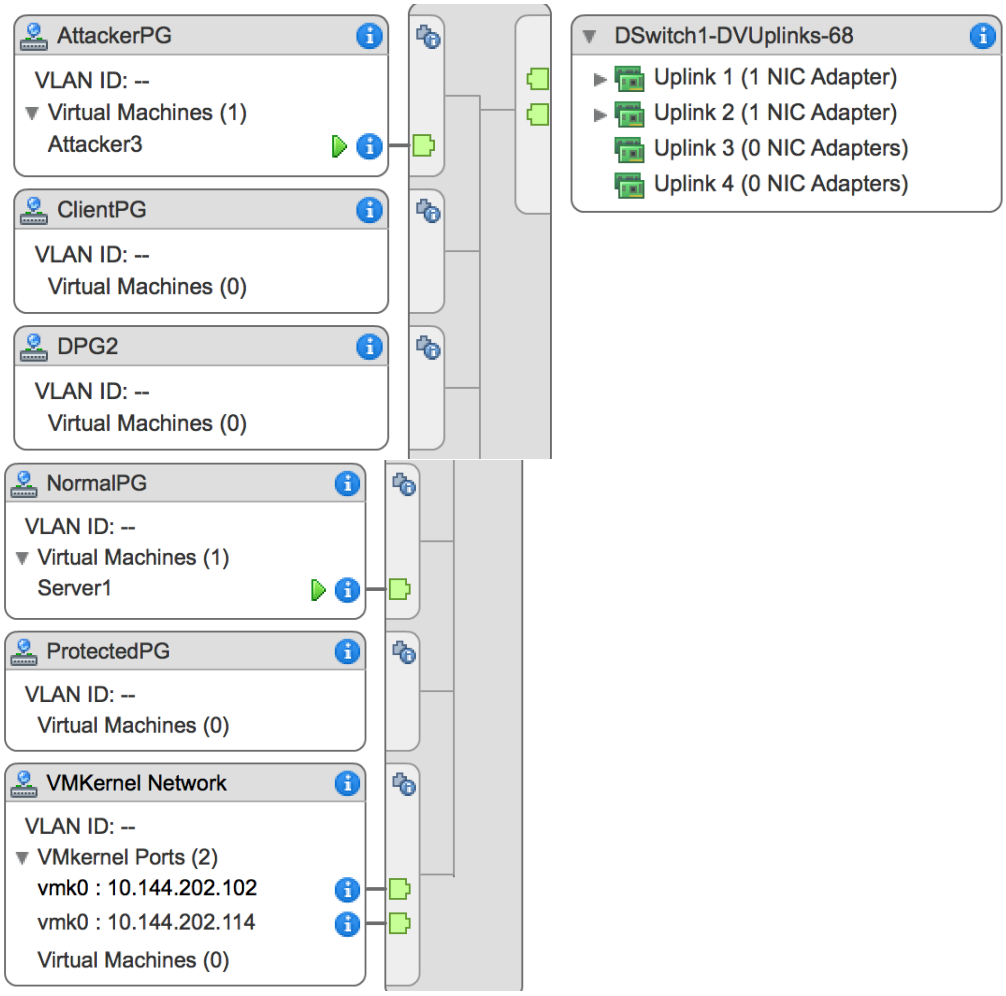

Figure 21: Distributed vSwitch Topology

this capability, nested virtualization can still be done by using 32 bit VMs instead of 64 bit ones. We choose the later option to simplify things. There are 4 types of VMs in our system: a client, an attacker, a server, and an IDS. We discuss the IDS in detail in section 3.4. We describe the other 3 here. For the server, a 32 bit, headless, mini Ubuntu VM is created. It has a 2 GB hard drive, 512 MBs of memory, and 1 CPU. Not much is stored on the server for the experiments, so using a headless, mini Ubuntu instance works well. For the client, not much memory or processing power is needed. Because of this, we initially looked at very lightweight VMs and decided to use the 32 bit Damn Small Linux OS, requiring only a 128 MB hard disk. However, we soon realized that a VM of that size was too small to be compatible with the VMware Virtual Network environment, as extra space is necessary to install certain tools needed to run commands remotely. Instead, we settled on using another 32 bit, headless, mini Ubuntu VM. The client is given the same specifications as the server. For the attacker, it is desirable to have a VM containing many built in network attack tools, making the process of compiling those tools much easier. Kali Linux is an OS that is specifically used because of its vast quantities of attack tool frameworks. A 32 bit, mini Kali Linux VM is used to create the attacker. The mini installation allows for customization of tools, requiring much less memory usage than a full Kali Linux VM. The attacker is given an 8 GB hard disk, 1 GB of memory, and 1 CPU. The "hping3" and "dsniff" packages are installed to get the network attack tools that are used in our experiments.

3.2 Security Design

Once the vCenter environment is set up, security components need to be added to be able to defend against the network attacks for our experiments. For some of the dynamic defenses, we utilize VMware's virtual, distributed security components. To use them, we install VMware's NSX. In this section, we discuss the

creation and setup of NSX security and networking, and the security components that will be used as a part of the dynamic defenses.

NSX allows for the use of advanced networking and security components, such as virtual gateways, logical switches, logical routers, etc. It also allows for the use of a distributed firewall and it is used to create the distributed vSwitch connected to the two hosts in the "compute-cluster". An NSX Manager VM is created on top of the physical host, where it is connected to the "VM Network" standard port group. In the vCenter view, it is placed in the "mgmt-cluster", where it has a full network view. To make hosts compatible and connectable with NSX components, the clusters containing the emulated hosts are selected to install the necessary NSX tools.

Two security components we heavily utilize in our system: the distributed firewall and spoof guard. The distributed firewall works like a normal firewall, except it is purely in software and is distributed similar to the distributed vSwitch. Any host or VM that is connected to a distributed vSwitch or that is in a cluster that has downloaded the NSX tools is covered by the firewall. Figure 22 shows the layout of the distributed firewall and example rules. The distributed firewall allows for very fined grained control of rule creation. For source and destination of packets, anything from a distributed port group to a whole datacenter can be selected. For our experiments, only distributed port groups or clusters (or "any") are selected for source and destination. There is also the ability to create a new definition of service type, by specifying the packet type and the parameters needed to trigger the firewall rule. Generally, we select the packet type of the attack being run. Finally, an action is selected: allow, drop, or block. For our firewall rules, block is always selected to activate enforcement of the rule.

The other component we utilize is spoof guard. Figure 23 shows an example of a spoof guard policy. Spoof guard is used to prevent VMs from spoofing MAC or

Figure 22: The Distributed Firewall GUI and Example Rules

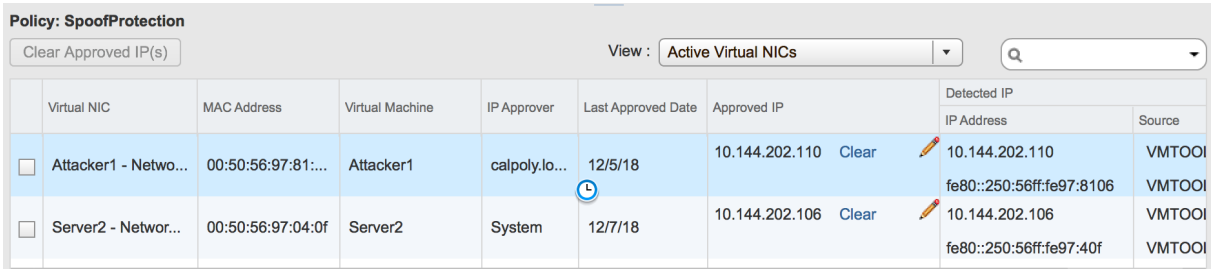

Figure 23: The Spoof Guard GUI and Example Policy

IP address information. To enforce spoofing protection, a new spoof guard policy is created. The port groups, or other components that should have spoofing protection, are specified in the policy. All the VMs in the connected port groups have their initial addresses registered. The spoof guard then checks all the packets coming from those port groups and if an address doesn't match the registered address, the packet is dropped. This prevents the VMs being monitored by the spoof guard from spoofing MAC or IP addresses.

3.3 Attack Design

For the validation of our dynamic virtual network, it is necessary to run real network attacks to test the feasibility of our security methods. To vary the testing, we use a few different categories of attacks to create a diverse set of experiments. In total, 6 different attacks are utilized: TCP Syn Flooding, ICMP Flooding, Smurf Attack, ARP Spoofing, DNS Spoofing, and NMAP Scanning. In this section we go

over how each attack is run in our system. We provide detailed explanations of each attack in section 2.1.

3.3.1 TCP Syn Flooding

TCP Syn Flooding is an attack where many TCP Syn requests are sent to use up a server's TCP resources, preventing or slowing other clients from creating new TCP connections [43]. We run the attack using the Kali Linux tool "hping3", which is used for running dos and ddos attacks. The command to run the attack is "hping3 -S –flood [victim-ip]", where the -S flag specifies that the attack should utilize TCP Syn packets and the –flood flag specifies the attack should send as many packets as fast as possible.

3.3.2 ICMP Flooding

ICMP Flooding is an attack where many ICMP packets are sent to a machine to try to use up bandwidth on that machine's links, preventing other data from traversing those links [31]. We also run this attack with hping3, using similar flags as the TCP Syn Flooding attack. The command to run the ICMP flood is "hping3 –icmp –flood [victim-ip]", where the –icmp flag is used to specify that the flooding attack should utilize ICMP packets.

3.3.3 Smurf Attack

A Smurf attack is when many ICMP requests (Ping requests) are sent to many different machines with the source address spoofed as the victim server's IP address [43]. The machines that receive the pings then send many ping responses to the victim server, with the goal being to cause a denial of service. We also run this attack with hping3, using similar flags as the ICMP Flooding command. The command to run the Smurf attack is "hping3 –icmp –flood -a [victim-ip]

[broadcast-ip]", where the first two flags are the same as the ICMP Flood command. The -a flag is used to specify the victim's IP address, which is used as the source address of the ping requests. The IP address at the end of the command specifies the broadcast address of a subnet, so that the request can be sent to all the machines in that subnet.

3.3.4 ARP Spoofing

ARP Spoofing is an attack where a malicious device modifies the victim's local ARP cache table so that another machine's IP address is associated with the wrong MAC address [11]. Generally the MAC address to be inserted belongs to the attacker, allowing for the attacker to intercept data going from the victim to that IP address. We run the attack using the Kali Linux tool "arpspoof", which is designed specifically for ARP poisoning/spoofing. The command to run the attack is "arpspoof -i eth0 -r -t [victim-ip] [gateway-ip]". The -i flag specifies the interface to run on, which in our case is "eth0". The -r flag specifies that the attack should poison the ARP cache tables of both of the IP addresses specified, allowing for the attacker to intercept all data going between the two devices. The -t flag specifies the victim to poison and the last IP address is the IP we want associated with our MAC address in the victim's ARP table. We ended up taking out the -r flag for the purposes of our experiments, since poisoning the gateway's ARP table would have taken too long. We ended up using the command "arpspoof -i eth0 -t [victim-ip] [gateway-ip]" to poison the victim server's ARP table with the gateway's IP address and the attacker's MAC address.

3.3.5 DNS Spoofing

DNS Spoofing is an attack where an attacker impersonates a DNS server, either by setting up a rogue DNS server or by intercepting DNS requests and trying

to reply before the real DNS server can [11]. For our experiment we use the latter method, intercepting DNS requests from the victim and responding with a forged DNS response. We also use ARP poisoning to make sure the request is never sent to the real DNS server, causing the victim to always receive the forged DNS response. For the ARP poisoning, we use the same command as we just described in section 3.3.4. Since the DNS server we use is outside of our physical host's subnet, all DNS request packets are sent to the gateway to forward to the DNS server. ARP spoofing causes the victim to send all data destined for the gateway to the attacker, meaning the real DNS server will never receive the DNS requests from the victim.

We run the DNS Spoof attack with the Kali Linux tool "dnsspoof". The dnsspoof tool takes a file specified by the user, saying which websites to spoof and which IP address to spoof it with. Once it is run, it starts sniffing for DNS request packets that are querying for any website that is contained in the file. If one is found, dnsspoof generates a DNS response packet for that website with the spoofed IP address specified in the file. The command to run the DNS Spoof attack is "dnsspoof -f [filename]", where the -f flag specifies the filename. The format of the file is "[ip-address] [url]...", where each line contains an IP address/url pairing. For our experiments, the spoofed IP address is always set to the IP address of the attacker.

3.3.6 NMAP Scanning

NMAP is a tool that uses raw IP packets to gain information about other machines, such as whether they are up or not, ports, operating system, etc [33]. It is generally utilized to gain information about a subnet or network. We use the NMAP tool to run a basic scan of a single victim, getting information on whether the machine is up, which ports are open, and the machine's MAC address. The command to run the nmap scan is "nmap [victim-ip]". We run the command on one

victim host to simplify the attack and make the scanning faster for the purposes of the experiments.

3.4 Detection

To make our experiments more realistic, there needs to be a way to detect the network attacks being run. Once they are detected, the dynamic responses can be used to try to protect the virtual environment from the attacks. To detect these attacks, we use an Intrusion Detection System (IDS), which can catch attacks based on packet signatures. For each attack, a rule is created in the IDS to catch it. We create a headless, mini Ubuntu VM similar to the server and client. It is given 8 GB for a hard disk and 1 GB of memory. Snort is then installed on the VM. In this section, we describe the creation of the IDS, the setup done, and the rules created for each attack. We begin with the IDS creation.

For our IDS we use Snort, an open source intrusion prevention system that does real-time analysis of packets and logs information [49]. Though Snort has the capability for attack prevention, it is used in our system purely for detection. Setup of Snort is simple and can be easily replicated. Because Snort is so widely available, it can be downloaded with an "apt-get install snort" command. Everything is set to the defaults, with the only edits being in the configuration and rules files. The "snort.conf" file is used for configuration of the Snort detection engine, while the "local.rules" file contains detection rules similar to a firewall. In the configuration file, all the automatically included rules are removed, leaving only the path to our rules file. We set the configuration file to log all alerts in plaintext. Then, in the rules file, the rules for detecting attacks are placed. Once everything is set up, Snort is started by running the "service" command. If any changes are made to any files affecting Snort, they only take effect once the IDS is restarted using the "service" command.

To detect the attacks for our experiments, we need to create detection rules so that Snort can send an alert that an attack has been detected. Figure 24 shows our Snort rules in "local.rules". Each rule creates a set of criteria that a packet or flow must meet for it to be considered an attack. If that criteria is met, an alert is logged to the file "alert" and the packet is stored in libcap format in a file "tcpdump.[random number]", where "[random number]" is replaced with some randomly generated number. For the experiments, only one rule is ever used at a time to lower overhead. The first 3 rules correspond to the TCP DOS, ICMP DOS, and Smurf attacks. All 3 rules are very similar with only slight differences. The 3 rules specify that if any one source sends more than 1000 packets per second of the specified packet type, either tcp or icmp, then it is considered a DOS attack and an alert is created. The 4th rule is to detect NMAP scan attacks. It uses a set classification, built into Snort, for what a scan attack is and the "{}" is replaced with the attacker's IP address. This is done to avoid detecting numerous other machines in the system that must scan the network. Due to the way the system is set up, it is not possible to put a range of IP addresses, because it most likely would include a VM that needs to scan the Virtual Network environment. In a realistic system, the "{}" could be replaced with a range of IP addresses of machines that shouldn't need to scan the network. The 5th rule is used to detect DNS Spoof attacks and was taken from an existing rule set. The rule checks to see if any packet with an IP address outside of the internal subnet and using port number 53, the DNS port, is sent to an address in the internal subnet. If the packet matches that criteria, the body of the packet is checked for the contents given in the rule. The contents help differentiate between a real DNS packet and a fake one. If the packet's contents match the rule, an alert is sent out.

Since ARP packets are at layer 2 of the OSI model, while the rest of the rules check packets at layer 3, the ARP rules require being checked earlier in Snort's

```
#alert tcp any any -> any any (msg: "DOS TCP Flooding Attack"; flags: S;
        classtype:successful-dos; detection_filter: track by_src, count 1000, seconds 1; sid:1000001)
#alert icmp any any -> any any (msg: "DOS ICMP Flooding Attack";
        classtype:successful-dos; detection_filter: track by_src, count 1000, seconds 1; sid:1000005)
#alert icmp any any -> any any (msg: "DOS Smurf Attack";
        classtype:successful-dos; detection_filter: track by_src, count 1000, seconds 1; sid:1000006)
alert tcp {} any -> $HOME_NET any (msg: "NMAP Scan Attack";
        classtype:network-scan ; sid:1000008)
#alert udp $EXTERNAL_NET 53 -> $HOME_NET any (
        msg:"DNS Spoof Attack"; content:"|81 80 00 01 00 01 00 00 00 00|";
                content:"|C0 0C 00 01 00 01 00 00 00|<|00 04|"; classtype:bad-unknown; sid:254; rev:4;)
```

Figure 24: Snort Rules for Attack Detection

```
# ARP spoof detection.  For more information, see the Snort Manual -
preprocessor arpspoof: -unicast
preprocessor arpspoof_detect_host: 10.144.202.1 d0:67:e5:d7:a5:59
preprocessor arpspoof_detect_host: 10.144.202.120 00:50:56:97:95:82
```

Figure 25: Snort Rules for ARP Spoofing Detection

system. To check ARP packets, a preprocessor rule is used to check the packet before it is sent to Snort's rules engine [4]. The preprocessor rules are created in the configuration file, under the preprocessor section. Figure 25 shows the ARP preprocessor rules. The ARP rules set IP-MAC pairings that the preprocessor should check on. If an IP or MAC that is listed in the pairings is found with a different IP-MAC pairing, Snort sends an alert that there is a mismatch. The "-unicast" rule detects if there are any unsolicited ARP replies being sent, and if any are detected an alert is sent out. Currently the preprocessor alerts don't give any information on IP or MAC when ARP Spoofing is detected, since it is not yet officially supported. We assume that the in the future, Snort's ARP Spoofing detection will list the attacker's IP address in the alert, since the spoofed ARP packet has the malicious node's IP address anyways.

For the experiments, the created logs are retrieved to confirm that the attack is detected. For all the attacks other than the DNS Spoof and Smurf attacks, only the alert file is needed to confirm the attack and get the attacker's IP address. For

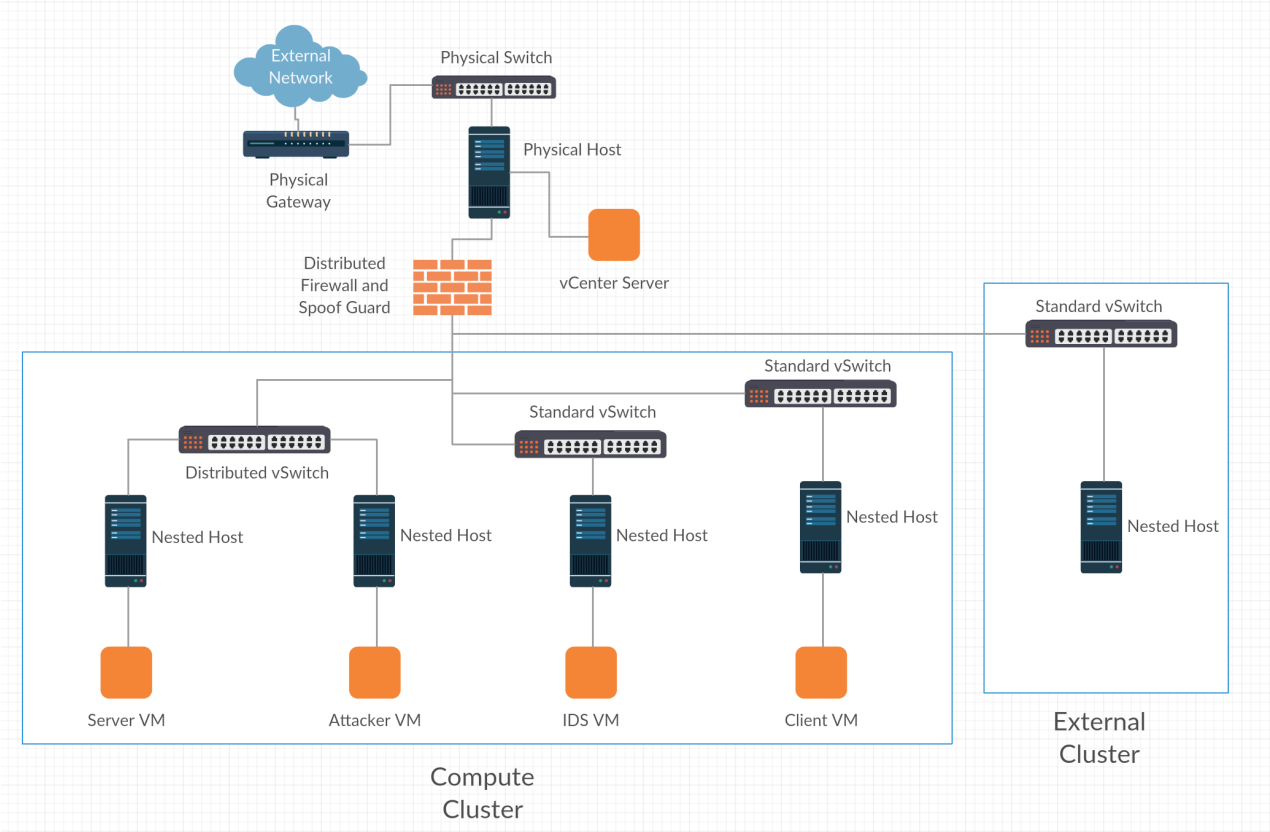

Figure 26: Virtual Network Overview with Nested Hosts and VMs

DNS Spoof detection, the alert file is used to confirm the attack occurred and the "tcpdump.[random number]" file is used to get the spoofed IP address contained in the malicious DNS reply packet. To convert the 'tcpdump.[random number]" file to plaintext, the command "tcpdump -r [filepath]" is used and the output is redirected to a file. The spoofed IP address of the url is then retrieved from the output and used to get the attacker's IP address. For the Smurf attack, since the IP address is spoofed, we assume that the attacker's IP address is determined through other methods. Note that the attacker's IP address is only needed for the experiments where the attacker is moved rather than the server.

Chapter 4

DYNAMIC DEFENSE DESIGN IN VIRTUAL NETWORKS

Now that we have described our system setup, we move on to the design of the dynamic defenses. To test the validity of our theory that Virtual Network flexibility can be used for network security purposes, we create a set of experiments. These experiments set up different scenarios and utilize different dynamic defenses. In this chapter we look at the dynamic defenses, how they are set up, and how the experiments are run. We divide this chapter into two sections: Dynamic Defenses and Measurements. In section 4.1, we describe the four dynamic defenses, as well as list the topologies used and the steps taken in running each experiment. In section 4.2, we describe how we quantify the success of a dynamic defense against each of the 6 network attacks used in our experiments.

4.1 Dynamic Defenses

Our defenses are divided into two groups of general defenses, Server Protection and Attacker Prevention, each of which utilize different virtual security components and scenarios. The Server Protection defense detects an attack targeting a server and moves that server to a protected location. Attacker Prevention defenses detect a malicious node and shift the attacker and topology to protect network. Each group of defenses has a set of experiments that are run for every attack (excluding one group of experiments that only runs the MITM attacks) and are run independently of each other. In this section, for each defense and the corresponding experiments, we describe the dynamic defense, the topologies used, the attacks used, and list the individual steps for each experiment. We also explain the real world scenario that each experiment is trying to replicate. We start with the Server Protection experiments.

4.1.1 Server Protection

Server Protection is a defense where an attack is detected against a server and the topology is shifted to protect that server. In this situation there is an insecure subnet and a secure subnet. Perhaps it is too resource intensive to put protections on the whole network, so only a subset of the network has heavier attack protections. The insecure subnet is a standard port group on a standard switch, while the secure subnet is a distributed port group with protections in the form of rules on the distributed firewall and spoof guard policies. The setup is such that the attacker is in an external/separate network or isn't yet identified and thus can't be touched. Once an attack is detected, the server is moved to the secure subnet to protect against the attacker and any future attacks. We test this defense with a set of experiments that vary based on the attack and the defense, which is based on the attack type, used. In this section, we first describe the topology used, followed by the defenses for each attack, and finally the steps for running the experiment.

For the experiments for the Server Protection defense, only one topology is used, containing one client, attacker, IDS, and server. Figure 27 shows the topology. The attacker is placed on standard host 5 in the external cluster, which represents the "external network" or "separate network". The server and the IDS are placed on standard host 1 in the compute cluster or the "internal network", with the IDS being connected to a different standard port group than the server. For the DOS attacks, a client is placed on standard host 2 in the compute cluster. Each VM is connected to an independent standard port group. The secure subnet, which the server is moved to in the experiment, is a distributed port group called "ProtectedPG" on distributed host 3 and is connected to a distributed vSwitch.

How the secure subnet is protected varies based on the attack. For the distributed port group there are two possible types of protection, distributed

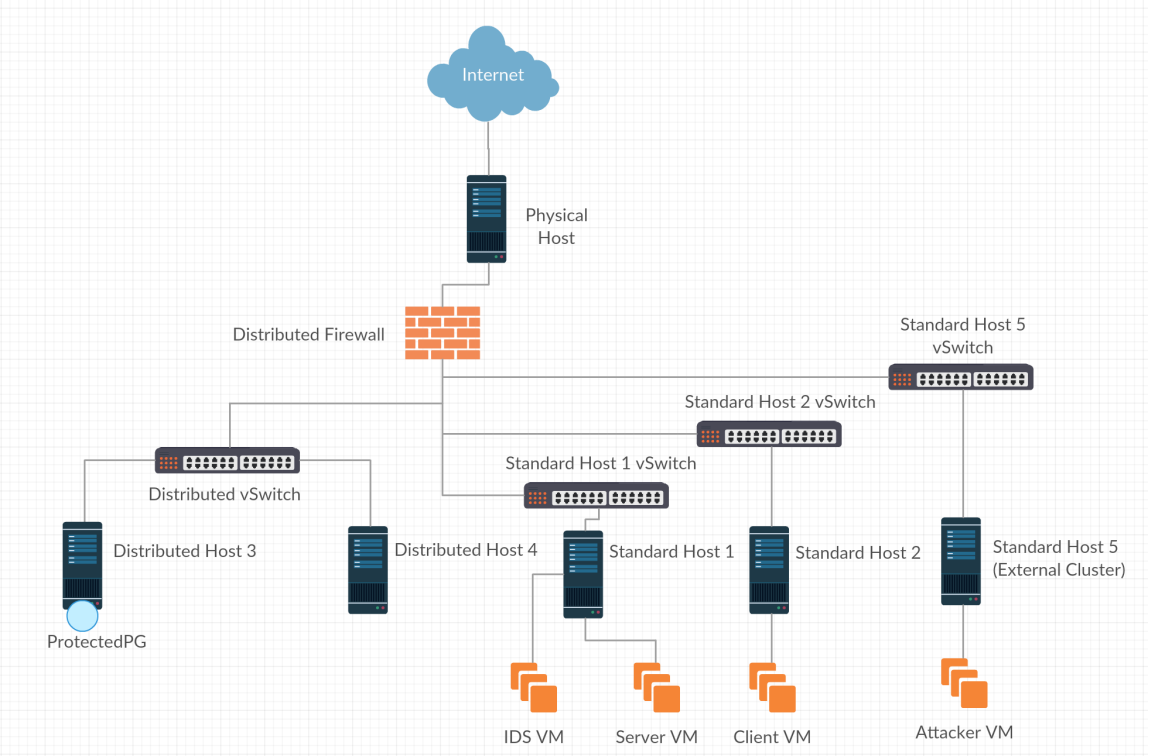

Figure 27: Topology for Server Protection Experiments

firewall rules and spoof guard. We go through each attack and describe which protections are used and how they are set up.

1. TCP DOS attack

   To protect against the TCP DOS attack, the distributed port group "ProtectedPG" and the distributed firewall are used. To add protection to "ProtectedPG", we create a firewall rule saying that any TCP packet going from the external cluster to the port group is dropped. Figure 28 shows the firewall rule. There is the possibility that there are clients in the external network who need to connect to the server. In this case, firewall rules can be created to allow verified users/clients to connect to the server using a whitelist. Once the server is moved to the secure subnet, the attack should be prevented, allowing clients to create connections with the server uninterrupted.

5	ICMP DOS Protection	1011	external-cl...	ProtectedPG	ICMP...	Block	Dist...
6	ICMP Smurf Protection	1015	10.144.20...	ProtectedPG	ICMP...	Block	Dist...
7	NMAP Scan Protection	1013	external-cl...	ProtectedPG	ICMP... TCP ...	Block	Dist...
8	TCP Syn Dos Protection	1012	external-cl...	ProtectedPG	TCP ...	Block	Dist...
9	Default Rule	1001	* any	* any	* any	Allow	Dist...

Figure 28: Firewall Rules for Protection from ICMP Dos, TCP Dos, Smurf and NMAP Scan Attacks

1	ARP Spoof Protect...	1014	external-cluster	ProtectedPG	Arp Spoof	Block	Distribut...
2	Default Rule	1004	* any	* any	* any	Allow	Distribut...

Figure 29: Firewall Rule for ARP Spoofing Prevention

2. ICMP DOS and Smurf attack

   Protecting against both the ICMP DOS and Smurf attacks is very similar to the TCP DOS attack, but with slightly different firewall rules. The distributed port group "ProtectedPG" is protected by a firewall rule saying that any ICMP packet going from the external network to the port group is dropped. A rule is also created saying ICMP packets can not be sent from the network subnet to the distributed port group to protect against the Smurf attack. Figure 28 shows both rules in the distributed firewall. Unlike with TCP, it isn't a big concern if an external network can not send ICMP packets to an internal network. However, if it is needed for some reason a whitelist can be used. Once the server is moved to the secure subnet, the attack should be protected against, allowing clients to create connections with the server uninterrupted.

3. ARP Spoof attack

   To protect against the ARP spoof attack, a new protection must be used for the distributed port group "ProtectedPG". Using a firewall rule targeting the

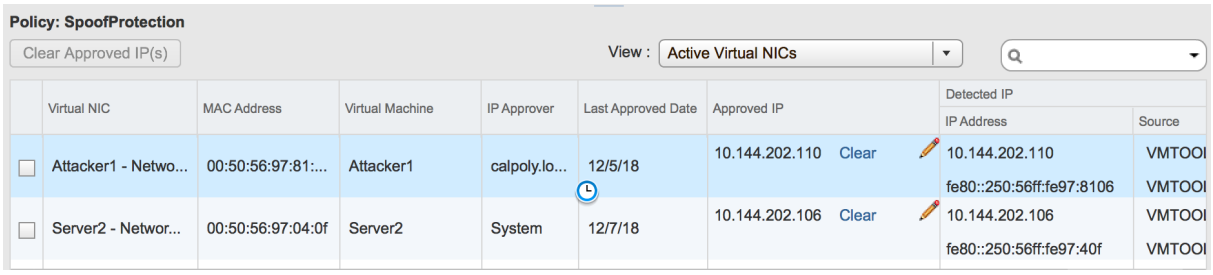

Figure 30: Spoof Guard Protection for DNS Spoof and ARP Spoof Attacks

external network specifically wont always help protect against spoofing, because a firewall rule relies on the information in the packet being correct. Spoofing would be able to get around most rules because the packet being sent has incorrect information. To protect against this, it is necessary to use the distributed firewall's spoof guard. A policy can be created with spoof guard to protect the "ProtectedPG" port group from receiving packets with spoofed IP addresses, MAC addresses, etc. Figure 30 shows the spoof guard being applied to the distributed port groups and the VMs in those port groups. We also create a firewall rule preventing ARP packets from be sent from the external-cluster to the server, which is shown in figure 29. Once the server is moved to the secure subnet, it should be protected from ARP poisoning, preventing the server's data from being intercepted by the attacker.

4. DNS Spoof attack

   Protecting against the DNS spoof attack is the same as protecting against ARP spoofing. Spoof guard is used to protect the "ProtectedPG" distributed port group because a firewall rule would be ineffective against a DNS Spoofing attack. With spoof guard protecting the secure subnet, the DNS Spoof attack should fail, protecting the server from illegitimate url information.

5. NMAP Scan attack

To protect against the NMAP scan attack, firewall rules are used to protect the distributed port group "ProtectedPG". Because the basic NMAP scan uses ICMP and TCP packets to gain information, firewall rules are created to drop any ICMP and TCP packets going from the external network to the distributed port group. Figure 28 shows the firewall rule. Like the DOS attack protections, if certain clients in the external networks need to use those protocols, a whitelist can be used. With the protections on the secure subnet, the server should be protected from scanning once it is moved.

We now list the steps of the experiment:

1. Set up the topology as described above
2. Select one of the attacks listed for this experiment
3. Set IDS rule for detecting the attack
4. Set firewall rule for protecting against attack to "block" that type of traffic
5. Using the measurement procedure for the selected attack, as described in section 4.2, run the pre move and attack (if is DOS attack) measurements
6. Check if attack is detected before running post move measurements
7. If attack is detected, move server to distributed host 3 and connect the server to the protected distributed port group "ProtectedPG"
8. Finish measurement procedure by recording post move measurements
9. Clean up by moving server back to original host and standard port group
10. Write measurements to file, save average for each measurement type
11. Sleep for five seconds

12. Repeat steps 1-11 20 times

13. Get average for all 20 runs for each measurement type and write it to file

14. Repeat for all attacks

4.1.2 Attacker Prevention

Attacker Prevention is a defense where a malicious node is detected and then dynamically contained. For each example of attacker prevention, a secure subnet is created and the attacker is moved to it to prevent any further attacks. The setup of each experiment is such that the attacker is in the internal network and can be moved around. Once an attack is detected, the attacker is moved to the secure subnet to stop the attack. There are 3 different types of experiment groups under Attacker Prevention, each named by the dynamic defense used: Isolated Subnet, Distributed Port Group, and Standard Port Group. For each one, we describe the topology used, followed by the attacks run and the corresponding defenses, and then the steps for running each experiment. We start with the Isolated Subnet experiments.

4.1.2.1 Isolated Subnet

Isolated Subnet is a defense where a malicious node is detected and moved to a port group that is isolated from the rest of the network, similar to a black hole. This isolated subnet is a standard port group that is given a unique VLAN so that it cannot communicate outside of its port group. Because the attacker is the only VM in the standard port group, it can't communicate with any other device. Note that the assigned VLAN must be a unique value not contained in any other port group using VLAN trunking. The defense is unrelated to attack type, since being cut off from communication should stop any attack. In the rest of this section, we

describe the topologies used, list the attacks and the defense, and describe the steps for the experiment for this defense.

For this defense, two topologies are used, each containing one client, attacker, IDS, and server. Figure 31 shows the two Isolated Subnet topologies. In the first topology, all the VMs are contained in the compute cluster, or the "internal network". The attacker and client are placed on standard host 2 in the compute cluster, while the server and IDS are placed on standard host 1, also in the compute cluster. Each VM is on an independent standard port group, even if they are on the same host. The isolated subnet which the attacker is moved to is a standard port group that is dynamically created on the same host and standard vSwitch as the attacker. The newly created standard port group is given a random VLAN to isolate it from the rest of the world. The second topology is almost identical to the first, except the client is moved from the compute cluster to standard host 5 on the external cluster or "external network". The second topology is only used for the experiments using DOS attacks, as only those experiments require the use of a client. The isolated subnet is the same for both topologies.

The following attacks are used in this experiment and all are defended by the isolated subnet in the same way:

1. TCP DOS attack
2. ICMP DOS attack
3. Smurf attack
4. ARP Spoof attack
5. DNS Spoof attack
6. NMAP Scan attack

We now list the steps of the experiment:

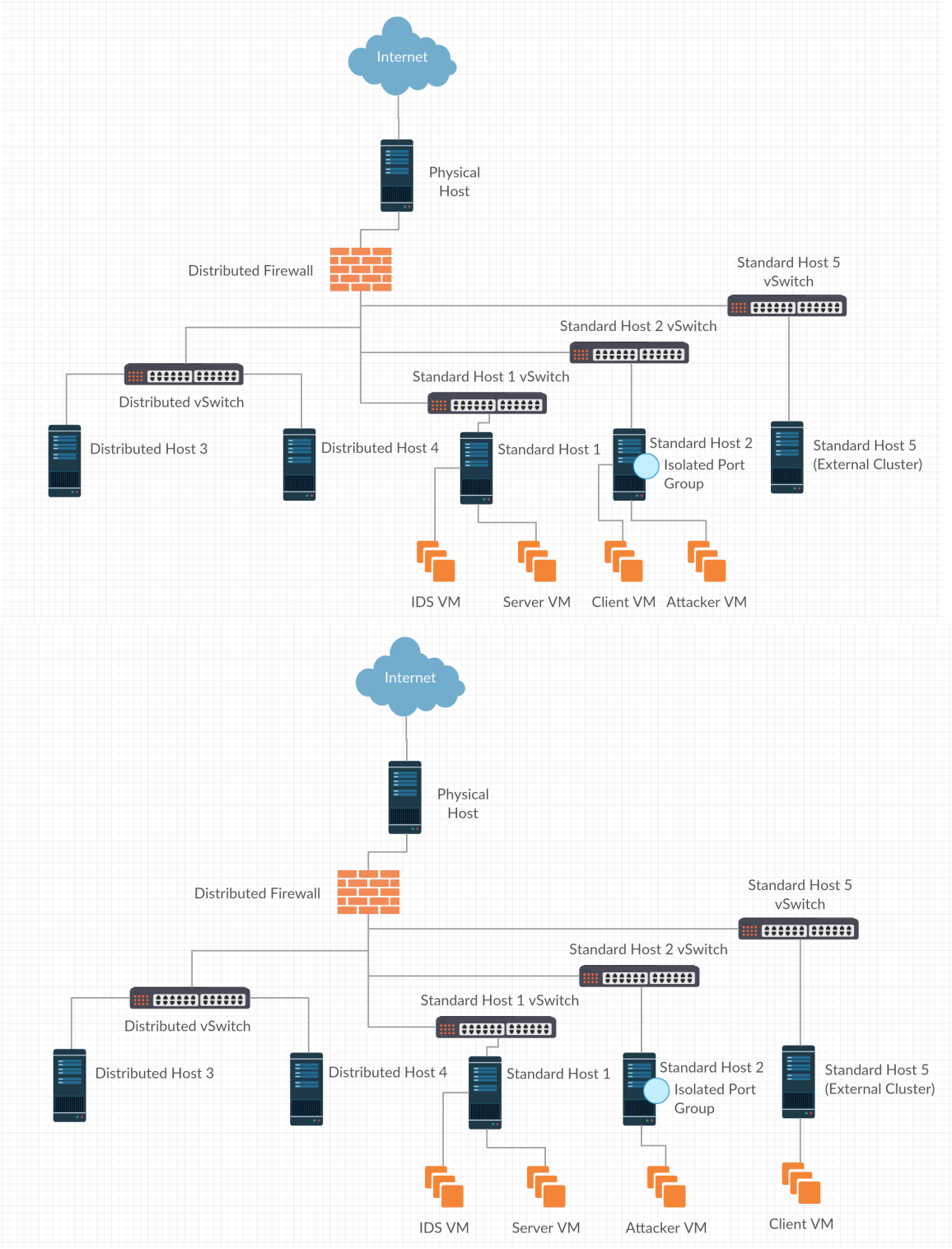

Figure 31: Topologies for Isolated Subnet Experiments

1. Set up the first topology as described above
2. Select one of the attacks listed for this experiment
3. Set IDS rule for detecting the attack
4. Using the measurement procedure for the selected attack, as described in section 4.2, run the pre move and attack (if is DOS attack) measurements
5. Check if attack is detected before running post move measurements
6. If attack is detected, create isolated subnet
7. If isolated subnet is created, connect attacker to it
8. Finish measurement procedure by recording post move measurements
9. Clean up by moving attacker back to original standard port group
10. Write measurements to file, save average for each measurement type
11. Sleep for five seconds
12. Repeat steps 1-11 20 times
13. Get average for all 20 runs for each measurement type and write it to file
14. Repeat for the second topology if selected attack is of type DOS
15. Repeat for all attacks

4.1.2.2 Distributed Port Group

Distributed Port Group is a defense where a malicious node is detected and moved to a distributed port group with protections on it. The distributed port group utilizes the distributed firewall and spoof guard for this attack prevention. The defense is almost identical to the Protected Server defense. The key difference

is that the firewall rules and spoof guard target the port group of the attacker rather than the server. In the rest of this section, we describe the topologies used, the attacks and the defenses used against them, and then describe the steps for the experiment.

For this defense, two topologies are used, each containing one client, attacker, IDS, and server. Figure 32 shows the two Distributed Port Group topologies. In the first topology, all the VMs are contained in the compute cluster, or the "internal network". The attacker is placed on distributed host 4 and is connected to a distributed port group, which is connected to the distributed vSwitch. The server is placed on distributed host 3 and is connected to a different distributed port group, which is also connected to the distributed vSwitch. The attacker and the server are on different distributed port groups and neither is connected to the protected distributed port group. Similar to previous experiments, the IDS is on standard host 1 and the client is on standard host 2, both connected to different standard port groups in the compute cluster. The protected subnet which the attacker is moved to is a distributed port group called "DPG2" and is on the same distributed vSwitch as the attacker and server. The second topology is almost identical to the first, except the client is moved from the compute cluster to standard host 5 on the external cluster or "external network". The second topology is only used for the experiments using DOS attacks, as only those experiments require the use of a client. The protected distributed port group is the same for both topologies.

How the distributed port group is guarded depends on the attack, similar to the Server Protection experiments. For the distributed port group, two types of protection are used, the distributed firewall rules and spoof guard. We go through each attack and describe which protections are used and how they are set up.

1. TCP DOS attack

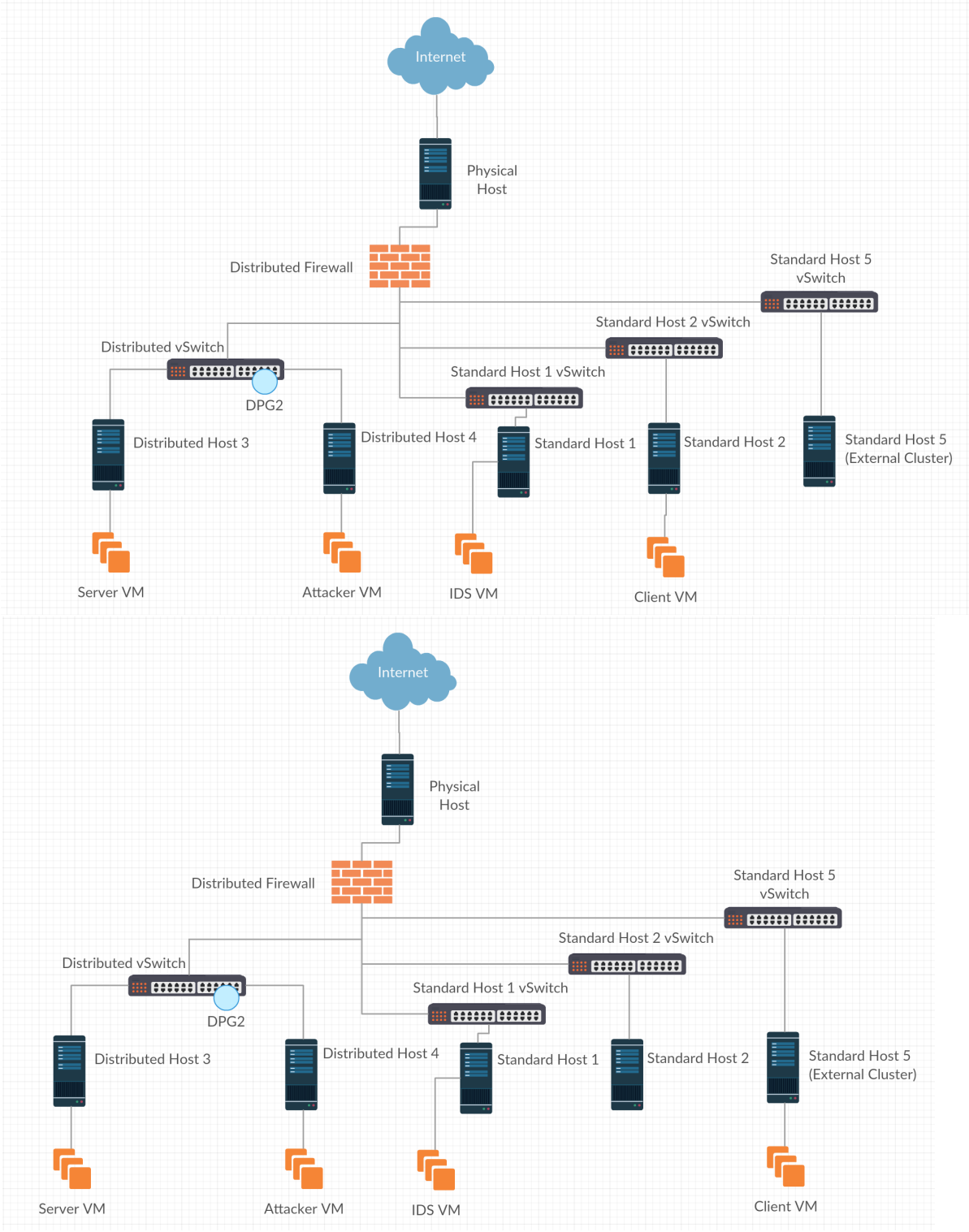

Figure 32: Topologies for Distributed Port Group Experiments

Default Section Layer3 (Rule 1 - 9)							
1	ICMP DOS Prevention	1005	DPG2	* any	ICMP Echo	Block	Dis...
2	TCP Syn DOS Prevention	1007	DPG2	* any	TCP Syn Protection	Block	Dis...
3	NMAP Scan Prevention	1010	DPG2	* any	ICMP Echo TCP Syn All	Block	Dis...

Figure 33: Firewall Rules for Prevention of ICMP Dos, TCP Dos, and NMAP Scan Attacks

To prevent the TCP DOS attack from happening, a protected distributed port group "DPG2" is created. To protect "DPG2", a firewall rule is created saying that any TCP packet originating from the the distributed port group is dropped. Figure 33 shows the firewall rule. Once the attacker is moved to the distributed port group, the attack should be prevented, allowing clients to create connections with the server uninterrupted.

2. ICMP DOS and Smurf attack

   Preventing both the ICMP DOS and Smurf attacks is very similar to the TCP DOS attack, but with slightly different firewall rules. The distributed port group "DPG2" is protected by a firewall rule saying that any ICMP packet originating from the distributed port group is dropped. A rule is also created saying ICMP packets can not be sent from the distributed port group to the broadcast IP address of the network to protect against the Smurf attack. Figure 33 shows both rules. Once the attacker is moved to the distributed port group, the attack should be protected against, allowing clients to create connections with the server uninterrupted.

3. ARP Spoof attack

   To prevent the ARP spoof attack, a new protection must be used for the protected distributed port group "DPG2". As discussed earlier, spoofing can get around the distributed firewall rules. Figure 30 shows the spoof guard policy used

on the distributed port group. Spoof guard is used to prevent any VM connected to the distributed port group from spoofing its IP or MAC address. Once the attacker is moved to the distributed port group, it should be protected from ARP poisoning, preventing the server's data from being intercepted by the attacker.

4. DNS Spoof attack

   Protecting against the DNS spoof attack is the same as protecting against ARP spoofing. Spoof guard is used to prevent any VM connected to the distributed port group "DPG2" from spoofing addresses. With spoof guard on the distributed port group, the DNS spoof attack should fail, protecting the server from illegitimate url information.

5. NMAP Scan attack

   To prevent the NMAP scan attack, firewall rules are used to prevent scanning behavior. Because scanning uses ICMP and TCP packets to gain information, firewall rules are created to block ICMP and TCP packets coming from the distributed port group. Figure 33 shows the rule. With the protections on the distributed port group, the attacker should be prevented from scanning once it is moved.

We now list the steps of the experiment:

1. Set up the first topology as described above
2. Select one of the attacks listed for this experiment
3. Set IDS rule for detecting the attack
4. Set firewall rule for protecting against attack to "block" that type of traffic
5. Using the measurement procedure for the selected attack, as described in section 4.2, run the pre move and attack (if is DOS attack) measurements

6. Check if attack is detected before running post move measurements
7. If attack is detected, move attacker to protected distributed port group
8. Finish measurement procedure by recording post move measurements
9. Clean up by moving attacker back to original distributed port group
10. Write measurements to file, save average for each measurement type
11. Sleep for five seconds
12. Repeat steps 1-11 20 times
13. Get average for all 20 runs for each measurement type and write it to file
14. Repeat for the second topology if selected attack is of type DOS
15. Repeat for all attacks

4.1.2.3 Standard Port Group

Standard Port Group is the defense with the smallest attack coverage. It is a defense where a malicious node is detected and moved to a standard port group with spoof protections on it. The port group is created with promiscuous mode and MAC spoofing set to reject, theoretically preventing spoofing attacks. Since the defense is spoof based, only the MITM attacks are used for this experiment. We test this defense with a set of experiments that are the same for all the attacks. In the rest of this section, we describe the topologies used, list the attacks and the defense, and describe the steps for the experiment.

For this experiment only one topology is used, containing an attacker, IDS, and server. Figure 34 shows the Standard Port Group topology. No client is needed, since all the attacks are MITM attacks. In the topology, all the VMs are contained in the compute cluster, or the "internal network". The attacker is placed on

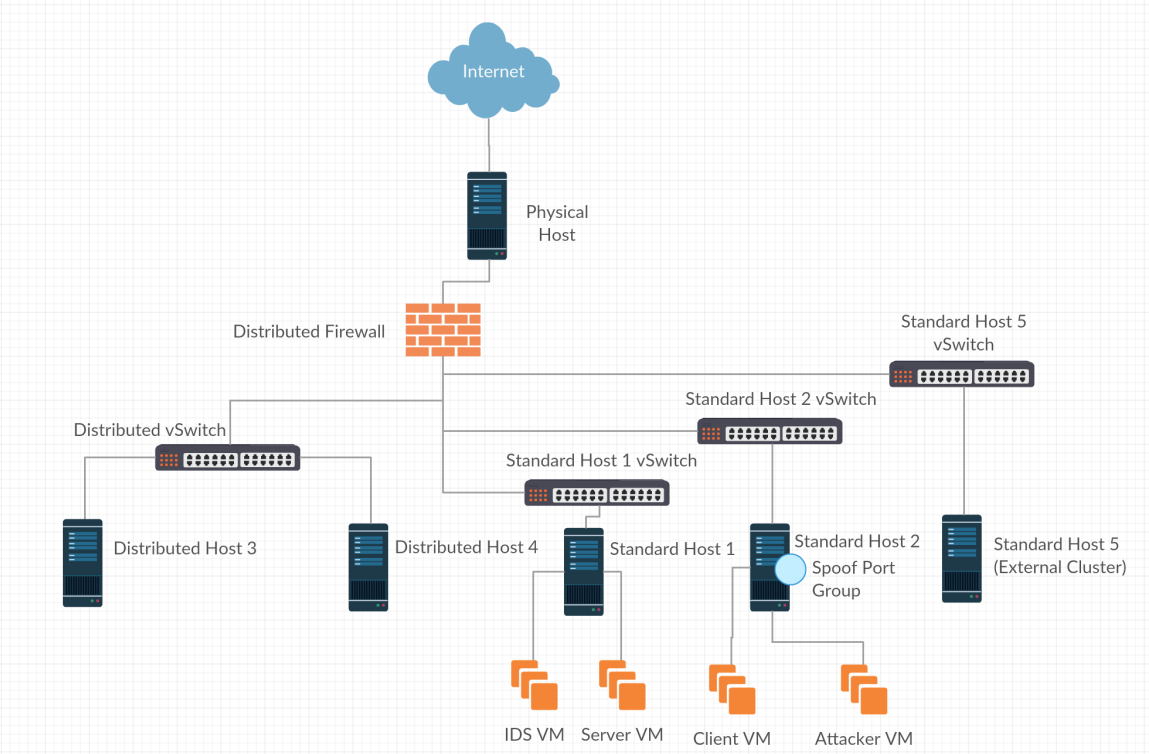

Figure 34: Topologies for Standard Port Group Experiments

standard host 2 in the compute cluster, while the server and IDS are placed on standard host 1, also in the compute cluster. Each VM is on an independent standard port group, even if they are on the same host. The protected subnet which the attacker is moved to is a standard port group that is dynamically created on the same host and standard vSwitch as the attacker. The standard port group is created with promiscuous mode and MAC spoofing set to reject rather than accept.

The following attacks are used in this experiment:

1. ARP Spoof attack

2. DNS Spoof attack

We now list the steps of the experiment:

1. Set up the topology as described above

2. Select one of the attacks listed for this experiment
3. Set IDS rule for detecting the attack
4. Using the measurement procedure for the selected attack, as described in section 4.2, run the pre move measurements
5. Check if attack is detected before running post move measurements
6. If attack is detected, create standard port group with promiscuous mode and MAC spoofing set to reject
7. If standard port group is created, connect attacker to it
8. Finish measurement procedure by recording post move measurements
9. Clean up by moving attacker back to original standard port group
10. Write measurements to file, save average for each measurement type
11. Sleep for five seconds
12. Repeat steps 1-11 20 times
13. Get average for all 20 runs for each measurement type and write it to file
14. Repeat for all attacks

4.2 Measurement

To be able to compare the effects of the dynamic security measures, it is necessary to have a way to numerically determine how successful an attack is. What determines success varies by attack and thus the way we measure success must be determined for each attack type. The Denial of Service attacks are all similar enough that they can be measured in the same manner. However, the rest of the

attacks are different enough that they each need to be run differently and therefore measured differently. In this section, we look at how each attack is set up, run, and quantified in our experiments. Note that what we describe for measuring each attack (even when saying that it is repeated) is only considered one run of an experiment. Each run is repeated multiple times for each experiment as we explain in section 4.1. We first look at quantification of success for the DOS/DDOS attacks.

4.2.1 DOS/DDOS Attacks

In this section, we first discuss the setup for the DOS attack experiments and how a measurement is derived from running the attacks. Then, we explain the repetition used to create the dataset, which is considered one run of an experiment.

Our DOS attacks are all run in a similar manner and have the same goal. Because of this, we are able to use the same set up and measurement for all three attacks. Section 3.3 shows how the three DOS attacks are run using Kali Linux tools. To measure the effects of the DOS attacks, we utilize a client, a server (victim), and an attacker. Since the goal of the DOS attacks are to slow down or deny access to the server's resources, we can determine success by measuring the time taken to get a resource from the server. To measure attack success, we determine the time in seconds it takes for the client to download an approximately 32 MB file (filled with random characters) from the server. If for some reason the download can not be completed, the time taken is set to 100 seconds. On average a normal download of the file for a client takes about 0.5 seconds, so 100 seconds is relatively large. This will raise the average download time significantly to show that the download was stopped, but not move it to an unnecessarily large value.

To run and time the download, we use two bash scripts placed on the client. Putting the commands in bash scripts allows the download to be triggered remotely through the command line. Figure 35 shows the two bash scripts. The first script is

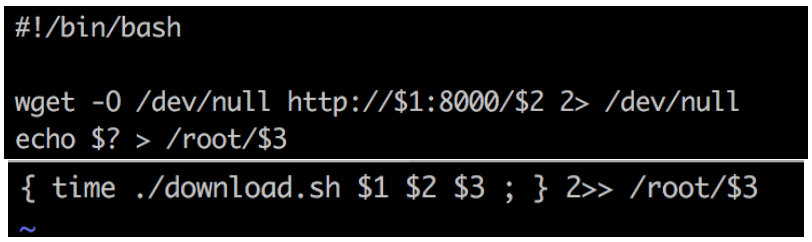

Figure 35: Client's Bash Scripts

used to download the file using wget and check whether the download was successful. It takes three variables: the IP address of the server, the name of the file to download from the server, and the name of the file to store the download's success. The second script is used to time the download, using the time tool, and takes the same three variables as the first script. Only the second script needs to be run, since it calls the first script. Both scripts reside on the client and the second script is run to cause the client to download a file from the server and time it. The success and time taken are output to a file on the client. The information is then remotely retrieved from the client.

To create the dataset for one run, we first measure the time taken for the client to download the file from the server without any attack running. This is called the "baseline" or "pre move" measurement and is repeated 10 times. Then, the DOS attack is run targeting the victim server and again we measure the time for the client to download the server's file. This is called the "attack" measurement and again is repeated 10 times. Finally, the dynamic defense takes place and the DOS attack is run again while the client downloads the file from the server. The measurement, called the "post move" measurement, is then repeated 10 times. The pre move, attack, and post move measurements are each retrieved from the client and averaged to get 3 different, average download times. The pre move averages serve as a baseline for how fast a normal transmission of data is between the client and the server. This is then used to compare to the attack and post move averages

to see how effective the dynamic defense is. We further explain how this comparison is used to determine effectiveness in section 5.

4.2.2 MITM/Scan Attacks

The measurement of the MITM and Scan attacks vary by attack, unlike the DOS attacks which all are measured in the same manner. Despite the differences in how each attack is run and quantified, the range of values for the MITM/Scan attacks is always between 0 and 1. A 1 signifies a completely successful attack and a 0 signifies a failure. A decimal value between 0 and 1 means an attack is partially successful. All the MITM and Scan attack experiments require only an attacker and a server, with no client necessary. In the rest of this section, we look at how we quantify the success rate of the ARP spoof, DNS spoof, and NMAP Scan attacks. We first describe the setup and how each attack success is measured, followed by how the measurements are repeated to constitute one run. We start with describing the ARP Spoofing quantification.

4.2.2.1 ARP Spoofing

Measurement of ARP spoofing requires only the use of a server and an attacker, as stated above, with no client needed. Section 3.3.4 explains how the attack is run. The goal of the attack is to successfully poison the server's ARP cache table. This is done so that data sent from the server to the subnet's gateway is instead sent to the attacker. Originally, we planned to try to poison the ARP table of the gateway as well. However, poisoning the gateway of the subnet was not possible due to time constraints on our experiments. Initially, the measurement would be derived from ARP poisoning the server and using tcpdump to confirm that the data was being sent to the attacker instead of the gateway. However, we

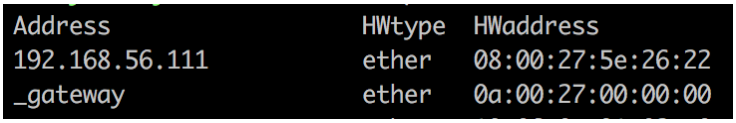

Figure 36: Example of ARP Table Output for Victim Server

realized that since we have access to the server as well, it would be easier to simply check the ARP table for the forged MAC address.

To measure the attack, we look to see if the gateway's IP address is in the ARP table containing the attacker's MAC address. Figure 36 shows example output of a server's ARP table. The command "arp" is run on the victim server and the output is redirected to a file. The ARP table contents are then retrieved from the file to check for success. If the MAC address in the gateway's entry is the attacker's MAC, then the attack succeeded and is given a value of 1. Otherwise, it failed and it is given a value of 0. To create the dataset, we first run the attack before the dynamic defense is initiated and measure the outcome. This is called the "baseline" or "pre move" measurement, which is then repeated 10 times. Once the dynamic defense is initiated, the attack is run and measured again. This is called the "post move" measurement and is also repeated 10 times. The pre move and post move measurements are then averaged and are considered one run of an experiment.

4.2.2.2 DNS Spoofing

DNS spoofing requires similar measurement to ARP spoofing, but is run differently. The goal of the attack is to make a machine think that a url is associated with the attacker's IP address. Section 3.3.5 explains how this attack is run. At first we attempted to run the DNS spoof without using ARP poisoning. This proved difficult, because if the DNS request is still sent to a real DNS server the attacker must respond to the request before the real server. With cached urls this rarely happens. Even when we tried finding obscure urls, requesting them too

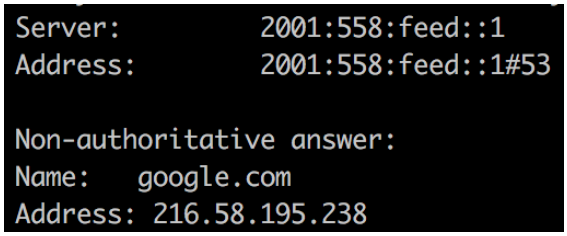

Figure 37: Example Nslookup Output

many times would get them cached locally, making it very uncommon for our attack to beat the real DNS server. We ended up settling on running the ARP spoof attack before running the DNS spoof attack. This prevents the DNS request from reaching a real DNS server. We also decided to completely randomize the urls sent by the victim server, making the process of selecting urls automated. This automation is necessary to be able to repeat the attack many times, preventing the need to manually select many different urls. We randomize each url by using a random number generator and converting each number to a lowercase, english alphabet, ASCII character. This is then repeated 5-15 times to create a domain name, with ".com" being added to the end of the string to create the full url.

To measure the attack, we have the server run the command "nslookup [random-url]" and redirect the output to a file. Figure 37 shows an example nslookup output. The file's contents are then remotely retrieved to examine the result. If the returned IP address matches that of the attacker's, then the attack succeeded and is given a value of 1. Otherwise, if the IP address doesn't match, it is given a value of 0. Same as with ARP spoofing, the measurements are called "pre move" and "post move" and are repeated 10 time and averaged. This is considered one run of an experiment. The pre and post move averages can then be used to measure the effectiveness of the security measure used to protect against the attack.

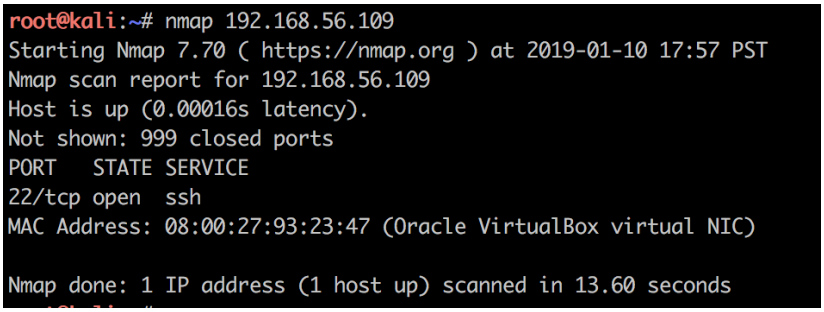

Figure 38: Example NMAP Output from Attacker

4.2.2.3 NMAP Scanning

While NMAP Scanning is not a MITM attack, it is still measured in a similar manner. Section 3.3.6 shows how the attack is run. The goal of the attack is to gain information about a machine(s) and to learn more about a network or subnet. Figure 38 shows example output of running nmap on the attacker. This output is redirected to a file and later retrieved from the attacker to check for success. For simplicity, we decided we would only run the basic NMAP command on one victim server. Scanning is a little more complex to measure, since it is possible to get a range of information. Because of this we couldn't rely on only using values of 0 and 1. We discovered that with some defenses the scan can still detect the server as up, but can't detect the port information. Since the only open port on our server is ssh, we use that to determine whether the scan is fully successful. If the scan is able to determine the state of the victim and its port information, a value of 1 is given. If the scan detects a host is up but doesn't have the ssh port (meaning it had no port information), a value of 0.5 is given. If the scan timed out or said the host is down, a value of 0 is given. Same as with the MITM attacks, the measurements are called "pre move" and "post move" and are repeated 10 time and averaged. This is considered one run of an experiment. They can then be compared to determine the validity of the dynamic security measure used.

Chapter 5

RESULTS

After running the experiments, we compiled the average for each dynamic defense for the pre move, attack (if a DOS attack was used), and post move measurements of success. For each result, we look at how much the success rate shifts based on the dynamic defense mechanism. For the experiments with a DOS attack, the difference between the attack measurement and the post move measurement is the most revealing, though the difference between all 3 measurements reveals something about the success of the defense. The difference between the attack measurement and the post move measurement shows how effective the defense mechanism was in directly stopping the attack. The pre move measurement is used as a baseline to compare against both the attack and the post move measurements. The pre move success rate informs how much the post move rate is above the norm, further identifying the effectiveness of the dynamic defense. Without the pre move measurement, there would be no way of knowing if the defense slowed down the attack only slightly or completely prevented it. For the MITM/Scan experiments, since there are only the pre move and post move measurements, the only comparison that is needed is the pre move success rate vs the post move success rate. The pre move success rate acts as both the baseline and the attack measurement, removing the need for both a pre move and an attack measurement. Comparing the pre move to the post move success rate shows how effective the dynamic defense was in defending the attack. In this chapter, we look at the results of the four defenses, as described in the previous chapter, and analyze them. The results for each defenses are divided up into two categories, DOS experiments and MITM/Scan experiments, which are analyzed separately. We start with the results and analysis of the Server Protection defense, followed by the three Attacker Prevention defenses.

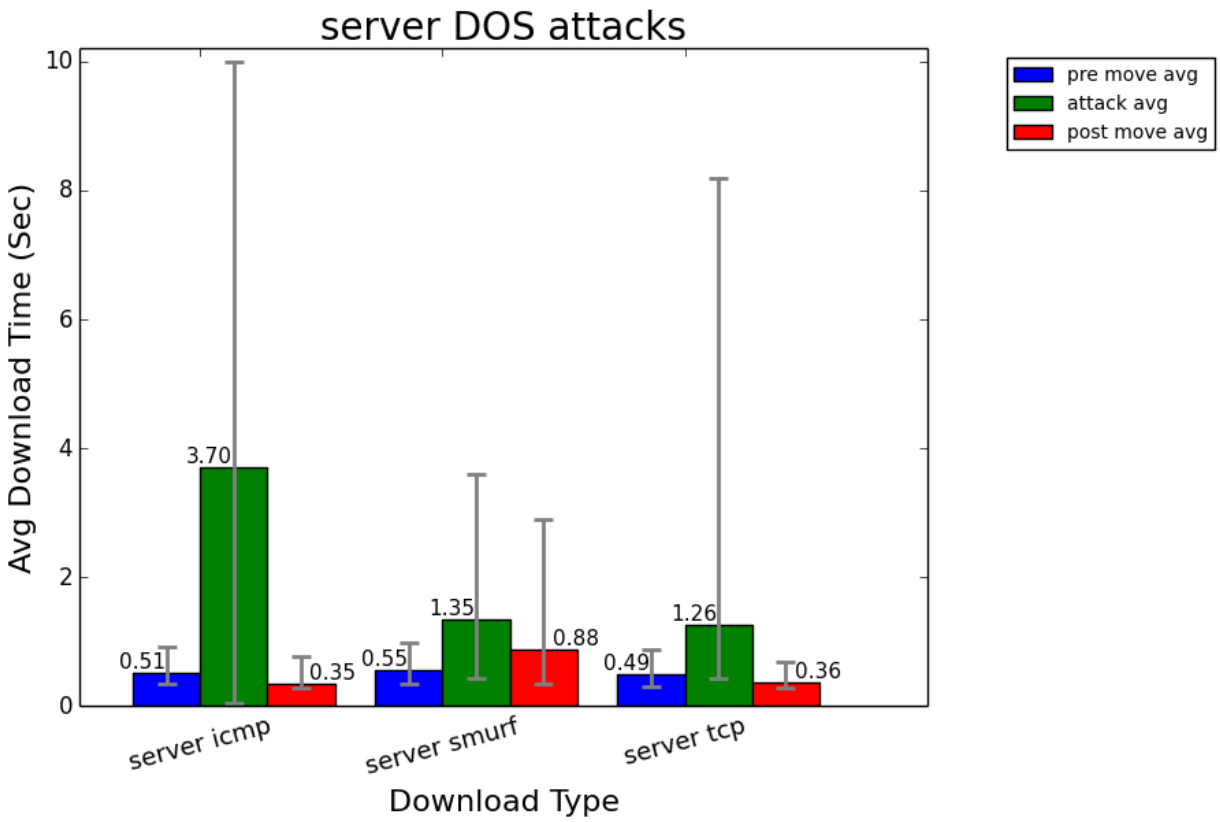

Figure 39: Results for Server Protection DOS Experiments

5.1 Server Protection

The Server Protection defense involves shifting a server to a location behind a firewall, protecting it from either a malicious node that is yet to be detected or an external attacker. In this section, we look at the results of the Server Protection defense and analyze them. We divide the analysis into two categories: DOS and MITM/Scan experiments. We begin with the DOS experiments.

Figure 39 shows the results of the defense for the DOS attacks. The graph shows the average number of seconds it took for a client to download an approximately 32 MB file from a server for the pre move, attack, and post move measurements of each DOS experiment. The three DOS experiments are the ICMP DOS, TCP DOS, and Smurf experiments. For all three experiments, the baseline

(pre move) measurement stays around 0.5 seconds. For the ICMP DOS experiment, the average download time increases staggeringly to 3.7 seconds once the ICMP DOS attack is launched, but before the dynamic defense is initiated. This constitutes a 628% increase in download time. Once the server is moved behind the firewall, the average download time plummets to 0.35 seconds, even faster than baseline time. While we are not sure why the average download time is faster than the baseline, though we think it has something to do with the baseline measurement being slower than expected, this shows that the dynamic shift of the server unequivocally protected it from the ICMP DOS attack. The TCP DOS attack shows a similar result, though the avg download time for the attack measurement is 1.26 seconds instead of 3.70 seconds. The Smurf attack experiment, however, shows slightly different results. While the average download time increases from 0.55 seconds to 1.35 seconds, a 143% increase, the dynamic defense only lowers the average download time to 0.88 seconds. The post move measurement is a 37% increase over the baseline (pre move) measurement. So while the firewall does somewhat protect the server from the Smurf attack, it does not do so fully. This is most likely due to the fact that a Smurf attack utilizes spoofing, something that the distributed firewall can not protect against. Even utilizing the distributed firewall's spoof guard wouldn't be effective, since it isn't aware of the attacker or its addresses. Despite the ineffectiveness of the firewall, it is still able to at least slow down the Smurf attack, lowering the download time by 53%.

Figure 41 shows the results of the defense for the MITM/Scan attacks. The graph shows the average success rate for each attack, as described in section 4.2. The three MITM/Scan experiments are the DNS Spoof, ARP Spoof, and NMAP Scan experiments. For all three experiments, the success rate pre move is almost 1 (100%). For the NMAP Scan attack, the success rate drops all the way to 0 once the dynamic defense of the distributed firewall and spoof guard is initiated. This

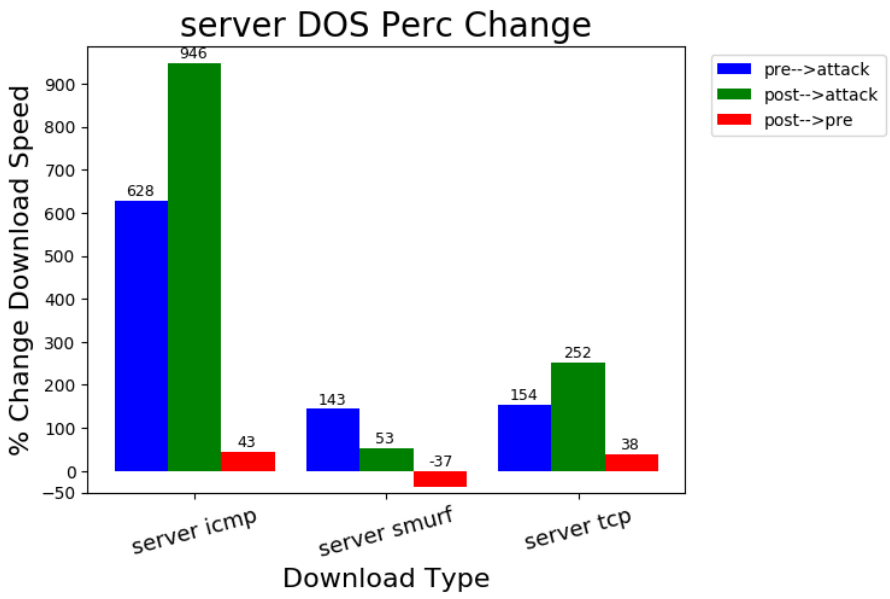

Figure 40: Percent Change for Server Protection DOS Experiments

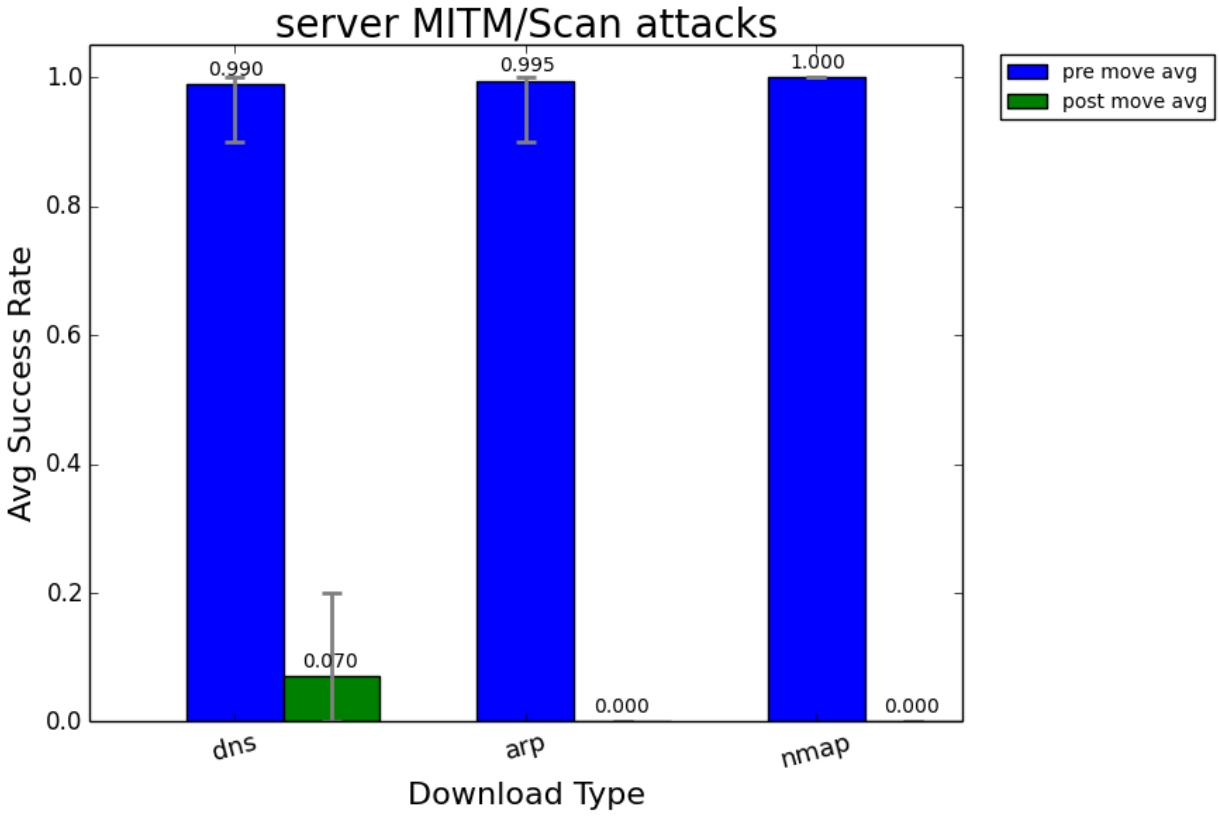

Figure 41: Results for Server Protection MITM/Scan Experiments

shows the effectiveness of the firewall in protecting against scanning. For the ARP and DNS Spoof attacks, the success rate also significantly drops post move. For the ARP attack, the pre move measurement is 0.995, while the DNS attack has a pre move measurement of 0.990. The post move measurements of ARP and DNS Spoof are 0 and 0.07 respectively. The distributed firewall is able to fully prevent the ARP Spoof from occurring, while the DNS Spoof is able to still have some success. Because the DNS Spoof attack is reliant on ARP Spoofing, its chances of success are greatly reduced due to the ARP Spoof attack being neutralized. Without the ARP Spoofing to reroute the DNS query to the attacker, the attacker has to get lucky and beat out the real DNS server to succeed. As seen from the results, this is not likely. So while the dynamic defense doesn't prevent the DNS Spoof completely, it does lower the average success rate from 0.99 all the way to 0.07. This dynamic defense relies on preventing any new ARP packets from being sent to the server from our external cluster, which is potentially unrealistic since no new devices would be able to reach the server. The spoof guard is ineffective in this scenario since it doesn't know any information about the malicious node. Without the dropping of ARP packets, most likely the dynamic defense wouldn't be very effective.

Overall, the Server Protection dynamic defense is very effective against the DOS attacks and NMAP Scanning, but isn't able to effectively prevent spoofing attacks without dropping ARP packets, due to not knowing the identity of the attacker.

5.2 Attacker Prevention

Attacker Prevention is broken up into three defenses: Isolated Subnet, Distributed Port Group, and Standard Port Group. These defenses all use a shifting Virtual Network topology to protect against a malicious node in the internal network. Unlike in the Server Protection DOS defense, two topologies are used for

the Attacker Prevention DOS experiments. In this section, we look at the results of each defense and analyze them, starting with the Isolated Subnet defense.

5.2.1 Isolated Subnet

The Isolated Subnet defense involve detecting a malicious node and dynamically creating an isolated standard port group to move it to. In this section, we analyze the results of this defense and its corresponding experiments. We divide the analysis into two categories: DOS and MITM/Scan experiments. We begin with the DOS experiments.

Figure 42 shows the results of the DOS Isolated Subnet defense for the first topology (using an internal client). The results are similar to the Server Protection DOS experiments, although there are a few differences. The average pre move download time for all three experiments is again almost exactly 0.5 seconds. For the DOS TCP, Smurf attack, and DOS ICMP experiments, the average attack measurement is around the same: 1.19, 1.19, and 1.45 seconds respectively. The increases in download time are 138%, 149%, and 192% for each experiment, constituting a large increase in download time for each. The post move average download times are all slightly below the baseline measurement. Unlike the in the previous experiments, the isolated subnet was able to prevent the Smurf attack completely. This is because in this experiment the malicious node is completely isolated from the rest of the Virtual Network. All three of the experiments show that the dynamic defense is very effective at stopping the DOS TCP, DOS ICMP, and Smurf attacks.

Figure 43 shows the results of the DOS Isolated Subnet defense with the second network topology. For the second topology, where the client is now in the external network, the results are very similar to the DOS experiments with the first topology. The main differences are that the ICMP Flooding attack is more

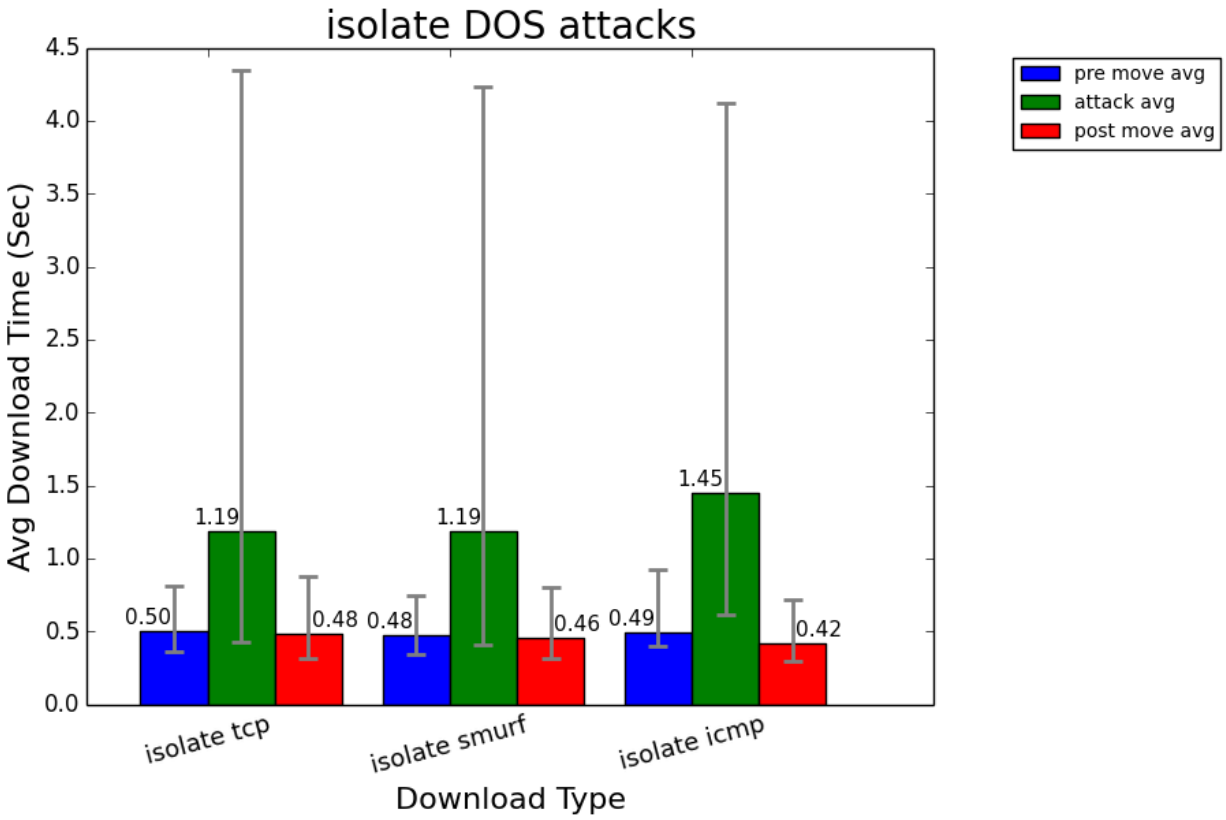

Figure 42: Results for Isolated Subnet DOS Experiments for Topology 1

disruptive and the Smurf attack is less disruptive before the dynamic defense is initiated. The TCP Flood stays at about the same effectiveness. The average download time pre move is again around 0.5 seconds. The attack measurements are 3 seconds (a 512% increase), 0.76 seconds (a 58% increase), and 1.06 seconds (a 116% increase). As with the first topology, the post move download time is lower than the baseline for all three experiments, with the DOS ICMP post move measurement being much lower at an average of 0.38 seconds. Again, any decrease from pre move to post move is most likely due to circumstances unrelated to the dynamic defense. However, the decrease does prove that the dynamic defense is effectively stopping all three DOS attacks. This is expected, as any device that is isolated is expected to be unable to execute any malicious activity. The information shows that by using a dynamic topology shift to isolate the attacker, the DOS attacks are prevented.

Figure 45 shows the results of the MITM/Scan Isolated Subnet defense. The results for the ARP Spoof, DNS Spoof, and NMAP Scan experiments all show a significant drop from almost a 100% success rate to a 0% success rate. The pre move average measurements are 1.0, 0.99, and 0.995 respectively. However, once the dynamic defense is employed, the post move success rate for all three attacks is 0. This is expected as isolating a node from the rest of the network would prevent any external interaction, making any malicious activity impossible. The change in success rate from 1 to 0 shows how effective the dynamic defense is in preventing these attacks. The ability of the Virtual Network environment to dynamically create an isolated subnet and shift the topology to isolate the attacker allows for the prevention of the MITM and Scan attacks in this malicious node scenario.

Overall, the isolated subnet is very effective in preventing all the attacks run in our experiments. For all 9 experiments, the isolated subnet dynamic defense fully neutralizes the malicious node.

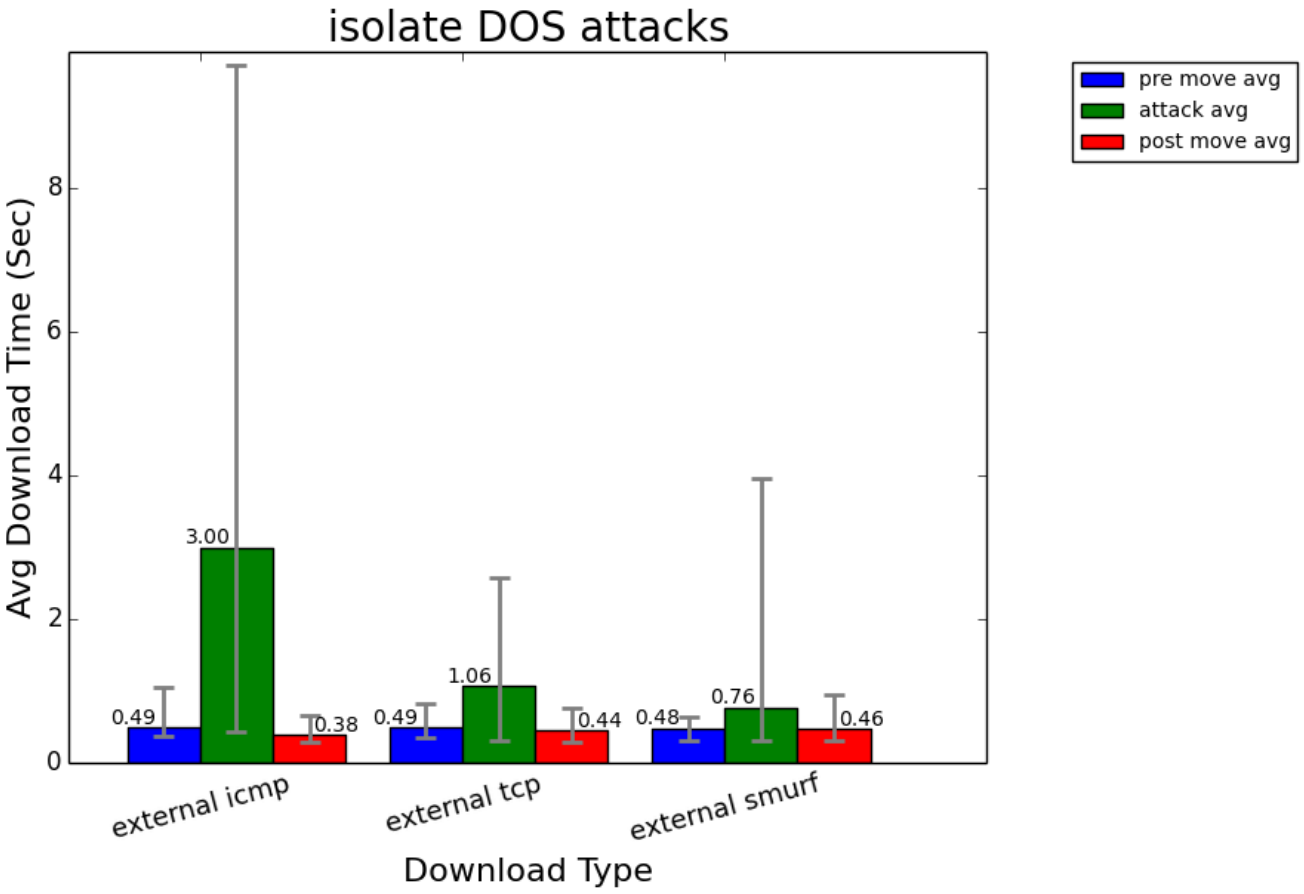

Figure 43: Results for Isolated Subnet DOS Experiments for Topology 2

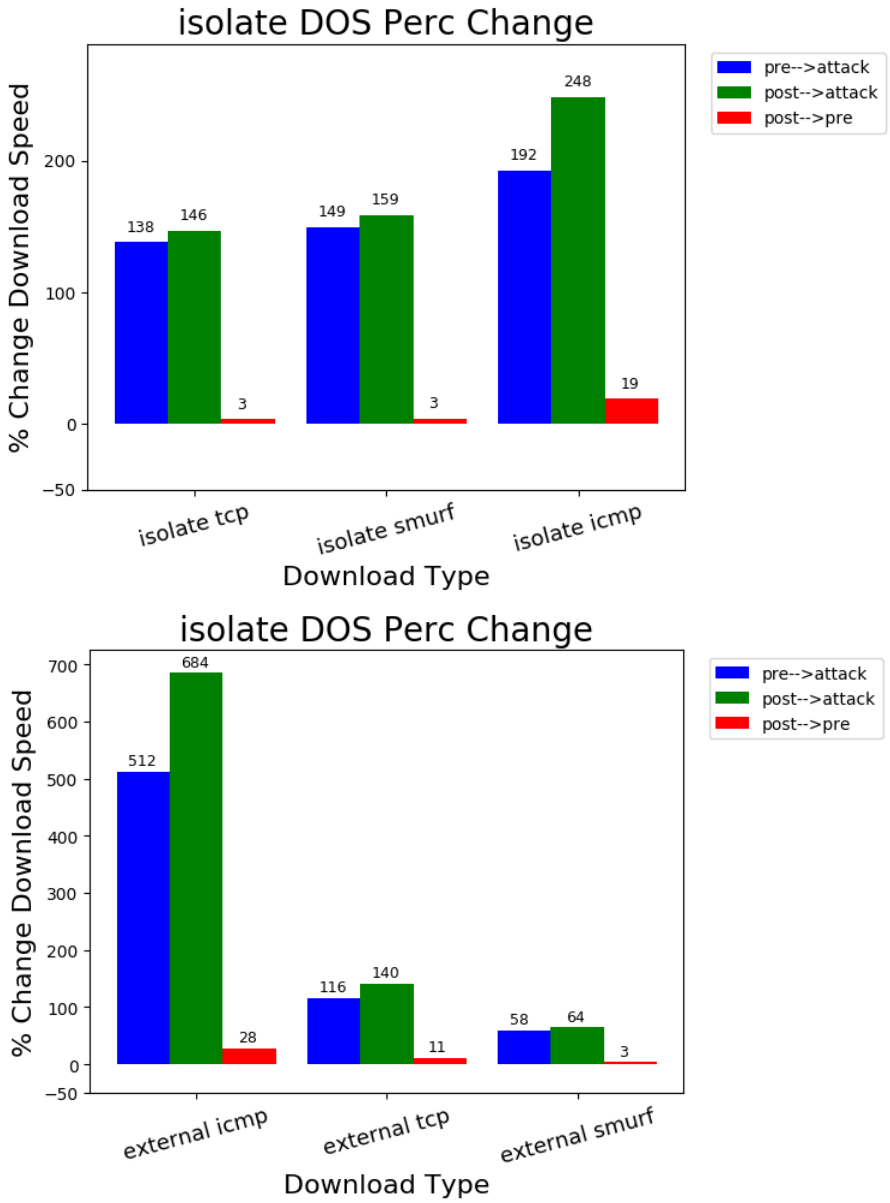

Figure 44: Percent Change for Isolated Subnet DOS Experiments

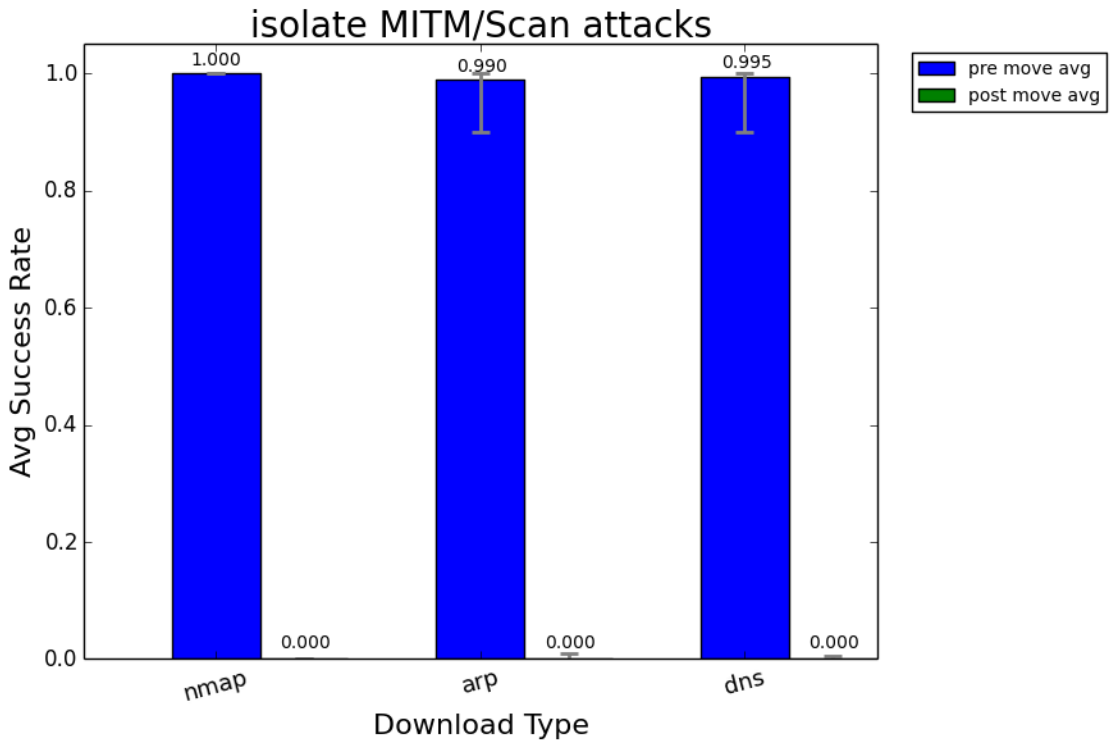

Figure 45: Results for Isolated Subnet MITM/Scan Experiments

5.2.2 Distributed Port Group

The Distributed Port Group defense involve detecting a malicious node and shifting the network topology to put the attacker behind a distributed firewall and spoof guard. In this section, we analyze the results of this defense and its corresponding experiments. We divide the analysis into two categories: DOS and MITM/Scan experiments. We begin with the DOS experiments.

Figure 46 shows the results for the DOS defense with the first topology. For the first topology, where the client is in the internal network, the results show similarities to the previous experiments. The baseline (pre move) average download time is still close to 0.5 for all three experiments. Because the topology of these experiments involves the attacker and the server initially connected to distributed port groups, the attacks aren't as effective as in previous experiments. They still do

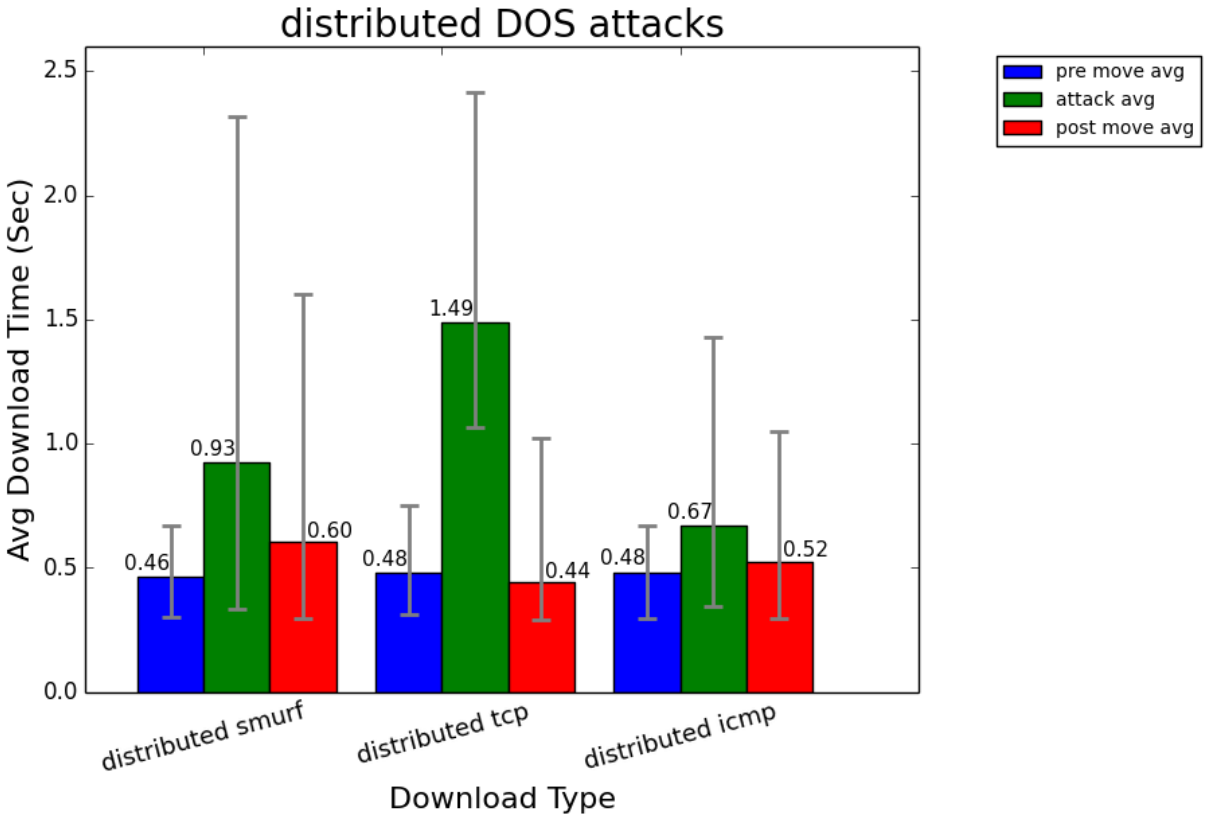

Figure 46: Results for Distributed Port Group DOS Experiments for Topology 1

slow down the client, especially the TCP Syn Flooding attack. On average, the Smurf attack measurement is 0.93 seconds (a 99% increase), the DOS TCP attack measurement is 1.49 seconds (a 211% increase), and the DOS ICMP attack measurement is 0.67 seconds (a 38% increase). Once the attack is detected and the dynamic defense is initiated, the download times of all three experiments drops, although to varying degrees.

Similar to the Server Protection experiments, the post move average download time drops for the Smurf experiment, but not all the way back down to the baseline level. This is confusing, as the spoof guard should be effective in preventing the attacker from spoofing once it is moved to the protected distributed port group. As we see later in the topology 2 DOS experiments, the Smurf attack is fully neutralized, making it likely that something is off with the topology 1 experiment that created extra overhead. For the other two attacks, the distributed firewall is fully effective, dropping the download time back to around the baseline. For the TCP DOS experiment, the average download time plummets from 1.49 seconds to 0.44 seconds, slightly below the baseline time. The ICMP DOS experiment also sees a drop in average download time from 0.67 seconds to 0.52, slightly above the baseline time but still a decline in average time. These two experiments show that the distributed firewall is fully effective in stopping the malicious node from initiating a denial of service on the server. Even for the Smurf attack, which has a slowed post move download time compared to the baseline, the download time still decreases by 55% from the attack to the post move measurement. This is a significant enough decrease to effectively prevent the Smurf attack, despite being slightly slower than before the attack.

Figure 47 shows the results of the second topology for the DOS defense, where the client is moved from the "internal" to the "external" network. In this set of experiments, the effect of the attacks are more muted, although they still hamper

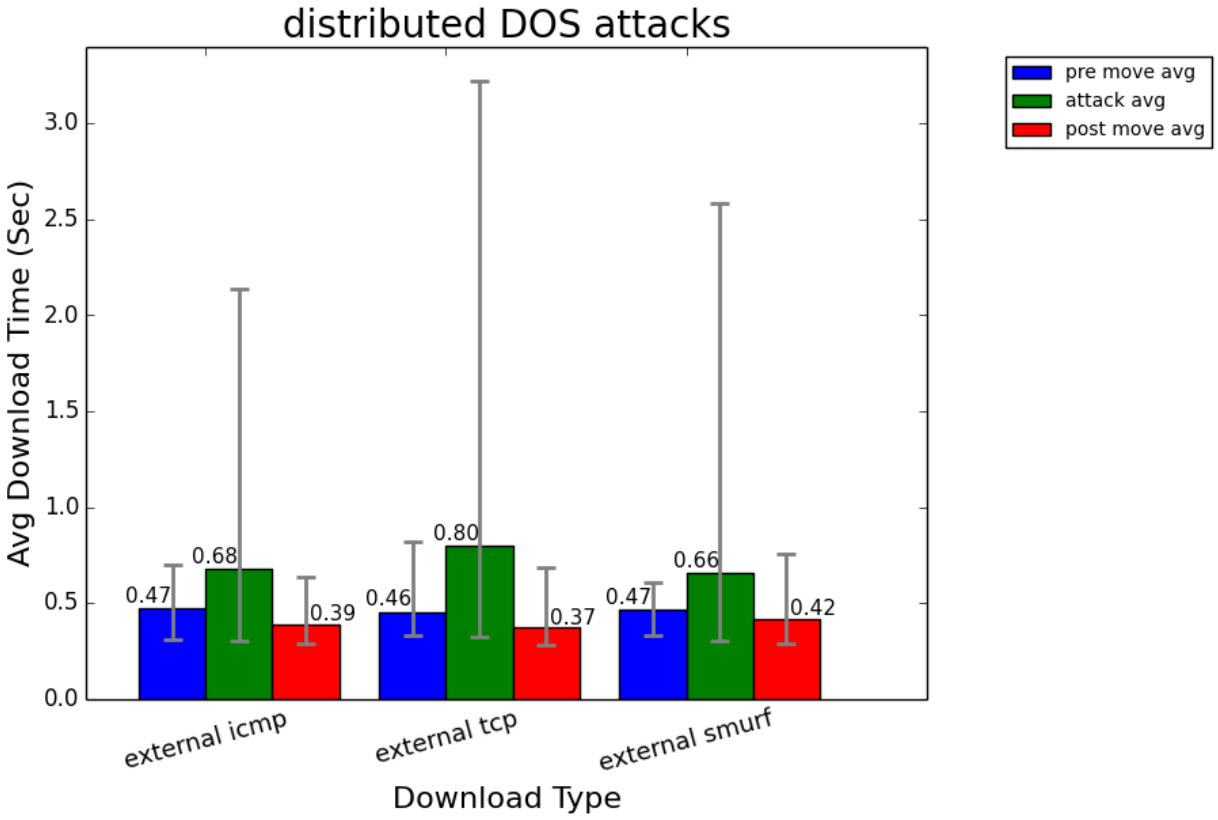

Figure 47: Results for Distributed Port Group DOS Experiments for Topology 2

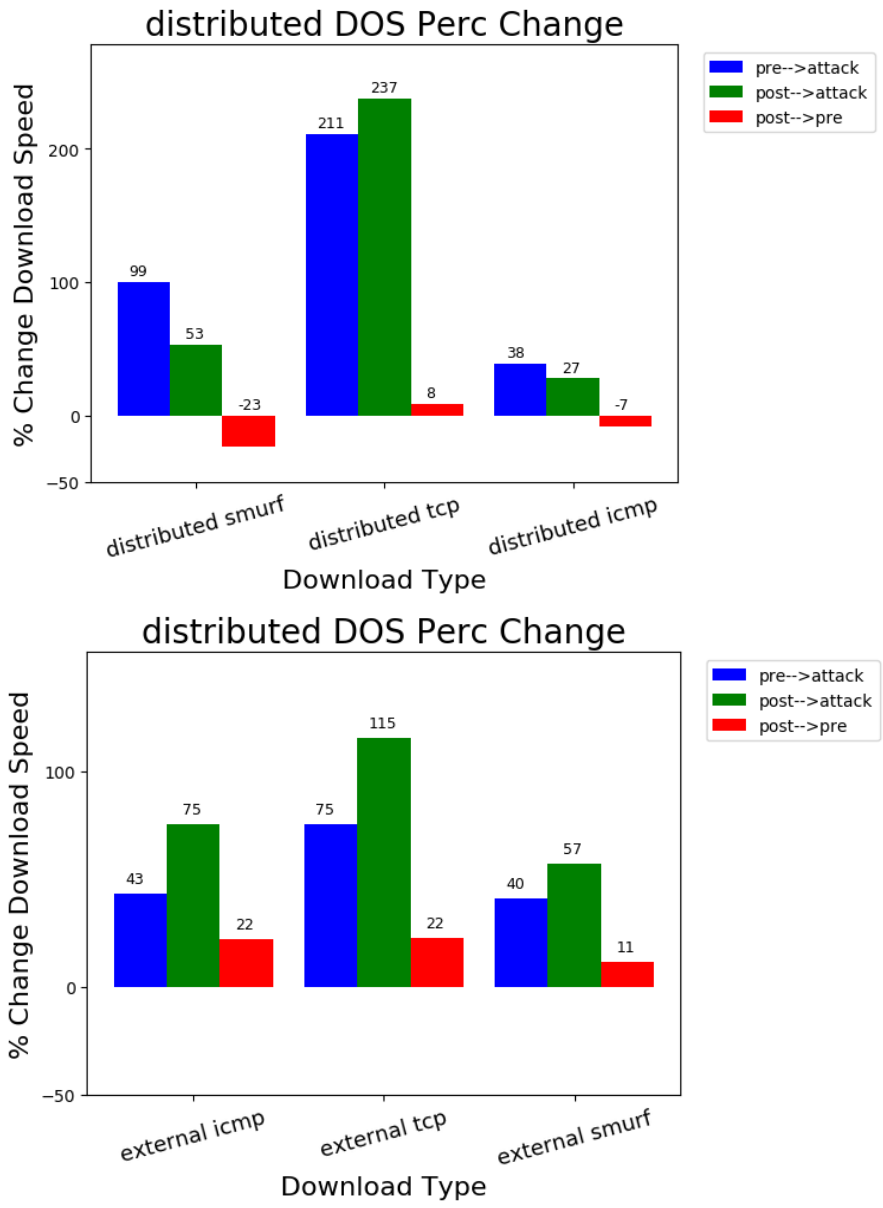

Figure 48: Percent Change for Average Download Time for Distributed Port Group DOS Experiments

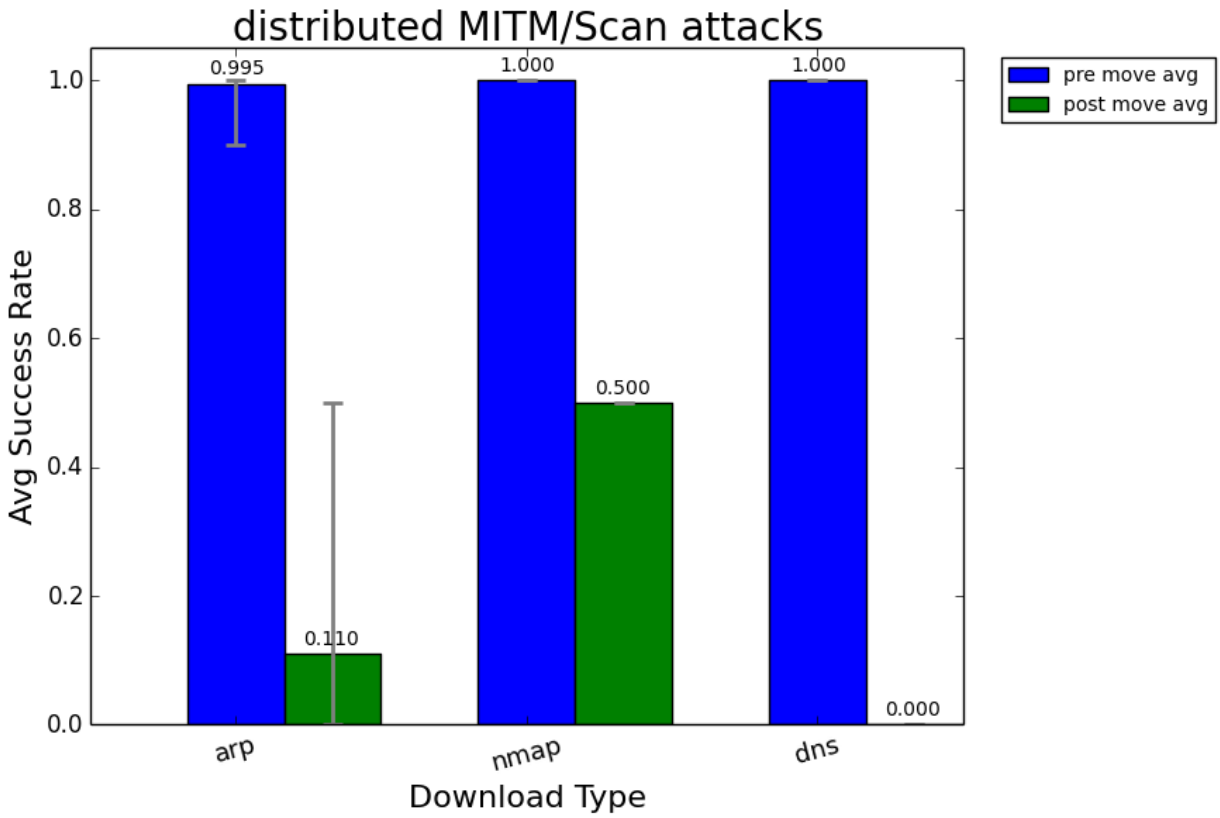

Figure 49: Results for Distributed Port Group MITM/Scan Experiments

the client's ability to communicate with the server somewhat. All three baseline measurements are again around 0.5 seconds. The average attack measurement is 0.66 seconds (a 40% increase) for the Smurf experiment, 0.8 seconds (a 75% increase) for the TCP DOS experiment, and 0.68 seconds (a 43% increase) for the ICMP DOS experiment. Once the dynamic defense is utilized, the average download time for the client for all three experiments drops below the baseline level. Despite the attacks not being as effective as previous experiments, it is clear that the shift of the Virtual Network topology to put the attacker behind the distributed firewall and spoof guard is able to stop the denial of service. In the future, it is necessary to replicate these DOS experiments with more attackers to be able to test the distributed firewall dynamic defense against fully effective attacks.

Figure 49 shows the results of the MITM/Scan defense. The pre move success rates for the ARP Spoof, DNS Spoof, and NMAP Scan experiments are again all near 1 (100%). However, once the malicious node is moved behind the distributed firewall and spoof guard, the success rates of all three drop significantly, albeit to varying degrees. For the NMAP Scan experiment, the post move success rate is 0.5. This means that the NMAP Scan is consistently able to verify that the server is up, but fails to determine the open ports on the server. While the attack is able to derive some information, the most important information of open ports, which is used to determine vulnerability in a machine, is blocked by the distributed firewall. So while the malicious node's scan doesn't fail completely, the dynamic defense hinders its ability to find vulnerabilities. For ARP Spoof, the success rate plummets after the attacker is shifted behind the spoof guard. The average success rate goes from 0.995 to 0.11. Most likely this should have dropped down to 0, but for some reason the first ARP attack out of ten for each run of this experiment was successful post move. This is possibly due to the spoof guard needing more time than expected to start detecting spoofing attacks, or perhaps something with the way the spoof guard works prevents it from detecting the first time a node tries to spoof a packet. Either way, the attack success rate still drops substantially, showing the dynamic defense's effectiveness. For the DNS Spoof, the spoof guard is able to completely stop the attack. The average success rate from pre move to post move goes from 1 all the way down to 0. Even if the spoof guard isn't able to stop the ARP Spoof part of the DNS Spoof attack, the spoof guard detects the DNS Spoofing and immediately drops the spoofed packets. The MITM/Scan experiments show that the dynamic defense can still prevent a large number of attacks, even if it is not 100% effective.

Overall, this dynamic defense is effective in stopping every attack almost completely. It is not able to fully prevent the NMAP Scan and Smurf attacks, though it substantially lowers the success rate of each.

5.2.3 Standard Port Group

For the Standard Port Group defense, a malicious node is detected and moved to a dynamically created standard port group with spoofing protections. In this section, we analyze the results of this defense and its corresponding experiments. Since the protections involve spoofing, there is only a MITM section of these experiments. We start by analyzing the ARP Spoof and DNS Spoof experiment results.

Figure 50 shows the results of the defense and the two experiments run. As can be seen from the results, the dynamic defense had no effect on the attacks. Both the ARP Spoof and DNS Spoof had an average success rate of 1 for the pre move and post move measurements. While not likely, it is possible that the standard port group was ineffective due to the use of nested virtualization. If these experiments are run on a full Virtual Network environment, it would show whether the standard port group dynamic defense is truly unsuccessful or whether the failure is the result of the use of nested virtualization.

Overall, this dynamic defense is completely powerless in stopping the ARP Spoof and DNS Spoof attacks.

5.3 Analysis

In this section, we summarize the analyses of all the different defenses and their experiments, and describe the trends of the dynamic defenses. Again, the analysis is divided between the DOS and the MITM/Scan experiments. We begin

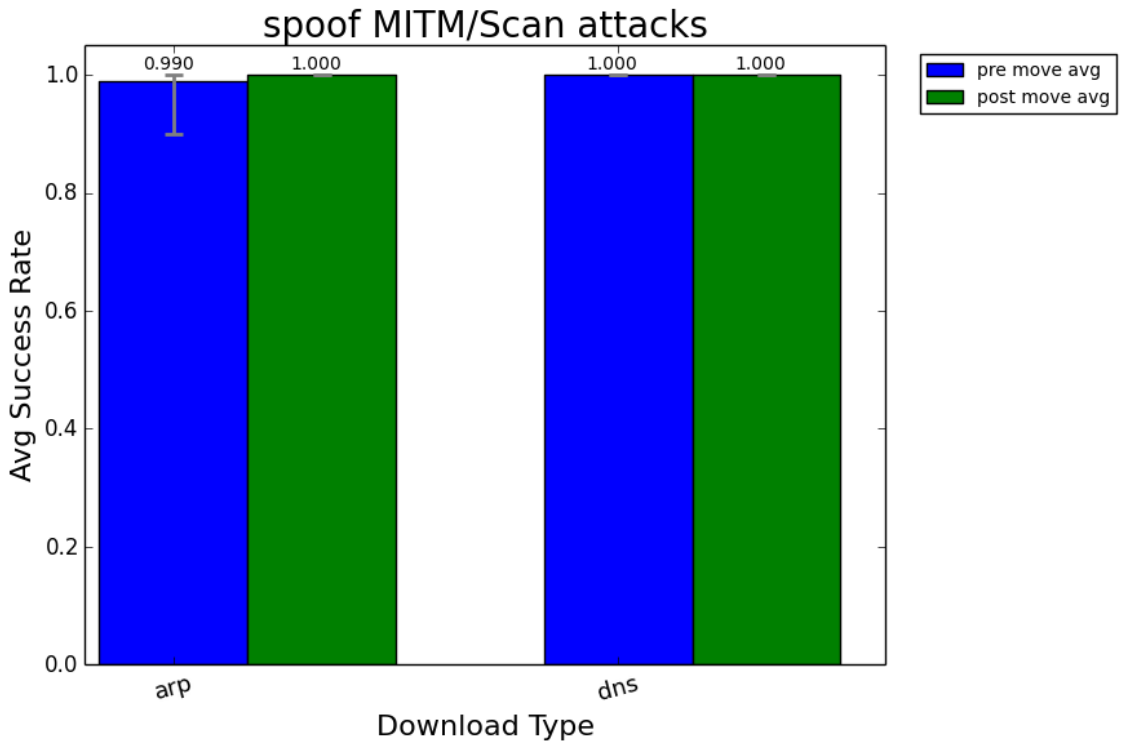

Figure 50: Results for the Standard Port Group Experiments

with a summary of the DOS experiments and the effects of the dynamic defenses on the DOS attacks, followed by the MITM/Scan experiments.

Overall, the dynamic defenses, outside of the standard port group, are able to effectively protect the server and neutralize the attacker. For the DOS attack defenses, the post move download times are almost all below the baseline (pre move) level. The exception to that is the Server Protection defense and one of the Smurf attack experiments for the Distributed Port Group defense. In those cases, the distributed firewall is not able to fully negate the effects of the denial of service, most likely due to the spoofing used in the attack. However, the dynamic defense is still able to slow down the attack and cut the average download time by around 50% in both cases. Outside of those two DOS experiments, the rest of the experiments show that the use of a shifting topology is able to successfully protect the server and completely stop the attacker. For some of the experiments, the DOS

attacks aren't as effective as they are in other experiments, but the results still show an obvious positive effect by the dynamic defenses. Those experiments could be re-run with more malicious nodes to fully prove the effectiveness of the dynamic defenses, but there is no reason to believe the dynamic defenses wouldn't still be able to fully prevent the denial of service.

For the MITM/Scan attacks, the dynamic defenses are fully effective too, outside of the standard port group. All the pre move average success rates for every experiment are around 1, meaning a 100% success rate. Once the dynamic defenses are initiated, most of the post move average success rates become 0. One of the ARP Spoof experiments shows a potential flaw with one dynamic defense, with the attack succeeding on the first attempt after the move, but failing all attempts after that. Despite this potential flaw, the dynamic defense still drops the success rate of the attack from 1.0 down to 0.11, a very effective defense. One of of the DNS Spoof experiments falls to around 0.07, because the spoofing of the DNS Packet is able to get around the distributed firewall. However, since the DNS Spoof attack relies on ARP Spoofing, which is prevented by the distributed firewall, the DNS spoof attack has to beat out the real DNS server in replying to the DNS request packet. This is not highly successful, which is why we see the DNS Spoof experiment's post move success rate fall all the way to 0.07. Besides the failure of the standard port group, the dynamic defenses lower the success rate of each attack significantly. Only one experiment, the distributed port group NMAP Scan experiment, has a post move success rate much higher than 0.1, and even that success rate is a 0.5 only because it determines that the host is up. However, the scan is not able to discover the port information of the server, the most important part of the scan. For all three MITM and Scan attacks, the dynamic defenses are fully effective in either completely neutralizing the attack or in causing the malicious node to have a very low rate of

success. In each case, outside the standard port group, the success rate is always significantly lowered once the dynamic defense is initiated.

For all experiments, the dynamic defenses, other than the standard port group, are very effective in stopping all six attacks. This shows that by dynamically shifting the topology, a Virtual Network can be protected from network attacks. This could be effectively utilized in the future to help secure Virtual Networks, if the use cases of Virtual Networks expands outside of data centers. There are some weaknesses to this system, but overall, Virtual Networks would benefit from fully using their flexibility to protect resources within the network.

Chapter 6

RELATED WORKS

Utilizing the flexibility of a network to shift topology for security purposes is not a unique idea. Dynamic topology shifting and testing through attack simulations has been done with different SDN and virtual environments. There are a few papers that look at utilizing SDN resources or creating SDN add ons to increase security dynamically. Gillani et al. [15] look at securing the weakness of SDNs where control and data packets share critical links, opening up an attack vector for DDOS attacks. They create a system called ReCON that uses existing SDN resources to dynamically defend against DDOS attacks of the control plane. It does this by lowering the amount of sharing of links for data and control packets, and by dynamically using underutilized resources to increase the capacity of control agents. Both these serve to make DDOS attacks on the control messages more difficult. ReCON quantifies the resilience of the control network by measuring link isolation and criticality. The system then uses the metric to decide where to place network links for the control and data planes. To test their system, the authors run two types of DDOS attacks: Link Sharing-based DDOS and Control Capacity-based DDOS. Their results show a decrease in DDOS effectiveness by about 40% and a doubled OpenFlow agent capacity. While this system does dynamically defend against network attacks, they only protect against DDOS attacks and not any other type of network attack. Their system is also built for SDNs rather than Virtual Networks, and they simulate their experiments rather than running them in a true network environment.

Yoon et al. [63] investigate the ability of Software-Defined Networks to replace hardware security functions. To do this they implement four types of network security functions in a SDN environment: in-line mode security, passive mode security, network anomaly detection, and advanced security. These functions

are implemented as applications in a Floodlight SDN controller and tested in a real SDN environment to discover the feasibility of SDN-based security. For in-line mode security, firewall and IPS applications are implemented. Passive mode security is implemented as an IDS application. For network anomaly detection, network scanning and DDOS detector applications are created. The advanced security functions implement stateful firewall and network reflector applications. Each implementation has different positives and negatives in regard to the fit in a SDN environment. Through experimentation the authors concluded that the passive mode and network anomaly detection functions were the most feasible in SDN environments, while the in-line mode functions could be implemented for networks that require security on only a small number of flows. For the advanced security functions, they were only effective when the modifications made to packets were processed in hardware rather than software. While the authors use dynamic security components, they use Software-Defined networks to achieve security, but not Virtual Networks. Their system also doesn't test the security of the functions, but rather the overhead they incur and the feasibility of their implementation.

Achleitner et al. [1] look at defending against advanced network scanning by developing the Reconnaissance Deception System (RDS). RDS is based on Software-Defined Networks and deceives scanners by simulating incorrect network topologies. The attacker is assumed to be a malicious node operating from within the network, allowing for the RDS to manipulate its view of the network. RDS works by first detecting the threat and discovering the malicious node. RDS then uses fake network features to simulate a network view which can only be viewed by the attacker. The RDS system adds the capability to forge packets and generate these fake networks into the SDN controllers. For their experiments, their results show much longer times for scanning a network while creating very little overhead. RDS adds time to malicious scanning at up to a factor of 115, while only creating a

network overhead of 0.2 milliseconds. This system also utilizes dynamic shifting of network topologies, but does this with a Software-Defined Network rather than a real Virtual Network. It also doesn't explore other defenses and attack types other than scanning. It also requires new SDN controllers and system setup rather than using portable software or existing technology.

Shin et al. [47] look at two potential security weaknesses in SDN environments and creates solutions for each. One of the weaknesses is that changes in flow and network statistics can only be detected by controller applications through polling, which could prevent applications looking for malicious behavior to respond properly. To solve this they create a technique called "actuating trigger" that allows the data plane to report network and payload information to the control plane. It could then trigger a predefined set of flow rules to cause an immediate reaction to any malicious behavior that is detected. Their experimentation shows that the actuating trigger adds very little overhead but allows for quicker threat detection and reaction. While they do look at dynamic reaction to malicious behavior, it is in an SDN and not a Virtual Network environment. It also involves adding to the SDN architecture rather than using existing features for attack mitigation.

Cox et al. [13] create Net Flow Guard (NFG), an SDN app which can detect rogue DHCP servers and disable them. It does so automatically and is modular. Their implementation includes an OpenFlow switch connected to a controller with NFG on it. The switch then forwards DHCP packets to the controller for analysis to determine the validity of the server. They utilize Mininet to evaluate their application and found NFG to be effective in discovering malicious DHCP activity. While they do protect against DHCP attacks, their system doesn't protects against other network attacks and doesn't make any changes to the network topology in

response. Their experiments are simulated rather than run in a real network and their system is for SDNs, not Virtual Networks.

Kreutx et al. [26] look at attack vectors that exploit certain weaknesses in SDNs. For each threat they describe how the attacks can be prevented by making changes to SDN's architecture. They then take all the solutions presented and use them to create a general outline of a more secure SDN architecture. The idea behind this is that SDNs should be inherently secure and not have to depend on external software to secure the environment. This system looks at Software-Defined Network security and recreating the architecture to have built-in security. However, they don't look at Virtual Networks and they also change the existing system rather than use any dynamic shifting in the network. They also doesn't test the suggested architecture with any experimentation.

Each of these works utilizes dynamic SDN shifts, either through flows or other means. However, none of them look at Virtual Network environments and most of them don't protect against a varied set of network attacks, usually only protecting against one attack type.

There are also works that involve the use of dynamic virtual security components in SDNs or virtual environments. These works look at ways to use those virtual components to dynamically secure the network. Park et al. [41] use Network Function Virtualization to create a new SDN framework that allows for dynamic provisioning of virtualized, lightweight IDS network functions. SDN controllers are used to deploy and control the IDSs throughout the network while looking at the high-level network view. Rather than create big IDSs, the system creates small intrusion detection network functions and chains them together, deciding on the fly which ones to create and where to put them. The system architecture relies on four different components: a traffic classifier, a network functions pool, a service chaining module, and a virtual machine manager. The

SDN controllers monitor the path of flows, and based on that determine which switches to put certain IDS network functions on. The service pool has two different types of IDS network functions: packet header inspection and deep packet inspection. The packet header and deep packet inspection are selected based on protocols and ports, allowing for only the necessary IDS functions to be used in each location. The virtual machine manager then creates the service chain using the network functions selected. This service chain is dynamically created and allocated based on situation. Their testing shows that their system uses less than 7 MB of RAM, much smaller than general IDSs. Their system also takes less than a millesecond on average to spawn a single network function. While this system also uses dynamic virtual network security, it is used in an SDN environment with physical networking components rather than in a Virtual Network environment.

Hongda et al. [28] create a new Network Intrusion Detection System (NIDS) called vNIDS, which is used to efficiently create and provision virtual NIDSs. The goal is to make NIDSs more flexible than how they are currently used as middle boxes, where they can't be moved or created easily. They also must make sure the virtualized NIDSs don't miss attacks and that they are provisioned properly and don't overuse resources. The two main goals are to create effective intrusion detection and non-monolithic NIDS provisioning. The first goal is accomplished by dividing detection states into local and global, lowering the amount of data that must be shared between virtualized NIDSs. The second goal is accomplished by dividing IDSs into "microservices": header-based detection, protocol parse, and payload-based detection. Each microservice can be created separately and placed anywhere in the network in any numbers, as well as chained together. Their evaluation compares the vNIDS architecture to the Bro NIDS. Their system shows a similar ability to detect attacks with increased flexibility. The vNIDS architecture does, however, have increased overhead compared the Bro NIDS, which the authors

attribute to the inefficiency of their implementation for the experiments and say can be mitigated. This system utilizes virtualized security, but it is more about provisioning pre attack rather than using flexibility to react to an attack that is occurring.

Shin et al. [46] look at the use of virtualized network security functions in securing a network. There is a problem of having to deploy countless security middle-boxes in different locations throughout physical networks and the difficulty of moving them if the security function is needed elsewhere. They create a system called NetSecVisor, which can be used by admins to determine which security functions to put where. The system utilizes pre-set security middle-boxes and SDN technology to accomplish this. NetSecVisor provides security by rerouting flows based on events, such as a VM being migrated from one host to another. If that VM is being protected by a security middle-box on the first host, the system dynamically reroutes the flows to still pass through that middle-box. Their evaluations show that their system adds very little overhead, while being able to reroute traffic to the IDSs and firewall. While this system provides dynamic isolation and protection of VMs, it still relies on physical middle-boxes rather than virtual network security components. It also isn't used in a true Virtual Network environment, instead using an OpenFlow SDN network that doesn't utilize any Virtual Network technology.

Beham et al. [6] look at the use of honeypots and IDSs in nested virtualization as compared to their use in regular virtualization. They look at the nested hypervisors of KVM on top of KVM for an IDS and a honeypot and compared the time for analysis versus those security components on top of just a single KVM hypervisor. Their results show a large increase in execution time for nested virtualization to produce similar results in detecting and analyzing malicious network behavior. These experiments look at comparing efficiency of nested

virtualization to regular virtualization, rather than looking at securing Virtual Networks.

All these papers either utilize virtualization in a physical SDN network or don't use dynamic topology shifts in reaction to attacks, but rather use pre-planned security. While there are a lot of works looking at SDN and Virtual Network security, none really looks at using a Virtual Network's flexibility to shift the network topology in reaction to a network attack.

Chapter 7

CONCLUSION

With the idea of Virtual Networks and the potential for their expanded use in the future, we explore the use of dynamic changes to a Virtual Network's topology to protect resources and defend against malicious activities. We utilize VMware vSphere and vCenter to set up our environment and utilize nested virtualization to emulate the use of multiple, underlying, physical network components. We theorize that the flexibility of a Virtual Network to dynamically change its topology can be used for network security purposes. To test our theory, we create a set of dynamic defenses and experiments to determine the validity of these defenses. There are 4 different groups of defenses: Server Protection, Isolated Subnet, Distributed Port Group, and Standard Port Group. The Server Protection defenses involves detecting an attack against a server and shifting the server behind a protected subnet. The other three defenses involve detecting a malicious node and shifting the Virtual Network topology so that the attacker is behind some protection. The Isolated Subnet defense dynamically creates an isolated subnet and move the attacker to it. The Distributed Port Group defense puts the attacker behind a distributed firewall and spoof guard. The Standard Port Group defense dynamically creates a standard port group with spoofing protections and moves the attacker to it.

For each defense we run 6 to 9 different experiments based on attack type, with the attacks being used to test the validity of each dynamic defense. The network attacks used are ICMP Flooding, TCP Syn Flooding, Smurf attack, ARP Spoofing, DNS Spoofing, and NMAP Scanning. The DOS defense experiments are run by having a client download a file from a server, with the attacker trying to slow down or prevent the download. The MITM/Scan defense experiments are each run with the attacker trying to compromise the client based on the attack. Our validation shows that most of the dynamic defenses are very effective in stopping

each attack. The Standard Port Group defense is the one dynamic defense that is completely ineffective. The ability to stop spoofing attacks from an unknown location is also a problem, as the Smurf attack is not able to be completely stopped in some of the experiments due to its spoofing. It is, however, still slowed to a point that the client-server connection is not much slower than before the attack is run.

The Server Protection defense shows positive results for both the MITM and DOS attacks. The download speed of the client falls from above 1 second when being attacked, to below the baseline level of 0.5 seconds for two of the DOS attacks. The third DOS attack falls to a 0.9 second average, which is still a 53% drop. The MITM/Scan attacks all fall from a success rate of around 1 to 0, or 0.07 for the DNS Spoof attack. The Isolated Subnet defense is very effective against all attacks, dropping the download times below the 0.5 second baseline level for all the DOS attacks and dropping the MITM/Scan attack success rates from 1 to 0. The Distributed Port Group defense shows similar results to the Server Protection defense. For the DOS attacks, 5 out of 6 download times fall below the baseline average, while the one outlier still falls by 55% from the attack measurement to the post dynamic defense measurement. The MITM/Scan attacks all have lowered success rates, falling from 1 to 0, 0.11, and 0.5. Despite NMAP Scan having a success rate of 0.5, it isn't able to collect information to determine vulnerability due to the dynamic defense. Finally, the Standard Port Group defense shows it is unable to protect against spoofing attacks, with the success rates of the attacks staying at 1.

Overall, our experiments show that the ability to dynamically shift topologies and quickly spin up new security components in a Virtual Network is very effective at protecting resources and preventing attacks. Currently, the flexibility of Virtual Networks is used more as a failsafe rather than for security purposes. We would recommend that in the future, as the technology of Virtual Networks is more widely adapted and potentially used more in spaces outside of

data centers, that the ability to switch the topology at a moments notice would be used for security purposes. In the future, we plan to expand on the experiments and the system built. We will create larger, more complex and more realistic topologies for experimentation, rather than the one client, one server, one attacker topologies that we used. Using more clients, more attackers, and more attack power would help the testing of the security responses. These expanded experiments would be a step in the right direction for better Virtual Network security.

BIBLIOGRAPHY

[1] S. Achleitner, T. L. Porta, P. Mcdaniel, S. Sugrim, S. V. Krishnamurthy, and R. Chadha. Cyber deception. *Proceedings of the 2016 International Workshop on Managing Insider Security Threats - MIST 16*, page 57?68, Oct 2016.

[2] A. Aljuhani and T. Alharbi. Virtualized network functions security attacks and vulnerabilities. *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, Jan 2017.

[3] Apache. Choosing a deployment architecture.

[4] J. Babbin, S. Biles, and A. Orebaugh. Snort cookbook.

[5] L. R. Bays, R. R. Oliveira, M. P. Barcellos, L. P. Gaspary, and E. R. Mauro Madeira. Virtual network security: threats, countermeasures, and challenges. *Journal of Internet Services and Applications*, 6(1):1, Jan 2015.

[6] M. Beham, M. Vlad, and H. P. Reiser. Intrusion detection and honeypots in nested virtualization environments. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–6, June 2013.

[7] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour. The turtles project: Design and implementation of nested virtualization. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 423–436, Berkeley, CA, USA, 2010. USENIX Association.

[8] S. Cabuk, C. I. Dalton, H. Ramasamy, and M. Schunter. Towards automated provisioning of secure virtualized networks. *Proceedings of the 14th ACM*

conference on Computer and communications security - CCS 07, page 235?245, 2007.

[9] E. Cavalcanti, L. Assis, M. Gaudencio, W. Cirne, and F. Brasileiro. Sandboxing for a free-to-join grid with support for secure site-wide storage area. *First International Workshop on Virtualization Technology in Distributed Computing (VTDC 2006)*, 2006.

[10] D. P. Center. Ddos attacks.

[11] M. Conti, N. Dragoni, and V. Lesyk. A survey of man in the middle attacks. *IEEE Communications Surveys Tutorials*, 18(3):2027–2051, 2016.

[12] E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets.

[13] J. H. Cox, Jr., R. J. Clark, and H. L. Owen, III. Leveraging sdn to improve the security of dhcp. In *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, SDN-NFV Security '16, pages 35–38, New York, NY, USA, 2016. ACM.

[14] Q. Cui, W. Shi, and Y. Wang. Design and implementation of a network supporting environment for virtual experimental platforms. *2009 WRI International Conference on Communications and Mobile Computing*, 2009.

[15] F. Gillani, E. Al-Shaer, and Q. Duan. In-design resilient sdn control plane and elastic forwarding against aggressive ddos attacks. In *Proceedings of the 5th ACM Workshop on Moving Target Defense*, MTD '18, pages 80–89, New York, NY, USA, 2018. ACM.

[16] A. Gueye, V. Marbukh, and J. C. Walrand. Towards a metric for communication network vulnerability to attacks: A game theoretic approach. In V. Krishnamurthy, Q. Zhao, M. Huang, and Y. Wen, editors, *Game Theory for Networks*, pages 259–274, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

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

[18] H. Hu, H. Zhang, Y. Liu, and Y. Wang. Quantitative method for network security situation based on attack prediction. *Security and Communication Networks*, page 19, 2017.

[19] M. Jonker. Millions of Targets Under Attack a Macroscopic Characterization of the DoS Ecosystem, Jan. 2017.

[20] K. Joshi and T. Benson. Network function virtualization. *IEEE Internet Computing*, 20(6):7–9, Nov.-Dec. 2016.

[21] Juniper. Understanding network virtualization with vmware nsx, Nov 2014.

[22] Juniper. Contrail architecture, Sep 2015.

[23] A. Keshri, S. Singh, M. Agarwal, and S. K. Nandiy. DoS attacks prevention using IDS and data mining. In *2016 International Conference on Accessibility to Digital World (ICADW)*, pages 87–92, Dec. 2016.

[24] I. Kotenko and M. Stepashkin. Analyzing vulnerabilities and measuring security level at design and exploitation stages of computer network life cycle. In V. Gorodetsky, I. Kotenko, and V. Skormin, editors, *Computer*

Network Security, pages 311–324, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[25] D. Kreutz, F. M. Ramos, and P. Verissimo. Towards secure and dependable software-defined networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 55–60, New York, NY, USA, 2013. ACM.

[26] D. Kreutz, F. M. Ramos, and P. Verissimo. Towards secure and dependable software-defined networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 55–60, New York, NY, USA, 2013. ACM.

[27] R. Kumar, K. Jain, H. Maharwal, N. Jain, and A. Dadhich. Apache cloudstack: Open source infrastructure as a service cloud computing platform. *International Journal of Advancement in Engineering Technology, Management and Applied Science*, 1(2):111?116, Jul 2014.

[28] H. Li, H. Hu, G. Gu, G.-J. Ahn, and F. Zhang. vnids: Towards elastic security with safe and efficient virtualization of network intrusion detection systems. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 17–34, New York, NY, USA, 2018. ACM.

[29] Y. Li and M. Chen. Software-defined network function virtualization: A survey. *IEEE Access*, 3:2542–2553, 2015.

[30] B. Linkletter. How to emulate a network using virtualbox, Oct 2017.

[31] K. N. Mallikarjunan, K. Muthupriya, and S. M. Shalinie. A survey of distributed denial of service attack. In *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, pages 1–6, Jan. 2016.

[32] J. Mattson. Running nested vms, Mar 2016.

[33] NMAP. Nmap.

[34] OpenStack. Scenario: Classic with open vswitch, Nov 2016.

[35] OpenStack. Scenario: High availability using distributed virtual routing (dvr), Nov 2016.

[36] OpenStack. Scenario: Provider networks with open vswitch, Nov 2016.

[37] OpenStack. Neutron ovs dvr - distributed virtual router, Oct 2018.

[38] OpenStack. Openstack docs: Rocky, Aug 2018.

[39] A. Ornaghi and M. Valleri. Man in the middle attacks, May 2003.

[40] Z. Pan, Q. He, W. Jiang, Y. Chen, and Y. Dong. Nestcloud: Towards practical nested virtualization. In *2011 International Conference on Cloud and Service Computing*, pages 321–329, Dec 2011.

[41] Y. Park, P. Chandaliya, A. Muralidharan, N. Kumar, and H. Hu. Dynamic defense provision via network functions virtualization. *Proceedings of the ACM International Workshop on Security in Software Defined Networks and Network Function Virtualization - SDN-NFVSec 17*, page 43?46, Mar 2017.

[42] S. M. Poremba. Types of ddos attacks, May 2017.

[43] S. S. Rao. Denial of Service attacks and mitigation techniques: Real time implementation with detailed analysis, 2011.

[44] C. Sarraute, F. Miranda, and J. I. Orlicki. Simulation of Computer Network Attacks. *ArXiv e-prints*, June 2010.

[45] SDNCentral. What is a virtual network?

[46] S. Shin, H. Wang, and G. Gu. A first step toward network security virtualization: From concept to prototype. *IEEE Transactions on Information Forensics and Security*, 10(10):2236–2249, Oct 2015.

[47] S. Shin, V. Yegneswaran, P. Porras, and G. Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 413–424, New York, NY, USA, 2013. ACM.

[48] A. Singhal and X. Ou. *Security Risk Analysis of Enterprise Networks Using Probabilistic Attack Graphs*, pages 53–73. Springer International Publishing, Cham, 2017.

[49] Snort. Snort - network intrusion detection and prevention system.

[50] W. Stallings. *Cryptography and network security: principles and practice*. Pearson Education, 2017.

[51] U. S. C. E. R. Team. Dns amplification attacks, Oct 2016.

[52] Y. Vardi and C. H. Zhang. Measures of network vulnerability. *IEEE Signal Processing Letters*, 14(5):313–316, May 2007.

[53] VMware. Thin provisioning.

[54] VMware. Virtualization overview white paper.

[55] VMware. Vmware infrastructure architecture overview.

[56] VMware. Vmware vsphere 4 - esx and vcenter server.

[57] VMware. vsphere networking.

[58] VMware. Nsx administration guide, Nov 2017.

[59] VMware. Virtualization technology and virtual machine software: What is virtualization?, Nov 2018.

[60] Vmware and P. A. Networks. Next generation security with vmware nsx and palo alto networks vm-series, 2014.

[61] L. Wang, S. Jajodia, A. Singhal, P. Cheng, and S. Noel. k-zero day safety: A network security metric for measuring the risk of unknown vulnerabilities. *IEEE Transactions on Dependable and Secure Computing*, 11(1):30–44, Jan 2014.

[62] C. Wueest. Threats to virtual environments. Technical report, Symantec, Aug. 2014.

[63] C. Yoon, T. Park, S. Lee, H. Kang, S. Shin, and Z. Zhang. Enabling security functions with sdn: A feasibility study. *Computer Networks*, 85:19?35, May 2015.

[64] F. Zhang, J. Chen, H. Chen, and B. Zang. Cloudvisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 203–216, New York, NY, USA, 2011. ACM.