Evaluation of Moving Target IPv6 Defense and Distributed Denial of Service Defenses

Peter L. DiMarco

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Joseph G. Tront, Chair
Amos L. Abbott
Randolph C. Marchany

November 8, 2013
Blacksburg, Virginia

Keywords: IPv6, DDoS, Security

Evaluation of Moving Target IPv6 Defense and Distributed Denial of Service Defenses

Peter L. DiMarco

(ABSTRACT)

A Denial-of-Service (DoS) attack is a network attack from a single machine that attempts to prevent the victim, the targeted machine, from communicating to other devices on the network or perform its normal tasks. The extension of these attacks to include many malicious machines became known as Distributed Denial-of-Service (DDoS) attacks. DDoS attacks cause an immense amount of strain on both the victim and the devices used to reach the victim. In reaction to these attacks, preexisting technologies were used as DDoS defenses to mitigate the effects. The two most notable defenses used are the firewall and Internet Protocol Security (IPsec). The technologies behind these defenses emerged over twenty years ago and since then have been updated to conform to the newest Internet protocols. While these changes have kept the technologies viable, these defenses have still fallen victim to successful attacks.

Because of the number of Internet connected devices and the small address space in Internet Protocol version 4 (IPv4), Internet Protocol version 6 (IPv6) was developed to solve the address space problem. With IPv6 however, there are new problems to address; therefore, these aforementioned defenses have to be further modified to accommodate the new protocol. Moving Target IPv6 Defense (MT6D) has been developed to attempt to leverage the new standard against DDoS attacks in the IPv6 arena. This research evaluates the DDoS prevention capabilities of the aging defenses relative to the newly developed MT6D to determine which defense is best suited to defend against these attacks for a variety of scenarios. The threat environment in this study is limited to Synchronize (SYN) flood, HTTP/GET flood, Denial6, Dos-New-IP6, and Slowloris attacks. Attacks on the MT6D key distribution mechanism are not considered. Strengths and weaknesses of the aforementioned defenses are presented and analyzed.

This project examines different metrics including the performance impact on the machines and the client throughput in an instrumented testbed. MT6D has high operating costs and low throughput compared to the other defenses. Under DDoS attacks, the firewall is unable to prevent attacks in IPv6 due to the inability to determine the same host from multiple Internet Protocol (IP) addresses. Overall, IPsec and MT6D effectively mitigate the DDoS attacks. Although, MT6D is susceptible to some attacks due to its operating at the guest level. At this point in MT6D's development, the difference in performance could be considered a reasonable price to pay for the added benefits from MT6D.

Contents

Abstract **ii**

List of Figures **vii**

List of Tables **viii**

List of Abbreviations **x**

1 Introduction **1**

1.1 Original Contribution 2

1.2 Organization 2

2 Background **3**

2.1 Denial of Service Attacks 3

2.2 Transport Control Protocol 4

2.3 Features of IPv6 6

2.3.1 Stateless Address Auto Configuration 7

2.3.2 Packet Format 8

3 Literature Review **10**

3.1 IPv6 Attacks 10

3.2 Novel Network Attack Defense 11

3.3 Firewall 12

3.4 Network Testing 12

4 DDoS Attacks **14**

4.1 SYN Flood . . . 14

4.2 HTTP Get Flood . . . 15

4.3 Slowloris . . . 16

4.4 Denial6 . . . 16

4.5 Dos-New-IP6 . . . 16

5 DDoS Defenses **18**

5.1 Firewall . . . 18

5.2 IPsec . . . 19

5.3 MT6D . . . 20

6 Testing Method **21**

6.1 Communication Types . . . 23

6.2 Test Variations . . . 24

6.3 Metrics . . . 24

6.4 Defense Variables . . . 25

6.4.1 Firewall . . . 26

6.4.2 MT6D . . . 26

7 Results and Analysis **27**

7.1 MT6D . . . 27

7.1.1 Client Overhead . . . 27

7.1.2 Performance . . . 29

7.2 Slowloris . . . 33

7.3 SYN Flood . . . 36

7.4 HTTP Get Flood . . . 39

7.5 Denial6 . . . 42

7.5.1 Destination . . . 42

7.5.2 Hop-By-Hop . . . 45

7.6 Dos-New-IP6 46

8 Conclusions **48**

9 Future Work **50**

Appendices **52**

A Master Scripts **54**

A.1 Master.sh 54

A.2 Collector.sh 63

B Attacker Scripts **68**

B.1 Attacker.sh 68

C Client Scripts **73**

C.1 Client.sh 73

Bibliography **78**

List of Figures

2.1 TCP Initialization Handshaking Process [TCP Handshake. https://commons.wikimedia.org/wiki/File:Tcp-handshake.svg. used under fairuse, 2013] . 5

2.2 TCP Finalization Handshaking Process [TCP Close. https://secure.wikimedia.org/wikipedia/commons/wiki/File:Fin_de_conexi%C3%B3n_TCP.svg. used under fairuse, 2013] 6

2.3 IPv6 SLAAC Message Exchange [Stephen Groat. Privacy and Security in IPv6 Addressing. Master's Thesis. Bradley Department of Electrical and Computer Engineering. Virginia Tech, April 2011. used under fairuse, 2013] 8

2.4 IPv6 Header Format [IPv6 Header. https://en.wikipedia.org/wiki/File:Ipv6_header.svg. used under fairuse, 2013] 9

4.1 SYN Flood Attack [TCP SYNFlood. https://en.wikipedia.org/wiki/File:Tcp_synflood.png. used under fairuse, 2013] 15

6.1 The Network Topology 22

7.1 Client Overhead Performance 28

7.2 Client Overhead Transactions 29

7.3 MT6D SYN Flood WGET 1M Performance 30

7.4 MT6D SYN Flood WGET 1M Transactions 31

7.5 MT6D SYN Flood WGET 1M Performance With Best Fit and Error Bars . 31

7.6 MT6D SYN Flood WGET 1M Transactions With Best Fit and Error Bars . 32

7.7 Slowloris WGET 50M Transactions 34

7.8 Slowloris WGET 50M Performance 35

7.9 Slowloris iPerf 1M Transactions 36

7.10 SYN Flood WGET 1M Performance 37
7.11 SYN Flood WGET 1M Transactions 38
7.12 SYN Flood iPerf 1M Transactions 39
7.13 HTTP Get Flood 50MB Transactions 40
7.14 HTTP Get Flood 50MB Server Performance 41
7.15 HTTP Get Flood 50MB Defense Performance 42
7.16 Denial6 Destination 1MB Transactions 43
7.17 Denial6 Destination 1M UDP Transactions 43
7.18 Denial6 Destination 1MB Defense Performance 44
7.19 Denial6 Destination 1MB Server Performance 45
7.20 Denial6 Hop-By-Hop 1MB Router 1 Performance 46
7.21 Dos-New-IP6 50MB Transactions 47

List of Tables

6.1 VMWare Host Machine Specifications 22
6.2 Virtual Machines Specifications 23
6.3 Variables in the Experiments 24

List of Abbreviations

ACK . . . . . .Acknowledgement

AH . . . . . .Authentication Header

ARP . . . . . .Address Resolution Protocol

ARQ . . . . . .Automatic Repeat reQuest

DAD . . . . . .Duplicate Address Detection

DDoS . . . . . .Distributed Denial-of-Service

DNS . . . . . .Domain Name System

DoS . . . . . .Denial-of-Service

FI . . . . . .filtering identifier

FIN . . . . . .Finished

GB . . . . . .Gigabyte(s)

HTTP . . . . . .Hypertext Transfer Protocol

IID . . . . . .interface identifier

IP . . . . . .Internet Protocol

IPsec . . . . . .Internet Protocol Security

IPv4 . . . . . .Internet Protocol version 4

IPv6 . . . . . .Internet Protocol version 6

LAN . . . . . .local area network

LOIC . . . . . .Low Orbit Ion Cannon

MB . . . . . .Megabyte(s)

MITM . . . . . .man-in-the-middle

MT6D . . . . . .Moving Target IPv6 Defense

MTU . . . . . .maximum transmission unit

NAT . . . . . .Network Address Translation

NDP . . . . . .Neighbor Discovery Protocol

NS . . . . . .Neighbor Solicitation

NTP . . . . . .Network Time Protocol

NUD . . . . . .Neighbor Unreachability Detection

OS . . . . . .operating system

OSI . . . . . .Open Systems Interconnection

PAC . . . . . .Pushback-and-Communicate

PMTUD . . . . . .Path MTU Discovery

PPTP . . . . . .Point-to-Point Tunneling Protocol

SLAAC . . . . . .StateLess Address Auto Configuration

SYN . . . . . .Synchronize

TCP . . . . . .Transmission Control Protocol

THC . . . . . .The Hacker's Choice

UDP . . . . . .User Datagram Protocol

Virginia Tech . . . . . .Virginia Polytechnic Institute and State University

Chapter 1

Introduction

The open nature of the Internet places private and sensitive information at risk when the information is transferred between machines. Transmitted information can be intercepted by unintended or malicious parties. Furthermore, malicious clients can prevent those who need the information from obtaining it. In this case, the attack attempts to obtain exclusive use of all available resources of a target that would normally be used to serve legitimate clients. Denial-of-Service (DoS) attacks cause the server to be unreachable or unable to process legitimate requests. A DoS attack is the term used to describe when there is a single attacker executing the attack. However, with the easy availability of computing and network resources, multiple attackers are able to readily coordinate an attack targeted on a victim and create a Distributed Denial-of-Service (DDoS) attack. In most cases, DDoS assaults result in communication being interrupted, but in extreme cases, they can lead to data or machine corruption.

Development on DDoS defenses was started by utilizing pre-existing technologies in response to the potential damage caused by DDoS attacks. The intent of the defenses was to prevent a DDoS attack from completely halting communication. However, a weaker, yet acceptable defense could partially succumb to the attack, as long as it maintained a reasonable communication rate with the legitimate clients. There is a need to re-evaluate the defenses to determine if they are still able to maintain communication after advances in technology have been made. In the case of Internet Protocol version 6 (IPv6) slowly becoming the Internet Protocol (IP) standard, it is time to re-evaluate our defenses to determine how effective they are against attacks under the new protocol. The firewall and Internet Protocol Security (IPsec), started by Columbia University and AT&T Bell Labs, were first were developed over twenty years ago then later used as a DDoS defense. These defenses may begin to show their age as the standard changes and they have to be re-adapted. The adoption of IPv6 to solve the address space problem with Internet Protocol version 4 (IPv4) brings new problems such as address management.

The purpose of this research is to evaluate a subset of the current DDoS defense mechanisms.

One defense that the thesis will also focus on is being developed at Virginia Polytechnic Institute and State University (Virginia Tech), named Moving Target IPv6 Defense (MT6D). The intent is to show this newly-developed defense, aimed at addressing the IPv6 protocol, is better suited to address the security needs for today's threats. The thesis will discuss techniques that demonstrate the ability of a DDoS defense system to mitigate the effects of a DDoS attack, whether completely or at some lesser level. Finally, the thesis will detail the best defense that is currently available for the new IPv6 protocol.

1.1 Original Contribution

Newly-developed technologies are evaluated to discern their applicability in the research field. MT6D has been tested under normal operating conditions, providing details of its ability to defend against hackers. The system has not yet been subjected to stress tests, specifically in the situation of DoS attacks. To evaluate the new defense, it will be compared against pre-existing defenses under different attack conditions. From these tests, the measurements will be evaluated to effectively determine the value of MT6D. In addition, this thesis analyzes IPv6 specific attacks that have been developed and discussed in recent literature but not yet evaluated. This work hopes to provide a contribution to the future development of IPv6 security.

1.2 Organization

The rest of the thesis consists of the following: Chapter 2 details technologies that will be referenced in this thesis. Chapter 3 reviews other work that has gone into the field of DDoS attacks and defenses. Under Chapter 4, the different types of DDoS attacks included in this research are discussed. Chapter 5 discusses the defenses being tested in this experiment, and Chapter 6 outlines the test environment used. The Results and Analysis of the research are contained in Chapter 7. Conclusions and Future Work in Chapters 8 and 9 conclude the thesis.

Chapter 2

Background

DoS attacks have been around for many years. These attacks have been documented and the damage from them has made headlines. Under IPv4, there were many vulnerabilities that hackers would exploit on both the network layer and the application layer. The most common exploit exists within the Transmission Control Protocol (TCP)[27]. With IPv6, these vulnerabilities still exist. Due to new features in the protocol, especially StateLess Address Auto Configuration (SLAAC)[30] and the new packet format, new vulnerabilities have emerged.

2.1 Denial of Service Attacks

The first well-documented DoS attacks occurred in 1974[15]. These attacks were developed by hackers to disrupt communication between a client and a server. They would be targeted against a victim machine, but can lead to other machines being affected. Depending on the attack, the victim could fail to provide a single service or fail to provide any network connectivity at all. There are many kinds of attacks that fall under the DoS category, some of which only affect the targeted machine and others which affect the targeted machine along with the machines that form the path from the attacker to the victim. When the path to the victim is affected by these attacks, any number of web services can be affected for different users. The most common attack type is to flood the victim with requests so that it is not able to respond to legitimate traffic.

Normally, these attacks would be targeted at high profile web servers such as banks[19] and other electronic commerce institutions[5], though these attacks can be applied to any web service as the flaws are inherent in the protocols. To circumvent the problem and provide better service, these high profile web servers began to maintain a large number of redundant machines to handle the volume of requests made by clients. A single malicious machine was not able to greatly affect the service while performing a DoS attack because of this strategy.

The DoS attacks were then further extended to include multiple attack machines which are referred to as DDoS [22] attacks. Under DDoS attacks, the intent is to still disrupt the victim's services by means of occupying its resources — though now the attacker is not a single machine, but rather a distributed cluster of machines more commonly referred to now as a Botnet[20], all performing the attack targeted at the victim. Botnets can comprise formerly compromised machines (zombies[8]) or willing attack participants that are all at once utilized to execute an attack. The probability that the targeted machine will fall victim to the attack is much greater when many machines are executing the DoS attack.

2.2 Transport Control Protocol

TCP is responsible for connection-oriented communication. The protocol provides reliable, ordered, and error-checked communication between two machines. It is deeply integrated within the IP and operates at the Transport Layer of the Open Systems Interconnection (OSI)[12] model. TCP is critical to many applications that need to transport reliable data, such as web servers, where the data integrity is essential to its functionality.

To provide this level of guaranteed data delivery, a pre- and post- transmission handshaking process is done between the communicating machines. The three-way handshaking process is depicted in Figure 2.1. The client first sends a Synchronize (SYN) packet to the server it wishes to start communicating with. The server then sends a SYN-Acknowledgement (ACK) packet to the client. Finally, the client sends an ACK packet to the server, completing the handshaking process. The client and server are now set up to send reliable data to one another.

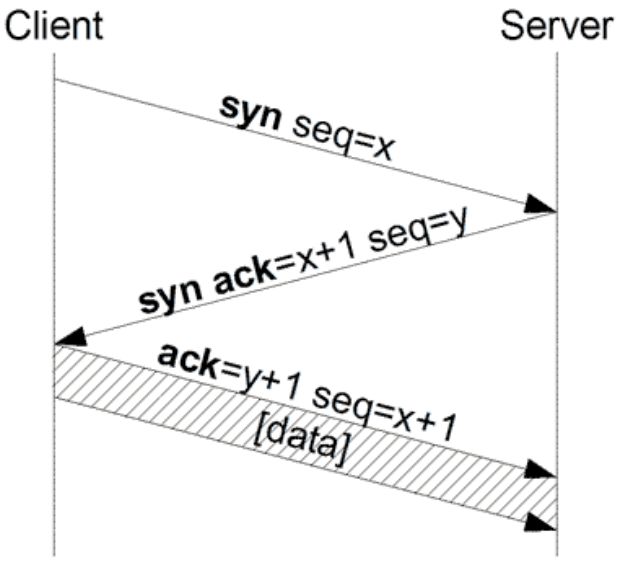

Figure 2.1: TCP Initialization Handshaking Process [TCP Handshake. `https://commons.wikimedia.org/wiki/File:Tcp-handshake.svg`. used under fairuse, 2013]

Similarly to the initial handshaking process, when the two machines want to close their connection, they perform another handshaking process depicted in Figure 2.2. The process is relatively the same as the initial handshaking process, with the exchange of SYN packets to Finished (FIN) packets. Until this exiting handshaking process is done or a timeout value is reached, the connection remains opened.

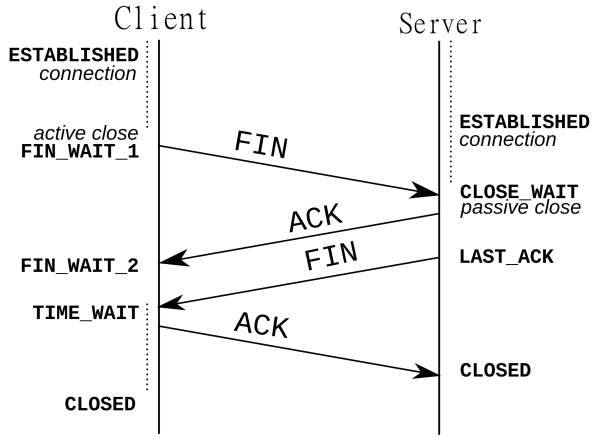

Figure 2.2: TCP Finalization Handshaking Process [TCP Close. `https://secure.wikimedia.org/wikipedia/commons/wiki/File:Fin_de_conexi%C3%B3n_TCP.svg`. used under fairuse, 2013]

TCP remains unchanged under IPv6 from IPv4. The implementation of TCP was only changed to accommodate the changes to packets, as discussed later. Not changing the protocol leaves it vulnerable to the same exploits that have been found. These exploits will be exposed in this work.

2.3 Features of IPv6

Internet Protocol version 6 (IPv6)[7] was developed primarily to solve the IPv4 address space problem. IPv4 addresses are comprised of 32 bits, providing 2^{32} IP addresses, which allows for approximately 4.3 billion unique addresses. Due to the increase in devices connected to the Internet, this allotment of addresses has become depleted. IPv6 address space was extended to 128 bits, providing 2^{128} or approximately 3.4×10^{38} addresses. This allotment of addresses is calculated to be approximately 7.9×10^{28} as many unique IP address as IPv4. This feature eliminates address space problems for the foreseeable future. There are other pertinent features that were introduced in IPv6 that will be referenced later in the research such as address management and packet options.

2.3.1 Stateless Address Auto Configuration

A new mechanism for assigning addresses within a subnet has been implemented in IPv6 to address the greater address space. This new mechanism allows for the machine joining the subnet, limited to subnets with a 64 bit address block[13], to determine its own address and is referred to as StateLess Address Auto Configuration (SLAAC)[30]. To facilitate this process, IPv6 contains the Neighbor Discovery Protocol (NDP)[25] which dictates the message types and sequences for determining a valid address assignment. The NDP serves many functions such as SLAAC, discovery of other nodes on the link, determining the link layer addresses of those nodes, duplicate address detection, address prefix discovery, finding available routers and Domain Name System (DNS)[23] servers, and maintaining this information. NDP replaces the Address Resolution Protocol (ARP) used in IPv4.

The SLAAC process is very simple and is illustrated in Figure 2.3. The process starts with the node reserving its link-local address, which is an address for communication between machines on the same link. The node then sends out a NDP solicitation message to the router for a router advertisement message or waits for the periodic router advertisement message for the subnet information. The subnet information delivered includes the network portion of the IPv6 address and other optional information, including the DNS servers and Network Time Protocol (NTP)[11] servers. To determine the second half of the node's IPv6 address, the node automatically configures an address, referred to as its interface identifier (IID) of the address. The node then combines these two pieces of information and attempts to allocate that address on the network. If the address is already allocated on the network, then the node goes through the Duplicate Address Detection (DAD) procedure and attempts the process again with a different IID. This process was designed to offload the administration of address allocation from the router to the entire subnet and is now included in the protocol. SLAAC allows for clients in a subnet to manage the available address space through mutual communication, which seen later will open up the protocol to exploitation from malicious clients.

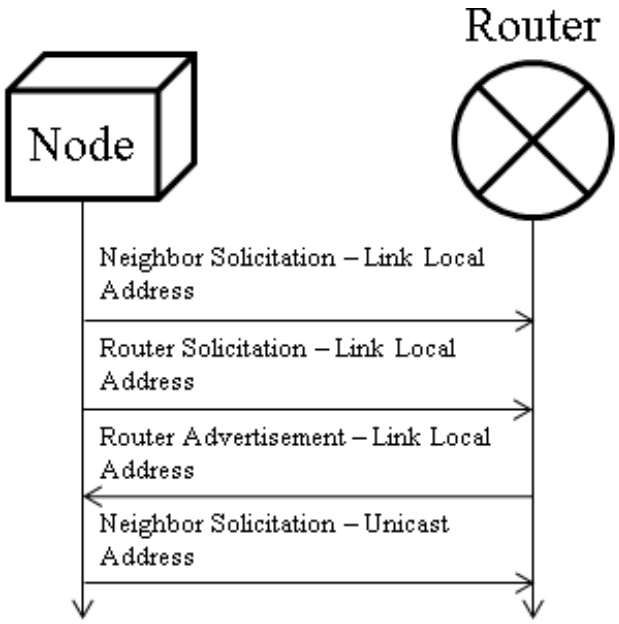

Figure 2.3: IPv6 SLAAC Message Exchange [Stephen Groat. Privacy and Security in IPv6 Addressing. Master's Thesis. Bradley Department of Electrical and Computer Engineering. Virginia Tech, April 2011. used under fairuse, 2013]

2.3.2 Packet Format

To make the packet more generic so that it could be further extended in the future, the IPv6 standard modified the structure of the IPv4 packet format. The IPv6 header[7], shown in Figure 2.4, consists of a fixed size block with minimal data required to detail the following payload for all packet types. This fixed size header contains the source and destination addresses, traffic classification options, the hop counter, and the type of optional extension or payload which follows the header. Designing the packet format in this manner allows for future development of options to be added to the IPv6 packet. The IPv4 packet did not have this functionality and packets had to be extended differently to accommodate features by passing them up to higher level layers. Unlike IPv4, the IPv6 protocol does not allow for packet fragmentation. It uses Path MTU Discovery (PMTUD)[24] to determine the largest maximum transmission unit (MTU) size between two hosts. The method relies on TCP to probe the path with progressively larger packets to determine the MTU[21], allowing for the entire packet to be processed at the maximum rate rather than in non-optimal pieces. These changes allow for greater extensibility of the IP and simpler processing.

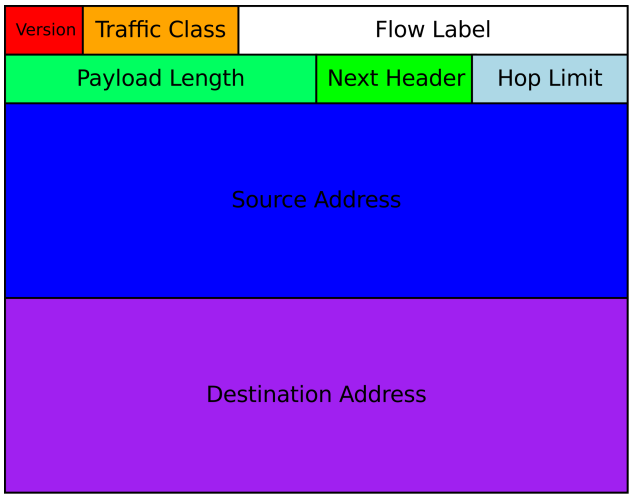

Figure 2.4: IPv6 Header Format [IPv6 Header. `https://en.wikipedia.org/wiki/File:Ipv6_header.svg`. used under fairuse, 2013]

Chapter 3

Literature Review

For many years now, substantial amounts of work have been put forth in the field of network security. Several different network protocols have been researched, exposing vulnerabilities which can be exploited to produce different adverse effects. The pertinent network security work analyzed here inspects exploits used to cause damage to a victim machine and the connected network. These works fall under a few categories: network defenses, network attack mitigation, IPv6, and network testing. This research differs from past efforts by comparing a newly developed defense mechanism against known network attack defenses.

3.1 IPv6 Attacks

IPv6 has many vulnerabilities that are carried over from IPv4 and also contains new vulnerabilities. The loose knit group "The Hacker's Choice" (THC)[1] developed the first set of IPv6 protocol attacks. THC describes in their presentation that there are several IPv6 attacks that are new to DoS research. With a focus on host-based attacks, the important attacks presented are related to exploiting the NDP. THC inspects the NDP and exposes vulnerabilities with the DAD, allowing a malicious client to intercept or terminate communication to a local area network (LAN). Some of these attacks has been incorporated into this project for further evaluation of their effect.

Generic DDoS attack exploration under IPv6 has also been done. Yang[33] goes into detail on the attacks that are exploitable under IPv6. This includes previous exploits that were carried over from IPv4 and new problems introduced by the IPv6 protocol. Yang provides a comprehensive list of attacks and analysis thereof under simple network implementations, without firewalls. While there still may be additional exploits in IPv6, a lot of work has already gone into compiling the following list of known exploit categories:

- Duplicate Address Detection Attack

- IMCP Smurf
- IMCP-Flood
- Neighbor Soliciation/Advertisement Spoofing
- Router Discovery Attacks
- TCP-Flood
- UDP-Flood

This work will utilize a subset of these categories of attacks to evaluate the proposed DDoS defenses.

3.2 Novel Network Attack Defense

Due to the potential damage from DDoS attacks, the need to mitigate them is apparent. Research on prevention has fallen into attack specific prevention and general DDoS defense improvement. Previous works approach the problem from both a macro and micro scale to prevent these exploits from damaging networks. The literature survey is focused on network layer based detection methods over the past few years. There was a focus on defenses that would prevent attacks discussed in Chapter 3.1.

Yatagai [34] introduces two methods of detecting an Hypertext Transfer Protocol (HTTP) Get Flood attack. The first method detects an increased number of requests for pages, based on the idea that a DDoS HTTP Get Flood would replicate the same order repeatedly. This method establishes patterns and recognizes if many requests fit that pattern to detect if the requests are part of a DDoS attack. The other method compares the browsing time versus the amount of information requested. The idea behind it is to see if large amounts of data are being requested but with small time intervals between requests. The algorithm assumes a ratio of request size to time between requests. When the ratio is smaller than the expected value set by the defense, it interprets these requests as a DDoS attack. These algorithms each have their own advantages and disadvantages based on the priority of the system and are subject to false-negatives and false-positives. Ranjan[28] and Xie[32] also present defenses targeted at preventing application layer attacks. Both research works present methods of preventing the application layer floods using past behavior detection.

Sun [29] details the $SACK^2$ defense against SYN Flood attack. $SACK^2$ attempts to match packets used in creating the TCP connection that a SYN Flood attack attempts to disrupt. By detecting the packets of a previously failed connection, it can try to filter out attempts that appear malicious. The algorithm detects many failed TCP handshaking attempts as a

DDoS attack and not as an unlucky client. Although $SACK^2$'s detection method is effective, it is susceptible to false positives and false negatives.

Trung [26] demonstrates a new means of preventing DDoS attacks using a method called Pushback-and-Communicate (PAC). PAC uses proprietary messages to tell routers closer to the attack source to filter attacks, thus distributing the attack amongst several routers and not just the edge router to the server. This process begins when the victim detects a DDoS attack, but there is not information on how this process is completed. This defense requires the proprietary code to be distributed to the intermittent routers in order to be effective, but the work states that it does not require all routers to deter an attack. A similar defense presented by Chen [6] provides the same type of defense. This defense also requires additional code on the intermediate routers which is outside the scope of this project.

3.3 Firewall

The concept of a firewall was developed to limit packets to and from a network based on rules set by an administer. The firewall's rate-limiting abilities make it a great tool to use against DDoS attacks. Research has gone into increasing the effectiveness of the firewall against these network attacks. Because the firewall was not developed as a DDoS defense, but rather a network limiter, the research work does not apply strictly to DDoS research. The research discussed here was limited to recent works that improve the performance or effectiveness of the firewall with respect to DDoS prevention.

Shen [17] attempts to improve the firewall defense implementation by including support for stateful multicast communication. The paper's findings do show that the extension can allow this extra feature while mitigating attacks against the defense, though multicast DDoS attacks are a small subset. Liu also demonstrates ways of improving the firewall defense by attempting to optimize the rules assigned to it in [18]. The work details a technique to make a firewall query take less time, which becomes more prevalent as the number of needed firewall rules increases.

3.4 Network Testing

In general, all the research that goes into evaluating DDoS attacks and defenses includes a means of testing the defenses. Quantifying the results of research in this area is crucial to gaining a better understanding and allowing users to decide on how to choose from among the trade-offs that inevitably come with any defense. The following research work is limited to testing existing DDoS mitigation technologies that are also evaluated in this thesis.

Zargar[35] presents an empirical method of evaluating a DDoS defense against flood attacks. The paper discusses the location of defenses and how that relates to different aspects such

as the scalability and system performance. This paper provides useful insight into how to determine the effectiveness of a DDoS defense based on its implementation.

Badishi[2] provides results on using a filtering identifier (FI) to prevent DoS attacks. They demonstrate transfer achievement rates of the systems to convey the effectiveness. This study is different because of the focus on host-based systems. Wen and Barylski, [31] and [3] respectively, evaluate the performance of IPsec. Both papers discuss communication throughput of the implementation while under varying conditions. These works are partially recreated under different conditions in our experiments.

Chapter 4

DDoS Attacks

DDoS attacks fall into different categories: volume based attacks (floods), protocol attacks, and application layer attacks. This research intends to analyze the effects of different attacks in these categories that are considered host-based in order to analyze the effects of DDoS attacks against host-based defenses. These different attacks expose different vulnerabilities on the host to disrupt communication to clients chosen from categories specified in Section 3.1. In this chapter, the attacks used are discussed to convey the technologies they exploit.

4.1 SYN Flood

The SYN Flood attack is a volume-based and protocol DDoS attack. This attack, visualized in Figure 4.1, intends to exploit the TCP handshaking process described in Chapter 2.2. A hacker is able to manipulate the protocol to his advantage in order to allocate resources on the victim. The TCP handshaking process has a half-opened state which occurs when a machine has sent the SYN packet to the intended recipient. These half-opened states have a timeout value associated with them in case communication is lost during the handshaking process. Once the victim machine receives the SYN packet from the malicious machine, the timeout timer is then started while the machine waits on the corresponding ACK packet. Because the hacker does not intend to complete the connection, he is able to move on and send another SYN packet relating to a different connection. This allows the hacker to theoretically allocate as many TCP connections on the server as SYN packets he can send out during the timeout value. The number of packets can be very great since the SYN packet is very simple and quick to craft, and the timeout value is a relatively large number (a default 60 seconds in the Ubuntu Linux distribution). Due to the limitations on the number of concurrent TCP connections, a system can become fully utilized in respect to TCP. Once a system is fully utilized, legitimate clients are unable to create a TCP connection, eliminating communication to several services. This attack is very prominent among various documented

DDoS attacks, research, and is included in the Low Orbit Ion Cannon (LOIC) hacker toolkit, utilized by the well-known Anonymous Hacker group.

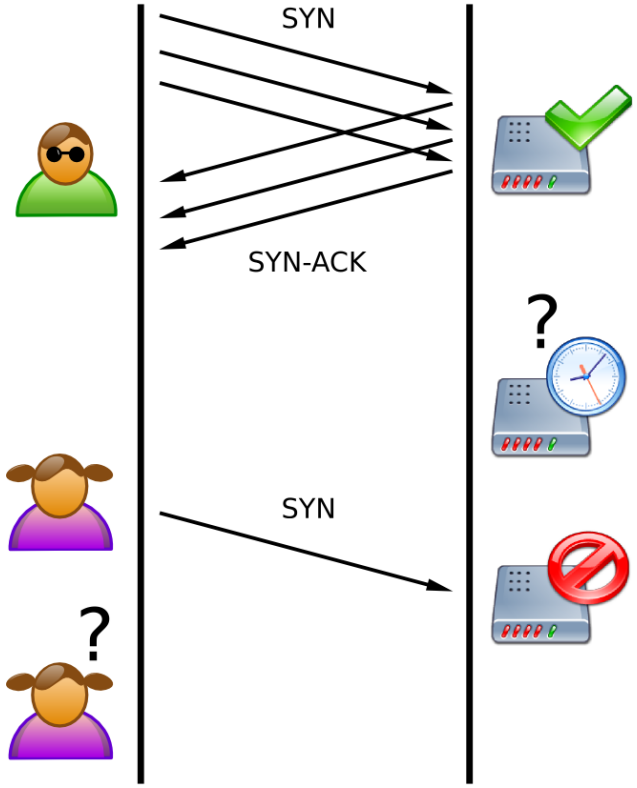

Figure 4.1: SYN Flood Attack [TCP SYNFlood. `https://en.wikipedia.org/wiki/File:Tcp_synflood.png`. used under fairuse, 2013]

4.2 HTTP Get Flood

The HTTP Get[10] Flood attack is the second half of the LOIC toolkit. This attack is also a flood attack, in addition to an application layer attack. It exploits a feature of the Web Server application. The Web Server allows the use of an HTTP-GET function for a client to retrieve data that is located on the server. A hacker exploits this service by sending lots of these requests, usually for the default data or data that does not exist. These requests become queued on the server and resources must be allocated to process this queue. When there are large amounts of requests queued, the Web Server process will need additional run time and/or processing power, affecting other services on the same machine. This idea alone can strain the Web service, but there is an additional side-effect: the attack also allocates TCP connections for sending the requests. When the attack allocates many TCP connections, the

attack can also prevent clients' requests at the protocol layer. A hacker will normally spawn many requests at once from a single machine, requiring only a few computers to perform a DDoS.

4.3 Slowloris

Slowloris is another attack aimed at the application layer, specifically Web Servers. This attack also exploits the timeout feature citerfc5482. Web Servers contain a timeout value, albeit different from the system TCP timeout, that determines when to close a connection with a client. The Slowloris attack exploits this by sending the minimal amount of data required to keep a connection opened. The attack then multiplies this process with as many unique connections as the Web Server will allow him to have. Because the attack only sends the minimal required amount of data, a hacker is able to allocate all the resources a server has available with very little machine power required. After the Slowloris attack has consumed all available victim connections, it continues to request additional connections to account for client connections ending. Overall, this attack will consume less processing power on the server than legitimate communication, making it harder to detect the attack as compared to flood attacks that result in high resource allocation.

4.4 Denial6

Denial6 is the first IPv6-specific attack utilized in this research and is part of the THC toolkit. This attack exploits the IPv6 header described in Chapter 2.3.2. The specific feature it exploits is the options extension. There are two kinds of options extensions, *Hop-by-Hop* and *Destination*. The *Hop-by-Hop* options extension header needs to be examined by routers on the path from the attacker to the victim, whereas the *Destination* options extension is only examined by the victim host. The large amount of unknown options requires the machine to use more processing power and/or time to process the packet, increasing the chance of the network queue filling up. This attack has the ability to flood the victim with different, large packets with either type of options extensions. Denial6 attempts to allocate the resources on both the intermittent machines and victim machine to prevent communication. To separate the effects of *Destination* and *Hop-by-Hop* exploits, Denial6 will be compared separately based on the type of option extension.

4.5 Dos-New-IP6

Dos-New-IP6 is the second IPv6 specific attack as part of the THC toolkit, and is also a protocol attack. This attack exploits the NDP described in Chapter 2.3.1. The joining

machine publicizes its intended IP address to determine if it has already been allocated on the network as part of the SLAAC process. It expects to hear from a machine on the network if this address is already allocated. Dos-New-IP6 exploits this by always telling machines that their desired IP addresses are already allocated. This will stop new machines from being able to enter the network. But the NDP also publicizes the machine's IP address intermittently on machines that already have an IP address; so after the specified interval between publicizing the victim's IP address elapses, if there is a malicious machine on the network running this tool, the victim will lose network connectivity. This attack would be able to end all network activity on a subnet after running for enough time.

Chapter 5

DDoS Defenses

DDoS attacks need to be handled to prevent interruption with the high reliance on web services. Because of the IPv6 features, there is a new idea of a mobile host. This mobile host may rotate between subnets while still maintaining its IID, making it locatable. This study focuses on host-based defenses that would be used to prevent DDoS attacks wherever the host may originate from because of this new mobile host. This research includes: the developed DDoS defenses (IPSec and firewall) and the new technology, MT6D.

5.1 Firewall

The firewall was first developed in 1988[14]. It gave the network administrator means of filtering packets both to and from a network. It allowed administrators to prevent outside clients from accessing internal network machines. The firewall filters traffic by comparing it against a set of rules set by the network administrator. When a packet is received, it is compared against the rule set to see if it matches a rule's parameters. If so, the packet can be accepted or dropped as specified. When the firewall was first developed, it allowed for basic rule parameters. Since then the firewall has been further developed to allow more mitigation and more complex rules. These improvements now enable administrators to maintain state, allowing them to formulate rules based on previous packets received from a host or targeted for a certain application. This turned the firewall from being known as state-less to stateful. There is also inclusion of application layer knowledge for well-known services, enabling detection on invalid use of a port or protocol. The firewall is able to do analysis on network traffic based on the state and application, enabling administrators great control over their network.

After the first DDoS attack was documented in the year 1999, the firewall was a great candidate for mitigating these attacks. The firewall was already able to mitigate the attacks based on rules before any work towards preventing these attacks was put forth. Rules were

able to be created to throttle traffic from IPs in an attempt to prevent flooding attacks and restrict traffic to a certain number of connection per host. The extensive rule options such as host and destination blocking, rate-limiting, and blocking based on previous traffic gave the administrators a tool to detect the DDoS attacks. Although this can lead to ambiguity between implementation rules and relies on the network administrator's knowledge. For this project, the standard version of IP6Tables distributed with the Ubuntu linux distribution was used.

The firewall is susceptible to several vulnerabilities. The foremost is the lack of preparation from the network administrator. A lack of rules to address vulnerabilities leaves machines behind the firewall susceptible to attack. Rate limiting is able to be overcome by brute force, allocating all the allowed connections within the period set by the rules. Finally, alongside with rate limiting is the IP connection limiting, preventing a single IP from taking up the available resources, a feature that can be further exploited in IPv6.

5.2 IPsec

IPsec was developed for securing IPv4 communications. It was first developed in 1993 then later included in the IP protcol[16]. IPsec provides IP packet authentication and encryption. The implementation does this by wrapping the headers with an IPsec Authentication Header (AH). This header contains the necessary information to confirm the packet is intended for the receiving machine.

IPsec provides end-to-end security and is implemented inside the Internet layer of the IP Suite, the lowest layer before hardware. The protocol uses a shared secret between two machines to use as the key for the encryption of packets. This shared secret is used to encrypt and decrypt traffic based on the IPsec options within the IPv6 header. IPsec is quickly able to determine unintended traffic with the knowledge that a source and encryption key are pairings. The traffic is mostly hidden from observers except for the header information, which alludes to the fact that IPsec is currently being used.

All traffic is handled regularly except that the defense machine encapsulates the packet with its address for the IPsec packet. This allows machines behind the IPsec defense machine to drop in and out of the network without needing to be configured for a specific location. This also hides the topology of the network behind the defense but gives attackers a single point of attack.

This type of security prevented man-in-the-middle (MITM) and replay attacks, furthermore it could prevent against DDoS attacks as well. IPsec allows communication to known clients, enabling it to easily discard unwanted traffic. The attacker's packets are dropped at the Internet layer, preventing application or protocol layer attacks. Given that IPsec has a known client list, it is able to mitigate attacks based on packet inspection.

5.3 MT6D

MT6D[9] was developed at Virginia Tech and intends to leverage the IPv6 protocols to provide DDoS mitigation. This defense was developed to work amongst known point-to-point clients and provide them protection against DDoS attacks, as well as protection from being located by hackers.

IPv6 provides new opportunities in the field of DDoS defenses, the first of which being the greatly-expanded IP address space. MT6D utilizes this vast space to provide the defense and clients the ability to move through their respective subnet address space, changing its address at specified intervals. This feature renders a client hard to discover, mitigating efforts to track the device or monitor its usage. Rotating addresses is feasible under IPv4 but due to the small address size, is not ideal. In addition to the address rotation, MT6D employs encryption to prevent MITM attacks when the obscured addresses are known. The MT6D implementation extends the known qualities of IPsec to further protect the clients from attackers.

Since an attacker may still be able to locate the machine, even if only temporarily, the defense must still mitigate an attack. First, MT6D borrows a technique from IPsec with authentication headers. MT6D encapsulates the original packet after encryption inside a proprietary MT6D packet. These headers can be processed quickly on the destination machine to determine authenticity. This allows for quick processing of unintended or malicious packets. Secondly, to allow for the machine to operate uninterrupted after rotating addresses, the NDP is utilized. The previously-used address is unallocated after rotating, so if an attack was being performed on the machine, Neighbor Unreachability Detection (NUD) messages would be sent to the attacker thereafter. This allows the machine to recover from an attack and spreads part of the defense over the routers using the protocol. The MT6D provides old and new security features to a field that is utilizing technology not developed for the intended purpose.

The current implementation of MT6D operates at the user level on the peer systems. This will have disadvantages as the operating system (OS) will have more impact during the processing of packets. Currently, there is work in progress to put MT6D inside the network stack of the Linux kernel. This would enable the defense to operate at a level comparable to IPsec and result in better performance for the defense.

Chapter 6

Testing Method

Possible methods for achieving evaluating MT6D range from mathematical modeling and analysis of the protocols and attacks involved to construction of physical testbeds and full implementation of attacks on the realized system. Queueing theory analysis is not considered feasible, given the complexity of protocols, network structure, and attacks involved. Software simulation of the entire structure is possible, but would require validation of the simulated behavior. The selected approach was to construct a physical testbed to compare results on a realized system because of these limitations. This approach avoided significant validation issues by using actual MT6D components. Although this may limit applicability of the results to larger scale network and attack scenarios.

This project uses a small virtual testbed designed to replicate real-world DDoS attack scenarios contained within a small environment. The testbed was kept small to allow for the different permutations, detailed later in this section, of test configurations to be tested within the project time-frame. Figure 6.1 depicts the simple yet extensive testbed used to determine the effectiveness of the DDoS defenses. The figure depicts the several components used, which are all equal machines detailed in Table 6.2. The five different components are: Server, Defense, Clients, Attackers, and Routers. It is possible to implement these defenses on the server itself; but to separate the operating costs of the defense from the server requests, they are separate boxes. Table 6.1 details the machine specifications of the VMWare host which all the virtual machines run under. Table 6.2 details the configurations of the different machines within the testbed.

Table 6.1: VMWare Host Machine Specifications

Manufacturer	Dell Inc.
Model	Poweredge R620
CPU Cores	Intel(R) Xeon(R) CPU E5-26700 @ 2.60GHz
Memory	96 GB RAM
Hard Disk Drives	10K SAS HDDs
VMWare Edition	vSphere 5 Essentials Plus

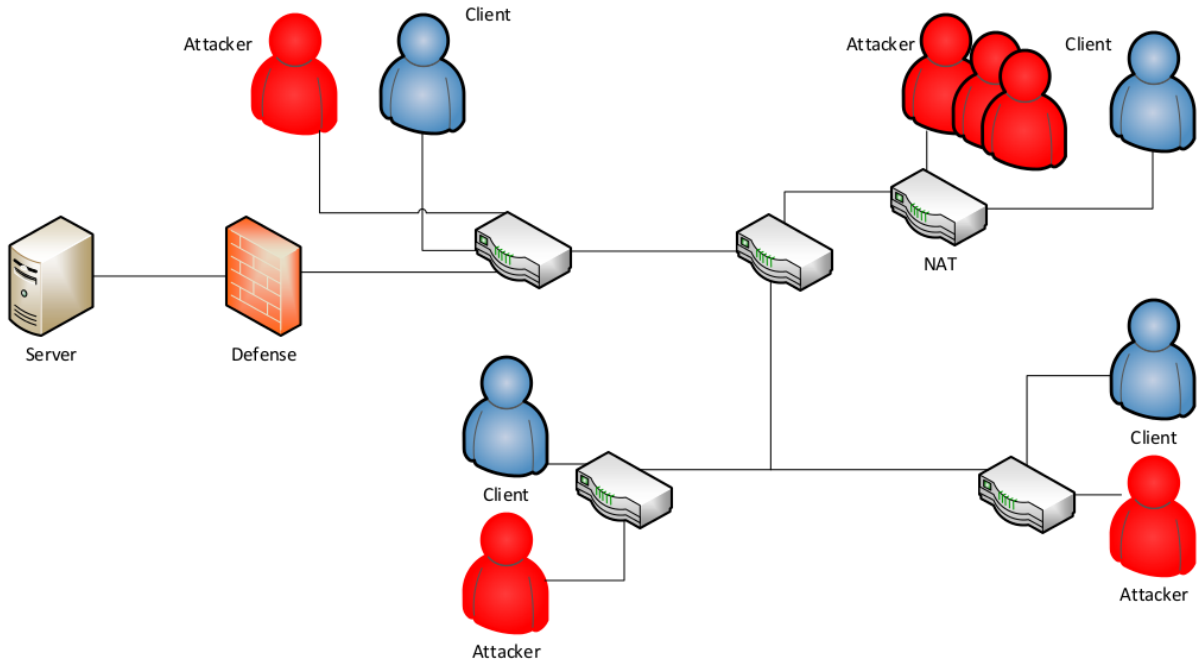

Figure 6.1: The Network Topology

Clients and attackers in the testbed were placed at several different levels from the server to emulate the Internet topology. Some were put at one hop away from the server to represent same subnet traffic, others placed several hops away to represent far reaching machines. The Internet topology is more aptly represented by the testbed by scattering the clients and attackers amongst varying distances in the system.

A special component of the setup is the inclusion of IPv6 Network Address Translation (NAT). NAT provides the ability to mask machines behind an edge router utilizing NAT with the router's outward facing IP address. This feature hides the network topology behind the edge router from outside viewers. While NAT has been obsoleted by RFC 4966, it is

Table 6.2: Virtual Machines Specifications

All Machines	
CPUs	1 Single Threaded Core
Memory	1 Gigabyte(s) (GB) RAM
OS	Ubuntu Server 12.04.1
Additional Services	SSH Server
Server, Defense, Routers	
Additional Services	MPStat
Server	
Additional Services	Apache2 Web Server, iPerf
Defense	
Additional Services	Respective Defense Service
Attackers	
Additional Services	The attacks described in Section 4

still functional under IPv6 and may be utilized if desired. NAT does provide some useful features such as security by hiding the private network behind it and functionality e.g., a load balancer. IPv6 NAT became officially supported in the Linux Kernel as of December 2012. It provides an added function against which to test the defenses.

6.1 Communication Types

The project includes both connection oriented and connectionless communication to further evaluate the DDoS defenses described in Section 5 under more environments. A goal of the project is to evaluate the defenses under many conditions, and the difference between how TCP and User Datagram Protocol (UDP) communication will be affected under defenses such as firewall.

The server includes the standard implementation of Apache2 that comes with Ubuntu distribution to incorporate TCP communication. The server included a mock website to emulate a content host knowing that many of the DDoS host attacks are targeted against websites. Modifications have been made to lower the available web service application layer processes to tune it to the intended client size, given the number of clients included in the testbed. The communication consists of HTTP Get requests for several differently-sized files to convey the difference between long- and short-lived connection-oriented communications. With connection-oriented traffic, there are many factors associated with the attack that affect the resources such as processor utilization and socket allocation, whereas UDP traffic is only affected when a machine is unable to send packets. This make it much easier to see the effects of a DDoS attack affecting the communication throughput under TCP communication.

The testbed also included connectionless UDP traffic to further emulate a real world network. For UDP communication, the iPerf toolset is utilized. The iPerf toolset runs its own server apart from Apache that measures different aspects of the communication. It enables the testbed to examine UDP communication at different total transfer sizes, allowing the same types of communication tests performed using TCP and Apache under a UDP environment.

6.2 Test Variations

With the testbed configuration, communication types, the attacks described in Section 4, and defenses described in Section 5, there are test variables listed in table 6.3.

Table 6.3: Variables in the Experiments

Number of Attackers	0-6
Number of Clients	0-3
Type of Attack	SYN Flood, Slowloris, HTTP-GET Flood, Dos-New-IP6, Denial6
Type of Communication	UDP, TCP
Communication Size	1 Megabyte(s) (MB),50 MB
Type of Defense	No Defense, firewall, IPsec, MT6D

The number of attackers and clients is determined by what the testbed able to support, which was kept small to limit scope. The attacks chosen were described in Section 4. The types of communication were covered in Section 6.1, and the transfer sizes of the communication times were chosen to reflect short- and long-lived communication. This allows variance in the amount of time the client is spent starting and ending connections versus transferring data. Defenses listed in the table are covered in Section 5; the no defense was added to include a baseline from which the defenses could be compared against in addition to each other. The tests executed will include all the different permutations of these variables. The test executables, seen in Section 9 are meant to facilitate this testing.

6.3 Metrics

The ability to determine the effectiveness of a DDoS defense system is the basis of the project. The project examines two different measurements, the performance impact on the machines and the client throughput in the testbed. The project will determine how effective a defense is at protecting the communication services with these metrics.

The first metric considered is the client throughput in the testbed. The amount of transactions that can be completed in a given amount of time by the combined effort of all the

clients in the testbed is recorded during an experiment. To accomplish this, a counter is kept local to the test and is incremented for every successful transaction achieved by the client. The counters are then aggregated for all the clients for a given test. The amount of time is held constant regardless of the client interaction type. Therefore it is expected that large data transfers will complete fewer transactions due to the increased transfer time. Then the type of attack and client transactions is varied, as discussed in Chapters 4 and 6.1, respectively. The ability for a defense to maintain high client throughput while under an attack is a favorable quality.

The other important means of determining the DDoS defense effectiveness is the impact on the supporting network machines, not only limited to the host defense machine. While the tests are executing, system performance measurements of the DDoS defense, the server, and the routers are recorded. To perform this, the MPSTAT unix function was utilized. MPSTAT command outputs the the activity in the system, most importantly the percentage of idle time. An example of the command is:

mpstat 1 TESTTIME

This command outputs the machine usage stats every second for the entire test time. This is examined to see how the communication (both attacks and requests) is affecting the machine. Since the attacks are executing at a constant rate for a given test, it is expected that the performance of the machine will remain constant for almost all of the test, after the initial change in performance at the start of an attack. The ability for all machines to maintain a low overhead from the network congestion of attacks and clients is the quality sought after.

During this project, the number of clients and attackers utilized varies between tests in the testbed to vary experimentation. Determining how the defenses, the routers, and the clients are affected, the different situations are examined for which best depict the qualities described as favorable. This encompasses the process of depicting the best DDoS defense. The best defense is determined as the one that provides a low processor overhead while still allowing high client throughput.

6.4 Defense Variables

The defenses in testbed require different setups to be effective. A pass-through box with no defense was used to represent the effects of client transactions and attacks on the system. IPsec requires minimal configuration with the only variable being the secret key used to encrypt the traffic, where different keys were used to differentiate clients. The other two defenses require more setup.

6.4.1 Firewall

The firewall is the standard IPTables implementation included with the Ubuntu Server distribution, which is a host-based implementation of the firewall. The firewall requires rules to be populated to filter traffic to its intended effect. The following traffic types were allowed to pass through the firewall:

1. ICMPv6 Protocol Packets, Including NDP Packets
2. Rate Limited: New Traffic Packets based on State, Including SYN Packets
3. Established Traffic Packets based on State
4. Valid Traffic

The first rule allows the IPv6 Packets required for normal operation to pass through to the server. This allows the server to discover other machines on the subnet, the router, DNS information, etc. The second rule allows a machine to attempt to open a new connection with the server. This item is rate limited to prevent flood attacks from affecting the server and to prevent any one attacker from consuming all available connections. The third rule allows traffic that is already established to continue operating uninterrupted. The final rule refers to the application layer knowledge the firewall has described in Chapter 5.1. This rule does not allow illegitimate traffic to the application. These rules allow the firewall to mitigate attacks from affecting the server.

6.4.2 MT6D

MT6D requires some additional variables to characterize its behavior as well. MT6D requires a secret key for performing encryption like IPsec. This key is also re-utilized in the address rotation to calculate the IP rotations. The differentiating variable is the address rotation period, or how long a particular IP address is utilized. Currently, MT6D only allows the period in seconds to be set to a power of 2. In the experiments, the period is set to the minimal allowed value of 2 to fully utilize MT6D to its extent. The low period also is expected to expose two other factors to their maximum: the packets dropped and conflicting IP addresses. Packets sent by one client to another may be received during a window where the two machines are not in sync with one another is possible with clock drifting and transport time. This will lead to lower total transactions in a system due to packet retransmission. During conflicting IP address occurences, all communication during the period will be lost as a client will send data to an unitended receiver. This is expected to be very rare due to the IP space available and to affect the system minimally even with the lowest allowed period.

Chapter 7

Results and Analysis

Testing network security tools to compare the protection offered requires many different tests and variations. In this section, scenarios are presented that highlight particular strengths and weaknesses of the defenses. Followed up with analysis on the reasons behind the defenses' reactions to DDoS attacks. These scenarios demonstrate the overall effectiveness of the defenses under the subset of internet attacks currently available. The results are organized into three different sections: results regarding different scenarios only concerning MT6D, results comparing MT6D to the aforementioned DDoS defenses, and IPv6 specific attack evaluations.

7.1 MT6D

The focus of this paper is MT6D in comparison to well known defenses. The defense will first be analyzed under normal operating conditions to better understand the how the defense reacts to DDoS attacks. Two aspects of the defense will be detailed: the overhead from the defense on the machine and the throughput of the defense under no attack. The analysis will provide a baseline for performance analysis against the other DDoS defenses.

7.1.1 Client Overhead

MT6D attempts to provide protection from attacks by both obscuring the address and mitigating incoming attacks. This protection can cause significant overhead on the defense system to maintain communication. Figure 7.1 shows the incremental performance for the additional clients in the system. Starting with no defense, there is little performance impact on the system forwarding the additional packets to support an additional client. IPsec shows a slim, but additional overhead derived from the need to encrypt and decrypt the

packets through the defense. The network level at which IPsec operates assists in keeping down the performance. The firewall overhead is slightly higher than IPsec, and here clients actually have a noticeable effect on the defense. This overhead is due to the additional packets to process against rules specified by the network administrator on the firewall. Last, MT6D shows a great performance impact to support additional clients. MT6D, currently a guest level implementation, is responsible for intercepting the intended packets, processing them, and putting the new or original packet back on the network queue. This overhead requires MT6D to perform a great amount of work to support an additional client. The current implementation of the defense requires much more powerful machines to support the additional overhead to communicate with clients in a network.

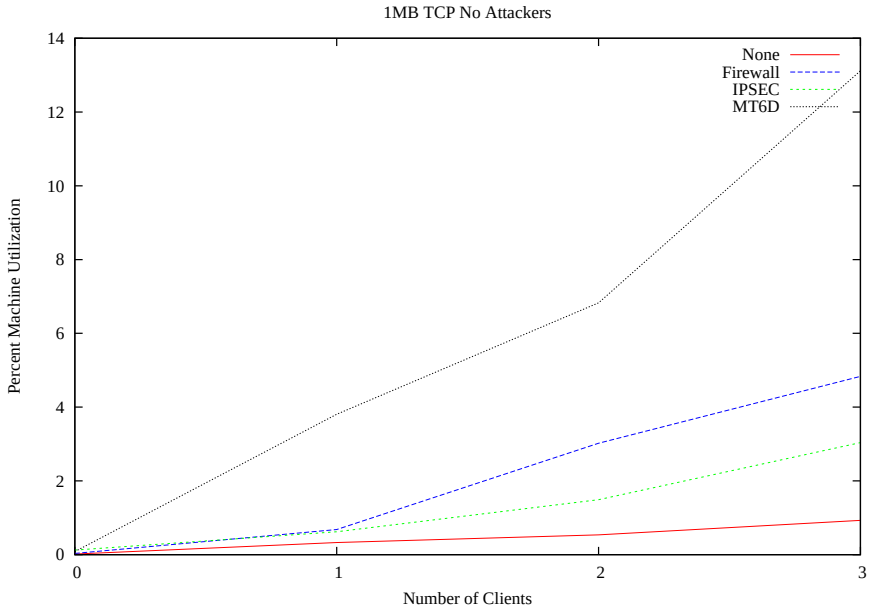

Figure 7.1: Client Overhead Performance

Figure 7.2 associates the relative client gains to the performance impact. MT6D follows the trend of the other defenses, but is not able to obtain the same number of transactions. The primary reason for this loss in completed transactions is packet loss and TCP accounting for loss packets. The TCP accounts for packet loss and the client will send Automatic Repeat reQuest (ARQ)s to tell the server to resend a packet. On top of the extra strain put on the system from encryption, encapsulation, and address generation, with clock drift there is a small window where packets are lost every rotation period. There is also the rare occasion where the address the defense or client wants to use is already allocated, in which the entire period of packets is lost. This effect will increase the variance between tests but is expected to be very rare given the address space available. These factors add up to the total transactions in the MT6D defense system being lower than the other defenses. When

there is no ongoing attack in the system, IPsec is able to perform the best, with moderate overhead for the additional clients and great communication gains, similar to no defense at all. The subsequent graphs will feature a static client count to focus on how the attack is affecting the testbed.

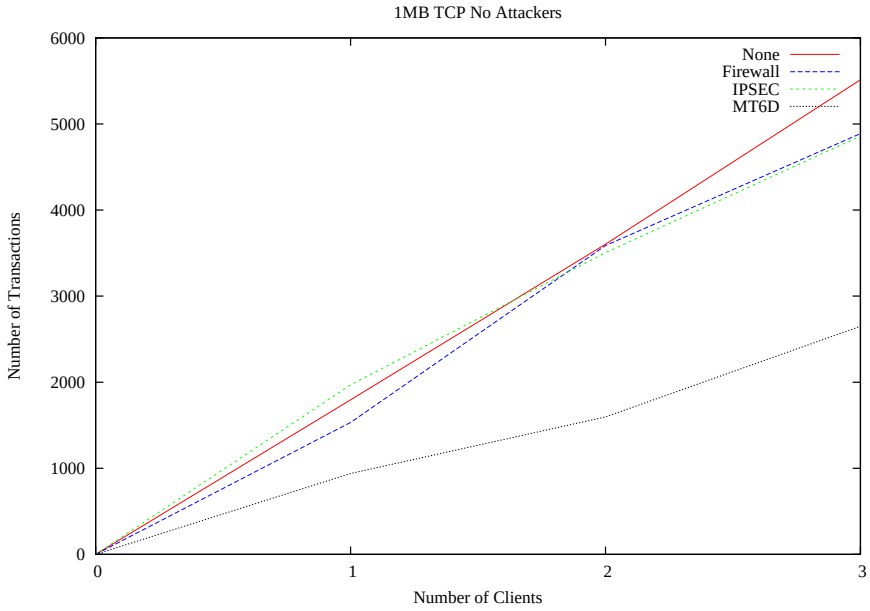

Figure 7.2: Client Overhead Transactions

7.1.2 Performance

MT6D is a unique defense due to its ability to change addresses during communication. This feature requires different scenarios to test under since it allows attackers to have varying amounts of time that the victim is known. Tests have been structured such that the percentage of time the victim IP is known to the attacker will vary. In addition to the destination being known, the origin being known alters the attack scenario. From this knowledge, there is a need for tests that include both scenarios where the origin IP address is known and unknown. Among the following results, MT6D results will have the format "X-Y", where X is whether the origin address is known and is either 0 (Not Known) or 1 (Known) and Y is the percentage of time that destination address is known (0% - 100%).

The first scenario analyzed is a connection-oriented transfer of 1MB with three active clients while under a SYN flood attack. The SYN flood attack was chosen because it causes the most direct traffic at the defense. Figures 7.3 and 7.4 depict the performance of the MT6D machine and the total transactions in the testbed respectively, show an the average result of

an accumulation of tests. Figures 7.5 and 7.6 take the previous data and performs a line of best fit on the curves along with applying the standard deviation. The standard deviation is shown to depict how closely the curves are acting alike.

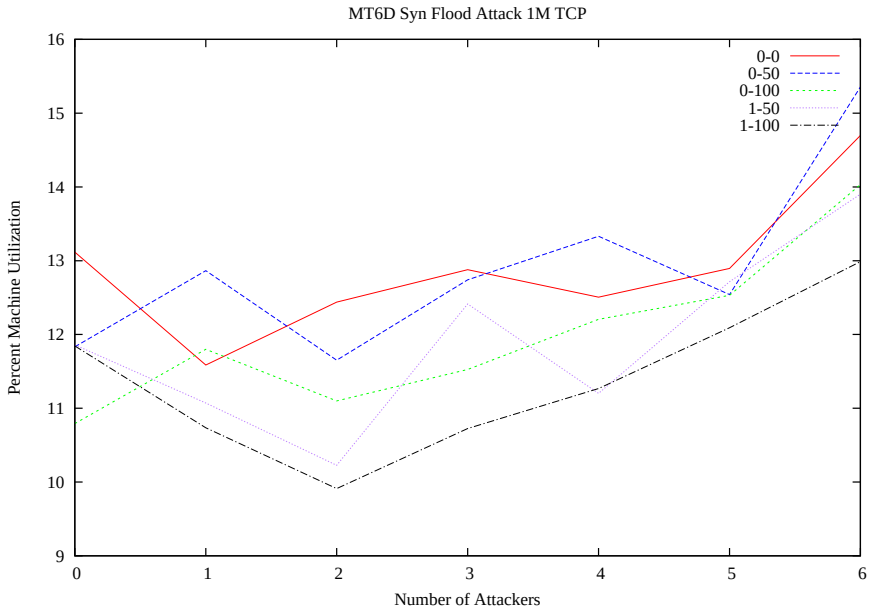

Figure 7.3: MT6D SYN Flood WGET 1M Performance

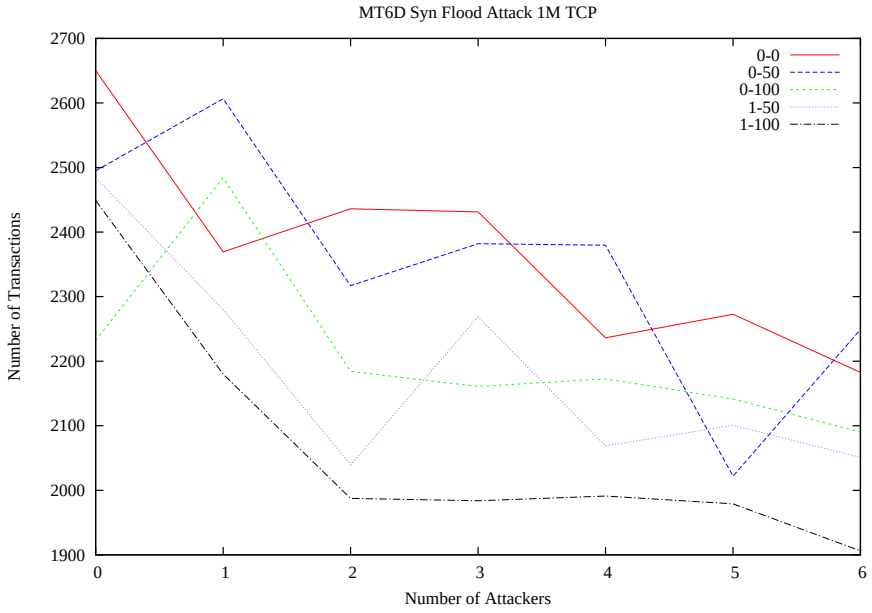

Figure 7.4: MT6D SYN Flood WGET 1M Transactions

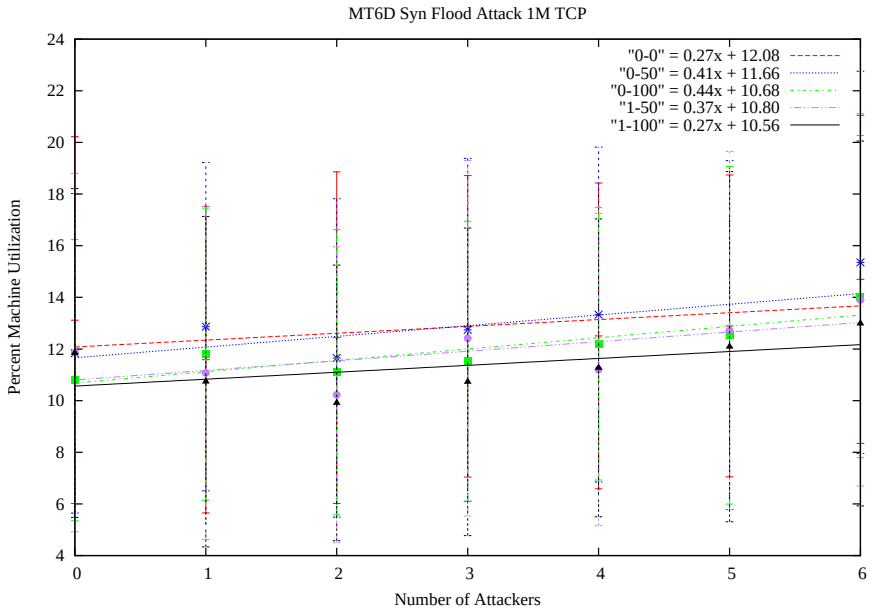

Figure 7.5: MT6D SYN Flood WGET 1M Performance With Best Fit and Error Bars

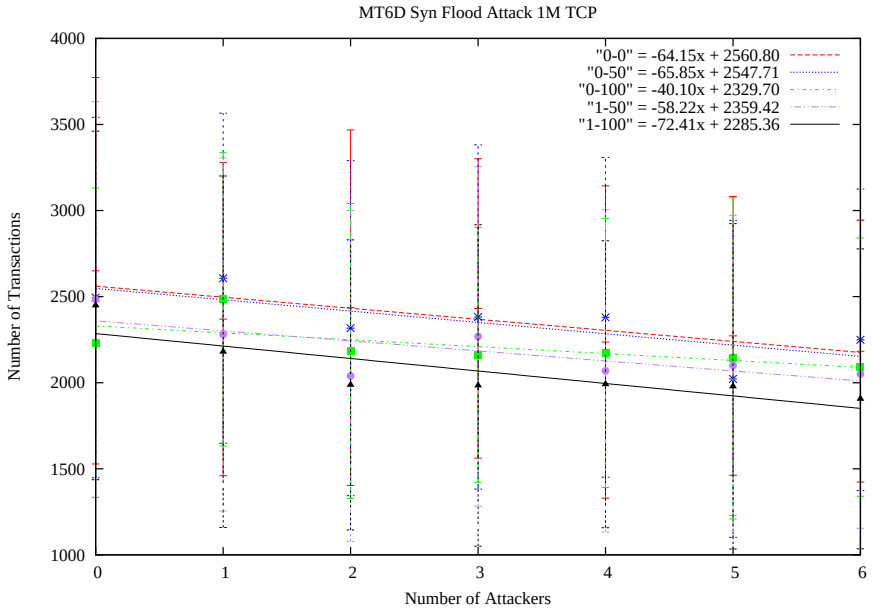

Figure 7.6: MT6D SYN Flood WGET 1M Transactions With Best Fit and Error Bars

First, it can be seen that the MT6D defense performs about the same regardless of the variables in the test. This happens because while MT6D is not directly attacked, the subnet is being flooded with Neighbor Solicitation (NS) messages. When the attacker is unable to determine the destination address, it continues to attack previous or random addresses on the subnet. This design decision was to allow the testbed to emulate attacking a previously-allocated address or an address the attacker believed to be the victim's. This flooding of NS messages does not significantly disrupt the machine but still must be processed to determine if it knows the requested destination. The defense must also forward these messages onto the private line to the server since the defense machine is not an endpoint but rather a pass-through. The message flood explains why, even when the defense is not able to be located by an attacker, its performance is still affected.

Understanding why the MT6D defense is affected while unable to be located is the one extreme; the other is when it is always known. If there is a failure to protect the system and the secret required to determine the address rotation progression is known to the attacker, it then is always able to find the victim machine. This then becomes a traditional situation of a DDoS defense thwarting off an attack. Again the attack is not able to greatly disrupt the defense. This time the slight increase in performance is reliant on MT6D packet encapsulation. An attacker analyzing network traffic, knowing the intended hosts, would see an IPv6 header followed by an encrypted payload. The attacker is unaware the header was provided on top of the header of the original packet. When an attacker's packet reaches the MT6D defense, it is first processed by the MT6D service. Failing authentication on

the destination or origin addresses would have the packet scrapped by the MT6D process. As the attacker knows this information, it will fail on the next step, attempting to retrieve the original packet. This process can be seen from the results to be fairly negligible on the system, but it does affect the overall testbed performance.

The results depict the two extreme situations as being affected similarly but by different methods, therefore the result comes from a common factor between the two situations. The packets received by the defense are different in the two scenarios with both legitimate and malicious intent. What occurs in both situations is that the network queue is being flooded with packets to process. This flood requires the system to allocate more resources to processing the queue, resulting in the unwanted behavior in the defense and testbed as a whole. The purpose of these results is to show that MT6D performs comparably under different attack situations. The processing of the network stack is outside the scope of this research and will affect every defense.

Further analysis in this paper will focus on the fully known MT6D system due to how closely the different tests act against an attack. This will show how MT6D acts against the attack and less upon how it acts when an attacker is attempting to locate it.

7.2 Slowloris

The first attack the defenses are compared against is Slowloris. Figure 7.7 shows the throughput achieved by the different defenses communicating 50MB transfers while under the Slowloris attack. The None line shows the attack with no defense to prevent it. It is obvious that with no defense to mitigate the attack, the None defense will succumb to the greedy attack, and no clients will be able to create a connection to the server, thus halting all legitimate traffic. The firewall defense shows a slight drop in throughput with no attackers due to its processing of the rules set to prevent attacks, but it still has a high throughput. Initially, the dip in throughput can be looked upon as acceptable considering the added protection they provide to the system. But due to the nature of the Slowloris attack, the firewall fails to protect the server from it. The firewall views the Slowloris attack as legitimate connections to the server, and thus allows them to pass through. The rate of effectiveness of the attack should have been partially alleviated by limiting the connections from an IP address, from which there would have been seen a tradeoff between NAT clients and total transactions. But since the attackers in the system could outnumber the server's available connections, the tradeoff would not be effective as the number of attackers could fully occupy the server. This rule becomes less effective in IPv6 when a single attacker is able to allocate several IP addresses, which the firewall would not be able to discern as coming from one machine. This describes how the firewall is extensible, although this analysis comes out of falling victim to an initial attack.

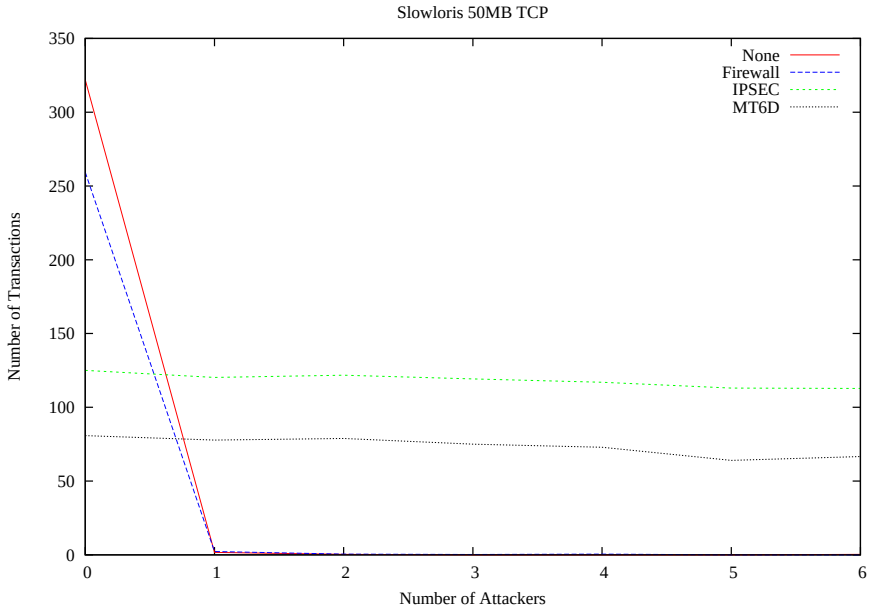

Figure 7.7: Slowloris WGET 50M Transactions

The IPsec and MT6D implementations show a substantial drop in throughput even without offsetting an attack. This comes from the overhead of the added benefits of the security systems. The Point-to-Point Tunneling Protocol (PPTP) encryption and packet encapsulation accounts for the performance drop seen from IPsec. IPsec is a very well developed technology and is unexpected to receive more substantial improvements. The same factors that affect IPsec also affect MT6D, with the added factor of rotating addresses. With clock drifting, allowing only the overlapping portion of the rotation period for communication; therefore all packets received during the non-overlapping portions between the two machines are discarded. MT6D is currently under development to improve its performance and variables such as the rotation time to address throughput, though it can be seen that it is close to the throughput of IPsec. While the throughput is lower than the firewall, these two defenses do maintain it while under the attack. The preconfigured clients allow for easier attack prevention, unlike the firewall which may not contain the available configuration to ever fully prevent this attack. The firewall is not guaranteed to prevent the attack due to increased IP addresses available to attackers.

The other viewpoint of the test, the defense performance, is depicted in Figure 7.8. As discussed in Section 4, Slowloris will actually cause less traffic amongst the testbed as it only sends the minimum required amount of data to keep the connection alive. The performance follows similar trends that we saw in the transaction graph. MT6D has a high performance impact to support the clients in the test as discussed in Section 7.1.1. The firewall follows the None trend with the additional rule processing overhead when there is only legitimate traffic as expected with the Slowloris attack.

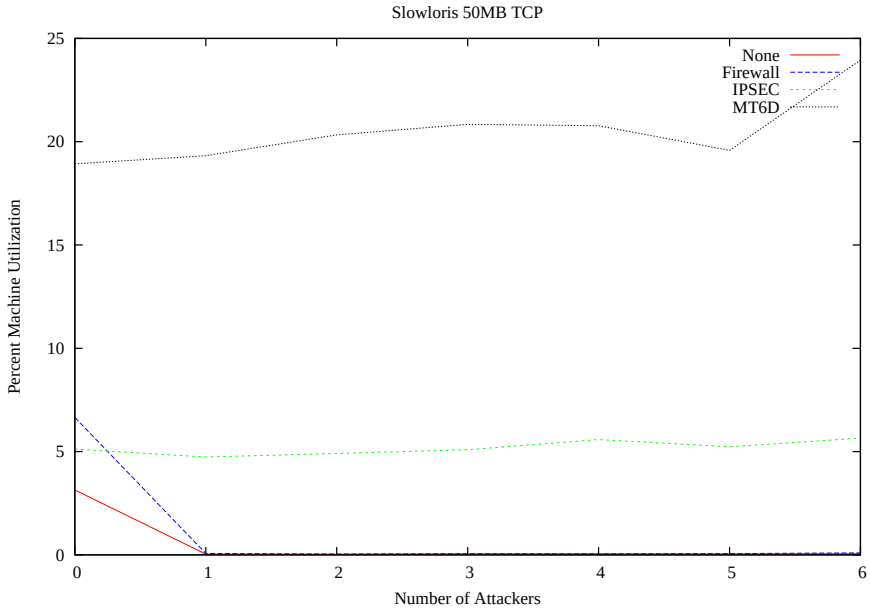

Figure 7.8: Slowloris WGET 50M Performance

Figure 7.9 depicts the Slowloris attack with UDP communication. It is easily seen that the application level attack does nothing to connectionless traffic as it operates completely separate from the HTTP server. The curves remain a flat trend to show that they are not affected by the attack; but again we can see the overhead of processing every packet into the defense with the firewall curve deviating from the other defenses' performance, reaffirming that pre-configured client defenses perform best under the conditions of the test.

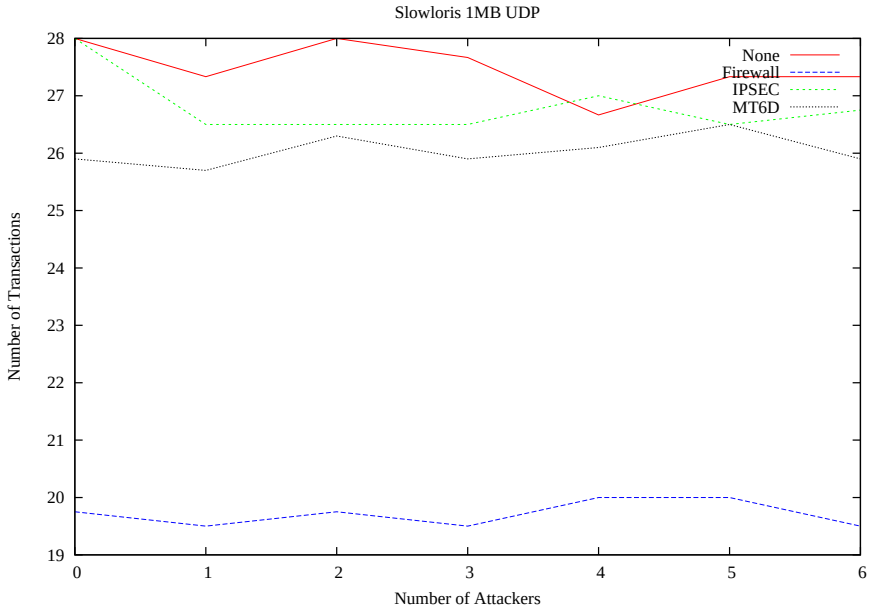

Figure 7.9: Slowloris iPerf 1M Transactions

7.3 SYN Flood

The next attack detailed is the SYN flood attack. Figure 7.10 shows the performance of the defense machines under the attack. The effects of the attack have already been discussed in regards to the MT6D in Section 7.1.2. The None line illustrates that when only legitimate traffic is passing through, more resources are required. This is because of the size of the packets: SYN packet are very light, whereas the file transfers are heavily-packed. The firewall line also follows the same trend, with the additional effect of rule analysis that has been seen in previous results. It can be predicted that the future analysis of the transactions will follow the trend of no defense from this performance trend. IPsec and MT6D show similar trends with a slight upward slope, determined to be the system requiring more resources to prevent the network queue from filling up.

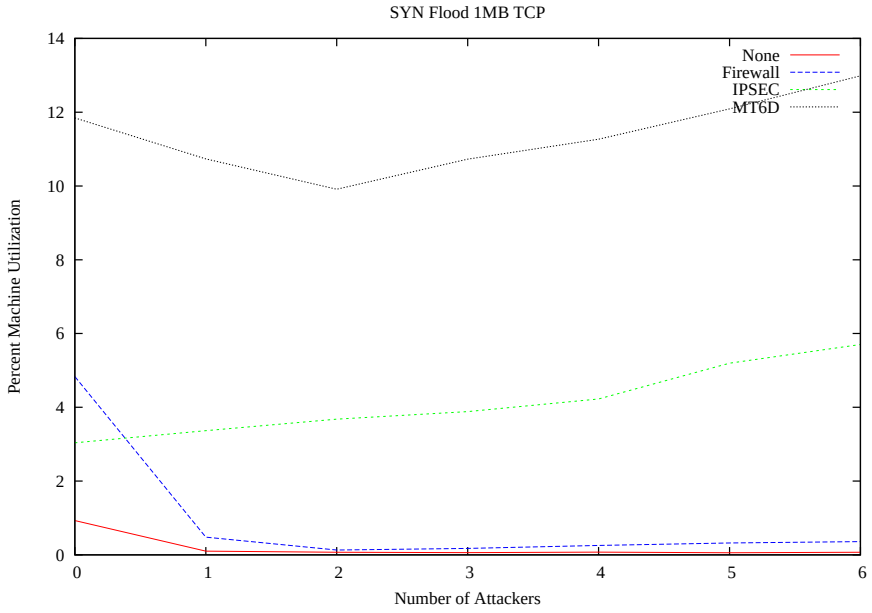

Figure 7.10: SYN Flood WGET 1M Performance

Figure 7.11 shows the transactions in the system under the same conditions. There is the same slightly downward trend seen on MT6D as before. As with the performance trend, IPsec follows MT6D with the slightly downward trend in overall transactions. IPsec is still able to get a greater number of completed transactions over MT6D. The None line shows that the server is not properly equipped to mitigate a SYN flood attack. The available sockets become occupied, and the clients are unable to connect to the server. The firewall also succumbs to the attack because the firewall is ill-equipped to handle the attack. With limiting the number of IPs and the amount of TCP connections, the attackers are still able to occupy the legitimate traffic the rules are set to allow. This is because the firewall is not knowledgeable enough to know where all these packets are coming from. Attackers' ability to spoof the originating address is unknown to the firewall, leaving the defense unable to discern and block the traffic. Furthermore, if the firewall were able to block the non-existent spoofed addresses, it would not have that ability against an attacker-spoofed legitimate addresses. It would then cut off communication between the firewall and that client. With the vast IP pool, the firewall needs additional means of determining the source of the packet in order to properly process it.

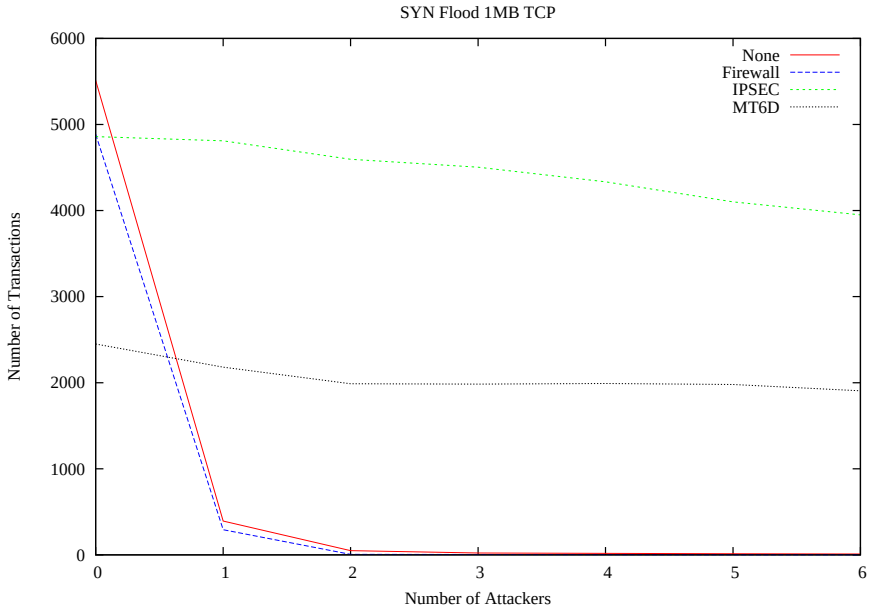

Figure 7.11: SYN Flood WGET 1M Transactions

Again, the attack is analyzed for its effect on connectionless traffic. Figure 7.12 shows the SYN flood attack with UDP traffic. As assumed, the connection-oriented attack does not affect the connectionless communication, nor does it affect the server enough to saturate all the resources available.

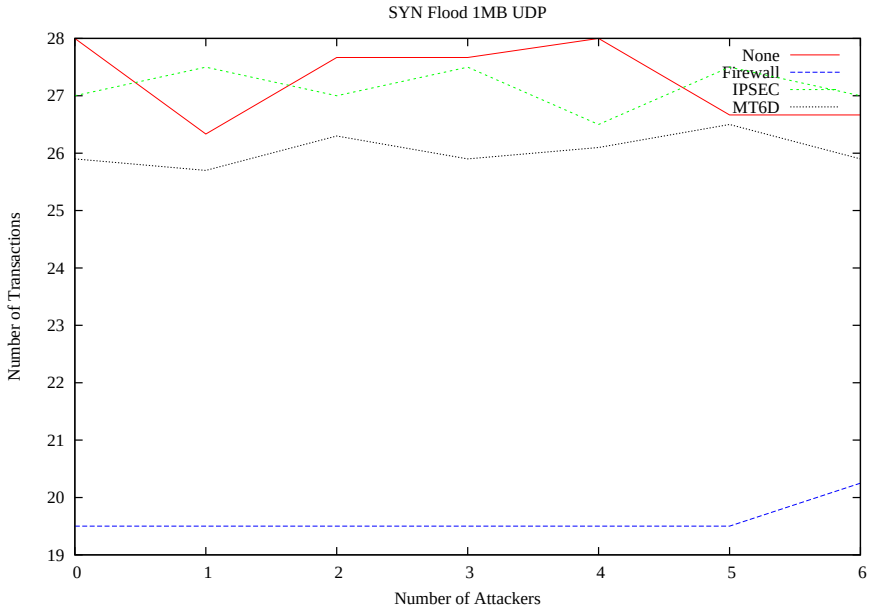

Figure 7.12: SYN Flood iPerf 1M Transactions

7.4 HTTP Get Flood

The HTTP Get Flood is presumed to cause the same type of damage as a SYN Flood attack because of the attack flooding the TCP connection stack. Then the attack additionally affects the application layer HTTP server performance. Figure 7.13 shows the transactions in the system during the attack. The graph depicts similar trends to the SYN Flood attack. This attack is able to allocate the server resources and cause a drastic drop in client throughput with no defense preventing it. The firewall shows a similar but less severe drop from the attack in the system. The firewall's rate-limited rules are being consumed by the attack, preventing the client from making the maximum amount of transfers, as seen in previous situations. The drop appears less drastic as compared to the SYN Flood results just by showing the higher transfer size. From the clients' standpoint, the large transfer size allows communication to remain unaffected for greater periods of time since only initiating the connection is affected by the attack. MT6D and IPsec are able to maintain a flat curve during this attack, showing that the flood had no noticeable effect on the system. This is logical as it prevents the attack at the network layer just as it blocks the SYN Flood attack.

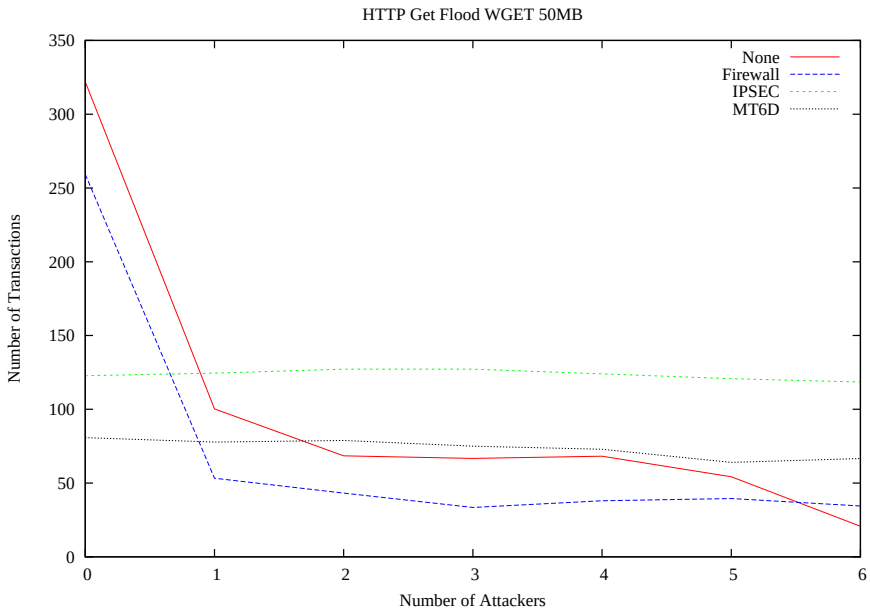

Figure 7.13: HTTP Get Flood 50MB Transactions

The performance of the HTTP Get Flood attack is much more revealing regarding its effect on the system. Figure 7.14 shows the performance of the server during the attack. The no defense curve shows the strain the attack puts on the system when it goes unmitigated. The server is left to process many junk requests and requires significantly more resources to do so. IPsec and MT6D did not show any affect from the transactions viewpoint and also show no affect here. The attack is almost completely dealt with at the defense machine. The firewall shows a trend similar to its client throughput because the defense prevents a single IP from flooding the server. The server is still processing the junk requests, but the rate at which it will receive those requests is much lower due to the firewall. This shows the application-side performance during the attack.

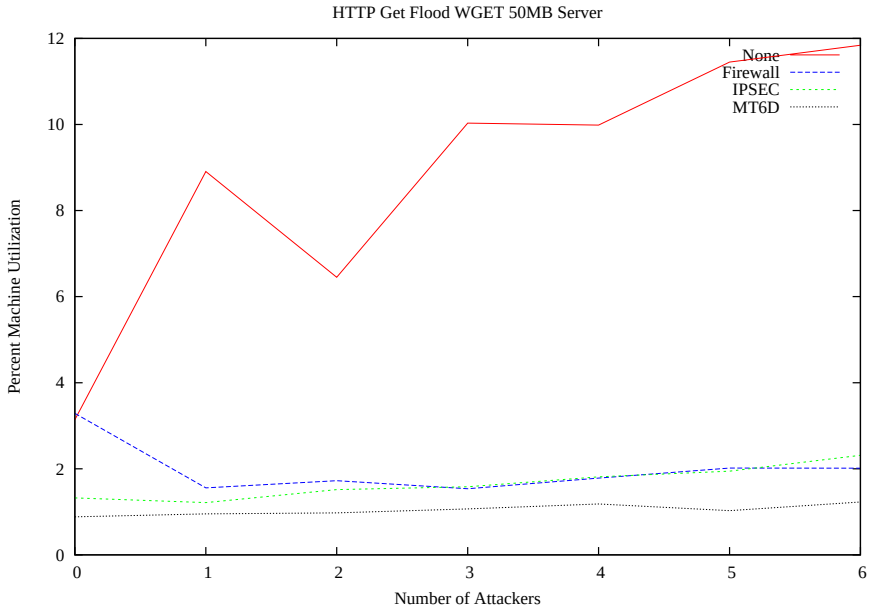

Figure 7.14: HTTP Get Flood 50MB Server Performance

Figure 7.15 shows the performance of the defense machines during the attack. The reoccurring trend among the firewall and no defense is seen here again. The legitimate traffic causes more strain than the illegitimate. MT6D shows a slightly upward trend, something seen from the other tests as the result of flooding, irregardless of the actual attack. The most significant result here is IPsec. It is forming a nearly linear trend upward. All other factors of the test align with the results of the SYN Flood attack. The differentiating factor of the HTTP Get Flood attack compared to the SYN Flood attack is the port associated with the packets. The HTTP Get Flood attack focuses on the same port which requires higher processing by the defense. The problem here is that the defense is not configured to drop the packets at that port. The defense would need to be changed to prevent traffic to the defense itself, which is causing this increase in utilization. This differs from MT6D because there is no machine address, meaning there is no way to talk to the defense without going through MT6D stack.

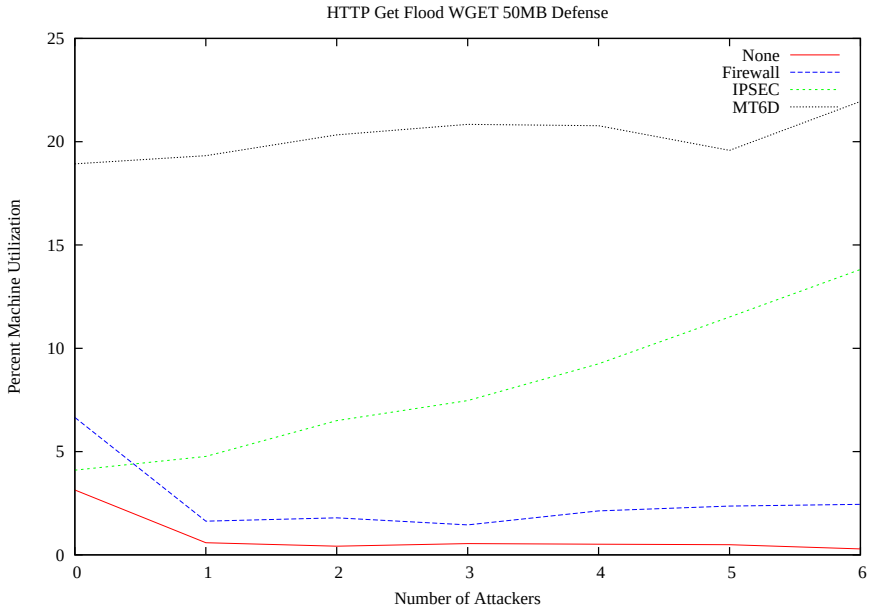

Figure 7.15: HTTP Get Flood 50MB Defense Performance

7.5 Denial6

Denial6 has two modes associated with the attack as described in Section 4. Analysis was kept separate to focus on the effects of the different exploitations. The defenses are not expected to act differently from the network setup because this attack effects the destination or hop-by-hop packet extensions. Packets across the network are directed at the server with no defense or the firewall. The packets are encapsulated in a packet directed at the defense machine with IPsec and MT6D. This makes the defense machine appear to be like the server for attackers under those defense conditions. Therefore the packet extension header exploitation will affect the defenses differently based on whether the actual server destination is known.

7.5.1 Destination

First, the destination option is analyzed. Figures 7.16 and 7.17 shows connection-oriented and connectionless transactions in the system during the attack. This is relatively uneventful, as the attack does not have any noticeable affect on the system. Even the test with no defense shows no difference in throughput and only shows deviation between tests. The effects are just being shifted to different resources, or the attack does not have a noticeable effect.

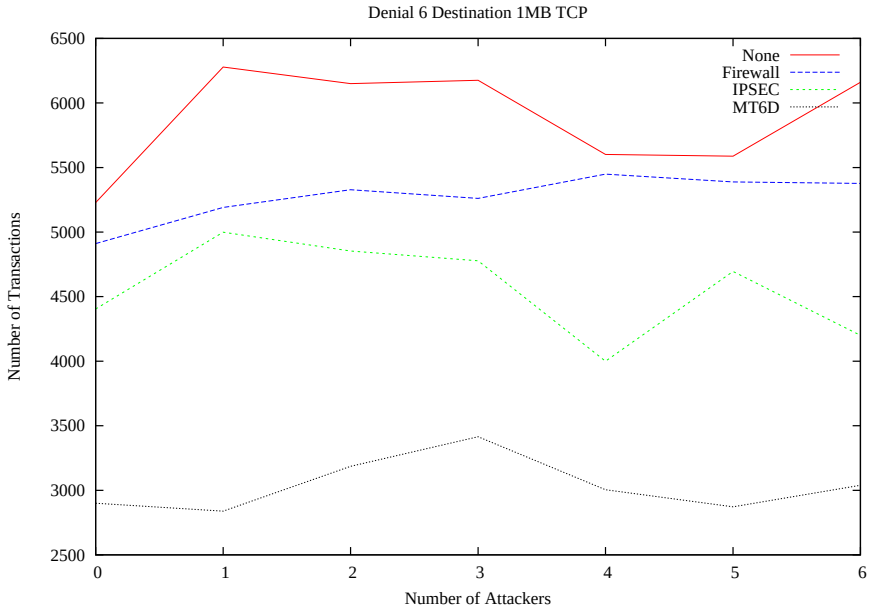

Figure 7.16: Denial6 Destination 1MB Transactions

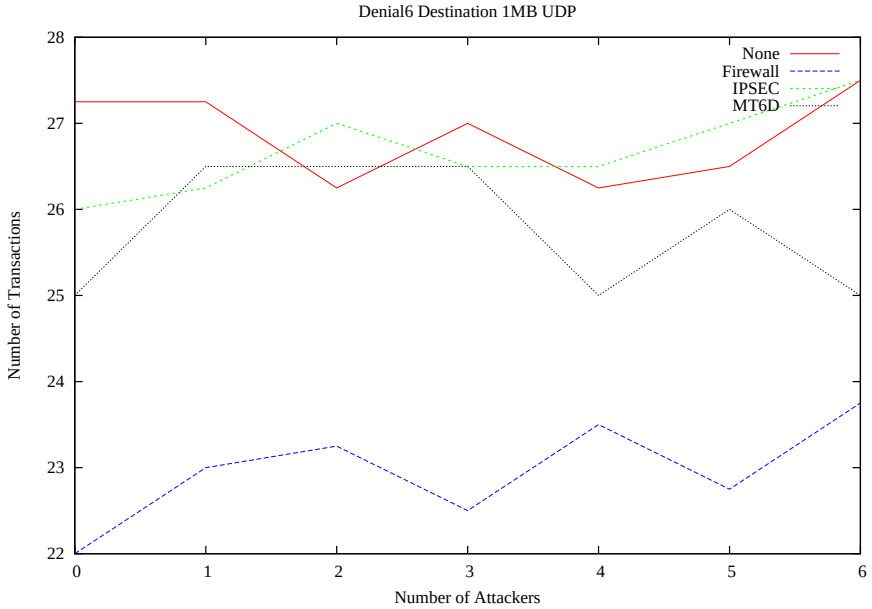

Figure 7.17: Denial6 Destination 1M UDP Transactions

Figures 7.18 and 7.19 show the performance of the defense and server machines respectively.

Figure 7.19 depicts the situation where the packet is processed for the extra options for firewall and no defense. The no defense curve shows the performance cost required to process all these extra options. The firewall shows a split between the two machines. The server is processing the packets that passed through the firewall, and the defense machine performance increase is from processing the packet flood. IPsec shows the best reaction to the attack: the server remains unaffected and the defense only shows a moderate upward trend seen in other flood attacks. This attack is where MT6D falls victim to an attack. MT6D becomes victimized by the attack because the operating system processes the extension options before passing that packet over to MT6D service. This has to do with MT6D operating at the guest level — MT6D would need to operate at a lower level to deter this effect. The attack is able to greatly increase the performance costs of the defense, although not enough to affect the actual client throughput at this scale.

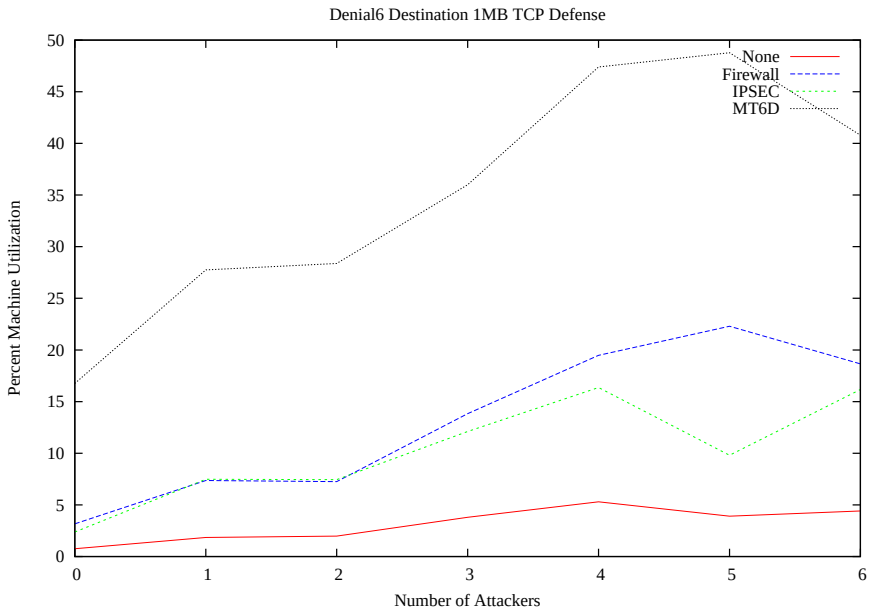

Figure 7.18: Denial6 Destination 1MB Defense Performance

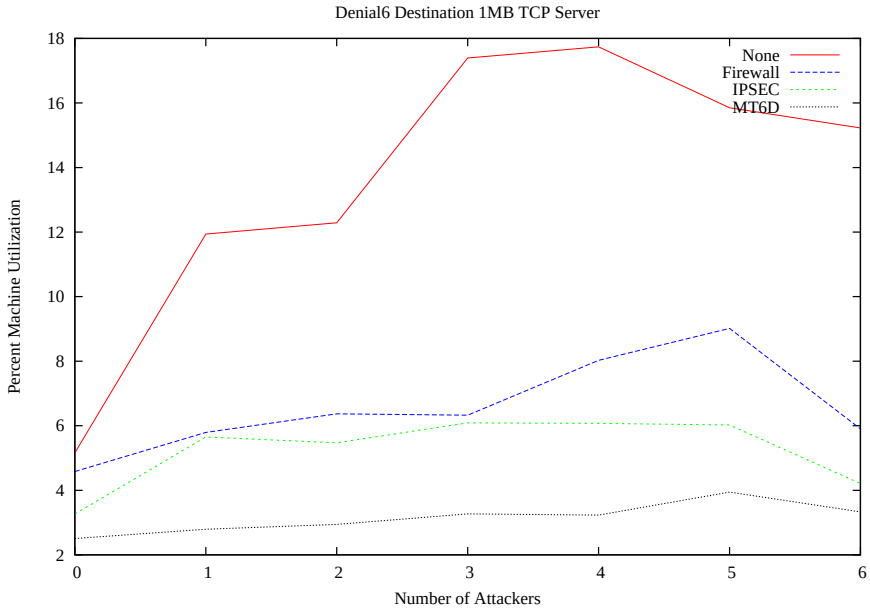

Figure 7.19: Denial6 Destination 1MB Server Performance

7.5.2 Hop-By-Hop

This attack is intended to occupy resources on the intermittent routers from the attacker to the victim. From Figure 7.20, the attack does not appear to have any meaningful effect on the system. The graph depicts the performance of the router before the defense, which all attackers pass through. There are negligible effects from the attack and the router shows not even one percent utilization as a result. The destination option processing occupied many resources on the victim machine(s) but this attack does not have the same effect to the victim routers. The attack did show that it was outputting packets and because the destination options and hop-by-hop options are just as extensible, the attack is not correctly implemented. This explains the non-existent performance costs. Fixing this attack to operate properly fell out of scope due to time constraints.

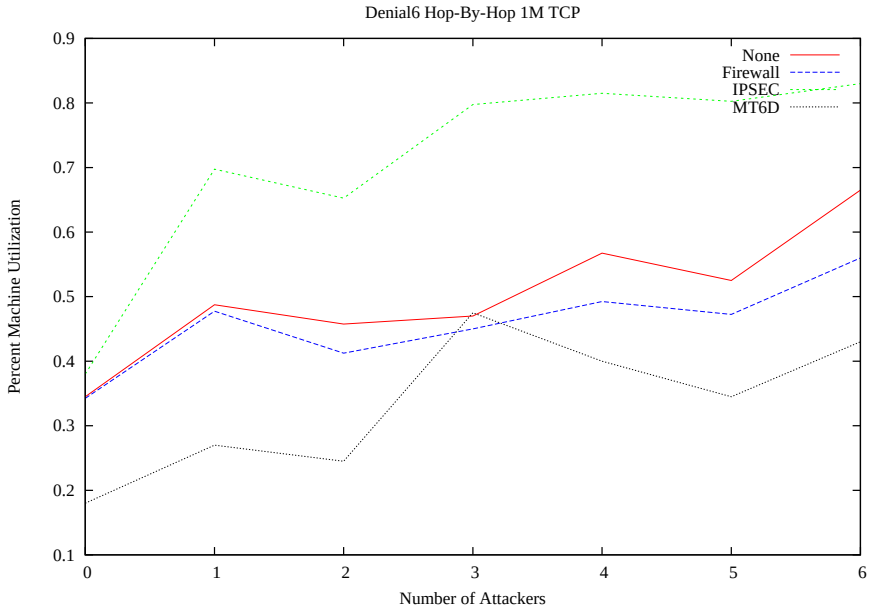

Figure 7.20: Denial6 Hop-By-Hop 1MB Router 1 Performance

7.6 Dos-New-IP6

The last attack analyzed in this research is the most effective. Dos-New-IP6 is able to completely halt communication to a victim subnet. This attack required additional time to execute effectively due to the NDP re-advertise time. But after the attack is allowed to execute while every machine in a subnet re-advertises themselves, it halts all traffic in that subnet. The defenses are not equipped to handle this sort of attack as it does not attempt to send illegitimate data or flood the victim. It merely sends legitimate, required data, but it does so in a malicious manner. From Figure 7.21, it can be seen that SLAAC allows the subnet to fall victim to this attack.

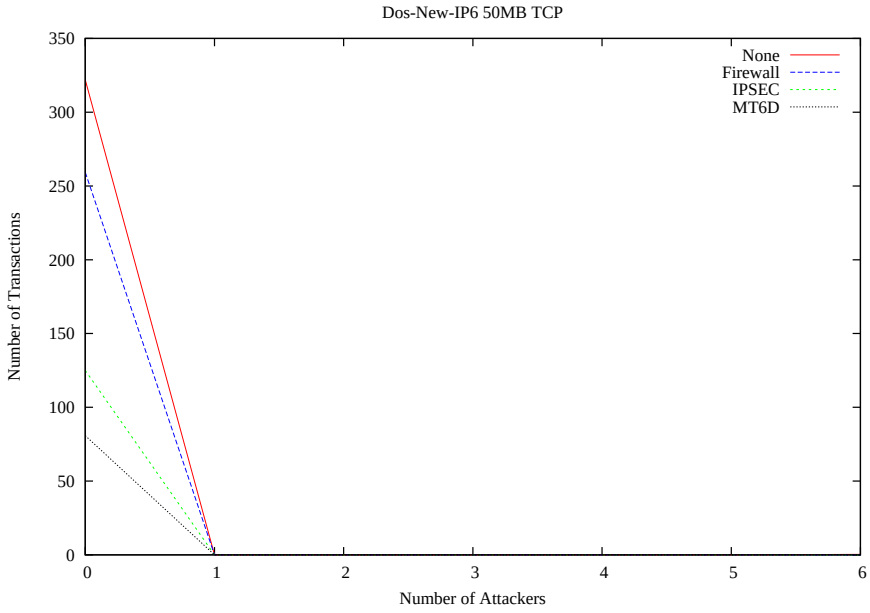

Figure 7.21: Dos-New-IP6 50MB Transactions

Chapter 8

Conclusions

There are many factors involved in determining the best DDoS defense. For this research, the focus is on protecting the targeted machines and maintaining high communication rates. There were no restrictions on network parameters besides being an IPv6 environment. As stated in Chapter 6.3, the best result is to maintain low performance and high throughput. This research hypothesized that MT6D would perform better than the competing defenses. From these tests, MT6D can be seen not providing the highest throughput nor maintaining the lowest overhead. The results even showed that the support to handle an additional client is very high for the added security benefits. The performance requirements of MT6D are considerably higher for the defense machine. These factors depict why MT6D would not be a suitable defense to use in this environment. But most importantly, MT6D does not succumb to the DDoS attacks it faces, with the exception of the option exploitation. The defense shows that it can mitigate an attack with only little additional effect on the system. And the option exploitation exploit can be closed off with additional development.

The firewall has shown that it is unable to efficiently deter attacks. IPv6 opens up a new plethora of problems for the firewall. Within IPv6, hackers are able to spoof a vast amount of IPs to allocate the firewall rules, which is easy to do with SLAAC or control of the IP allocation for the subnet. Without the ability to know that several attackers are merely one machine, the firewall rules are unable to prevent further attack without closing off the subnet. Analysis on the effects of closing off a subnet are outside the scope of this research.

IPsec does show the amount of work that has gone into it, and the fact that it is incorporated into the protocol allows it to operate very well under the attacks. It does, however, show its age through vulnerability under some of the test environments and requires some extra tools to account for these weaknesses. The firewall is a requirement for IPsec to deter effectively. It is the firewall which blocks the non-IPsec traffic. This complicates the setup and re-usability of the defense.

While the performance is quite high and communication rate low comparatively, MT6D

is very young and is still currently under development to improve performance. At this point in development, the difference in performance could be considered a reasonable loss for the added benefit by a network administrator. A test not examined (as it is outside the scope of the project) is the MITM attack gleaning information. IPsec uses static addresses for both the defense machine and the client machines. These static addresses leave the communication vulnerable to anyone who is located between the two machines. After enough time, communication could be compromised and different attacks (also outside the scope of this project) could be executed. MT6D deters this by further preventing the attacker from gleaming information.

The firewall cannot be ignored because of its performance due to the fact that IPsec and MT6D require preconfigured clients. This constraint is not applicable to all networked servers. A firewall is the only solution to attempt to mitigate an attack and leave valid communication when communication can't be to known clients and is instead open to all clients. From this work, though, it can be seen that additional effort will be required to formulate the firewall rules in an IPv6 environment.

Lastly, it has been shown that SLAAC is easily exploitable. This will lead to open and vital systems not being able to utilize this technology as it can easily be exploited. The requirement of trusting all machines within a network is something that will be decided by the network administrator.

Each network must have set list of requirements on determining the best DDoS defense. Based on what qualities are deemed valuable, it can be interpreted from these results that MT6D performs well and prevents DDoS attacks from eliminating valid communication while providing extra security of obscurity in the network.

Chapter 9

Future Work

With the ever evolving Internet Protocols, our tools need to be reevaluated to determine their effectiveness. The results provided in this research shows the current state of DDoS defenses' ability to mitigate attacks. But there is still additional work that should be done to provide more accurate state of the DDoS defenses.

This work has been kept at a small scale to better evaluate the affects of the attack. While the small scale allows better analysis of an attack on the total system, it can't provide a completely accurate picture of the current state of DDoS attacks. Defenses should be tested in a non-virtualized environment against a much higher scale of attackers to truly advocate one as more effective in attack mitigation. This will give way to a more natural setup of clients that enter and leave networks. A natural test environment also includes outside traffic not associated with the attack or the server-client interactions. This enables the test to see if the attack can disrupt unintended victims. Another means of extending these tests would be to utilize community testbeds such as DETER[4] which offers higher scalability.

Supplying content to the clients that is meaningful and not continuous static file size transfers, which is not natural client interactions will continue to provide a more accurate scenario. This realistic setting test allows attacks to behave against natural interactions, where the tests in this research were very static and continuous interactions. With the fixed sized file transfers, it can be seen if data is being transferred, but not if it is enough to disrupt the client.

An item that was originally planned to be addressed by this research was to test the link-local address. This includes clients and attackers communicating to the link-local address on the same subnet instead of its SLAAC assigned address. The premise behind this is to determine if the defenses are able to prevent malicious attacks when they're directed at this special IP address. Futher extending the tests in relation to SLAAC is to use Dynamic Host Configuration Protocol (DHCP) in placement of SLAAC. This will test how MT6D acts with the different address allocation method. The question to answer with this test extension is

whether MT6D can operate under DHCP and whether this will eliminate the packet flooding seen under SLAAC.

The new features introduced in IPv6 need to be tested in contained environments before they are exploited out in the public. Future work put forth in this field helps to prevent disruption among internet commerce and services.

Appendices

Please note that the code below was modified to fit within the page boundaries. The code may not be executable at this state with those changes made.

Appendix A

Master Scripts

These scripts depict what the master machine (not included in testbed figure) does to facilitate the testing among the testbed.

A.1 Master.sh

This script is to start up the tests in the testbed to a certain permutation of the test variables as stated in Section 6.

```
#!/bin/bash
#
# Run MT6D performance tests - Master Script
# This script is to run on Collector

#$1 is the defense type
readonly DEFENSE=$1
if [ "$DEFENSE" != "ipsec" ] && [ "$DEFENSE" != "firewall" ] &&
[ "$DEFENSE" != "mt6d" ] && [ "$DEFENSE" != "none" ]
then
echo "TEST TYPE ERROR"
exit
fi
readonly dos_new_ip6=$2
if [[ -z $dos_new_ip6 || $dos_new_ip6 != 0 && $dos_new_ip6 != 1 ]]
then
echo "Param 2 is 1 or 0, if this is strictly a dos-new-ip6 test"
exit
fi
```

```
#User settings
readonly USER=root
readonly PASSWORD=ibuypower
#Test time (2 minutes)
readonly TEST_TIME=120
readonly EXTRA_TIME=35
#SYN_FLOOD effects seem to linger after networking restart, need to wait
#through tcp timeout
readonly SYN_FLOOD_EXTRA_TIME=35
#Long time to avoid dos-new-ip6 lingering effects
readonly DOS_IP6_EXTRA_TIME=180
readonly SLEEP_TIME=$(( $TEST_TIME + $EXTRA_TIME ))
readonly SYN_FLOOD_SLEEP_TIME=$(( $TEST_TIME + $SYN_FLOOD_EXTRA_TIME ))
#Extra time before and after to allow affects to start/fade respectively
readonly DOS_IP6_SLEEP_TIME=$(( $DOS_IP6_EXTRA_TIME + $TEST_TIME +
$DOS_IP6_EXTRA_TIME ))
#The server who is being attacked/requested
readonly SERVER="dead:beef:0:1::300"
readonly DEFENSE_IP="dead:beef:0:1::301"
#MT6D Parameters
if [[ "$DEFENSE" == "mt6d" ]]
then
readonly MT6D_DEST=1
readonly MT6D_SOURCE=$3
if [[ -z $MT6D_SOURCE || $MT6D_SOURCE != 0 && $MT6D_SOURCE != 1 ]]
then
echo "MT6D_SOURCE_ERROR"
exit
fi
readonly MT6D_PERCENTAGE=$4
if [[ -z $MT6D_PERCENTAGE || $MT6D_PERCENTAGE -lt 0 ||
$MT6D_PERCENTAGE -gt 100 ]]
then
echo "MT6D_PERCENTAGE_ERROR"
exit
fi
else
readonly MT6D_DEST=0
readonly MT6D_SOURCE=0
readonly MT6D_PERCENTAGE=0
fi
```

```
#The machines that need to have performance evaluated
readonly EVALUATED_MACHINES_NAMES=( "server" "defense-$DEFENSE" "router1"
"router2" "router3" "router4" "router5" )
readonly EVALUATED_MACHINES_IPS=( "dead:beef:0:1::300" "dead:beef:0:1::301"
"dead:beef:0:1::1" "dead:beef:0:2::1" "dead:beef:0:3::1" "dead:beef:0:4::1"
"dead:beef:0:5::1" )
#Attackers/Clients
#Attackers start on 200 range
readonly SUB_ATTACKERS=( 1 0 1 1 3 )
#Clients start on 100 range
if [ "$DEFENSE" == "mt6d" ] || [ "$DEFENSE" == "ipsec" ]
then
#MT6D/IPSEC Clients (Subnet 5(NAT) doesn't apply)
readonly SUB_CLIENTS=( 1 0 1 1 0 )
else
#readonly SUB_CLIENTS=( 1 0 1 1 1 )
#Not using 4 client results for anything, don't run those tests
readonly SUB_CLIENTS=( 1 0 1 1 0 )
fi
#Attacks, the only attack for MT6D is syn_flood
#if [[ $dos_new_ip6 -eq 1 ]]
#then
#       readonly ATTACKS=( "dos-new-ip6" )
#elif [ "$DEFENSE" == "mt6d" ]
#then
#       readonly ATTACKS=( "syn_flood" )
#else
#       readonly ATTACKS=( "slowloris" "denial6_1" "denial6_2" "http_get_floo
"syn_flood" )
#       readonly ATTACKS=( "denial6_1" "denial6_2" "http_get_flood" )
#fi
#Client Requests
#readonly REQUESTS=( "wget_1m" "wget_50m" "wget_500m" "iperf_500k_udp"
"iperf_1m_udp" )
readonly REQUESTS=( "wget_1m" )
readonly ATTACKS=( "denial6_2" )

main () {
#Check inputs in case of accident
if [ ${#EVALUATED_MACHINES_NAMES[@]} -ne ${#EVALUATED_MACHINES_IPS[@]} ]
then
"Evaluted machines Input Size don't match"
```

```
exit
fi
if [ ${#SUB_ATTACKERS[@]} -ne ${#SUB_CLIENTS[@]} ]
then
"Subnet Size don't match"
exit
fi

#Pretest calculations
NUM_ATTACKERS=0
NUM_CLIENTS=0
for ((i=0; i < ${#SUB_ATTACKERS[@]}; i++)); do
NUM_ATTACKERS=$(( $NUM_ATTACKERS + ${SUB_ATTACKERS[$i]} ))
NUM_CLIENTS=$(( $NUM_CLIENTS + ${SUB_CLIENTS[$i]} ))
done
#Add 1 to account for boundaries
NUM_ATTACKERS=$(( $NUM_ATTACKERS + 1 ))
NUM_CLIENTS=$(( $NUM_CLIENTS + 1 ))

#Determine Test End time
syn_flood_included=0
for attack in "${ATTACKS[@]}"; do
if [ "$attack" == "syn_flood" ]
then
syn_flood_included=1
fi
done

if [[ $dos_new_ip6 -eq 1 ]]
then
run_time=$(( $DOS_IP6_SLEEP_TIME * $NUM_ATTACKERS * $NUM_CLIENTS *
${#ATTACKS[@]} * ${#REQUESTS[@]} ))
elif [ $syn_flood_included -eq 0 ]
then
run_time=$(( $SLEEP_TIME * $NUM_ATTACKERS * $NUM_CLIENTS *
${#ATTACKS[@]} * ${#REQUESTS[@]} ))
else
#Account for fact that SYN_FLOOD needs more time between tests
num_attacks=$(( ${#ATTACKS[@]} - 1 ))
syn_flood_run_time=$(( $SYN_FLOOD_SLEEP_TIME * $NUM_ATTACKERS *
$NUM_CLIENTS * ${#REQUESTS[@]} ))
other_run_time=$(( $SLEEP_TIME * $NUM_ATTACKERS * $NUM_CLIENTS *
```

```
$num_attacks * ${#REQUESTS[@]} ))
run_time=$(( $syn_flood_run_time + $other_run_time ))
fi
curr_time=`date +"%s"`
end_time=$(( $curr_time + $run_time ))
seconds=$(( $run_time % 60 ))
minutes=$(( $run_time % 3600 / 60 ))
hours=$(( $run_time / 3600 ))
eval echo -e "Test began at `date`"
eval echo -e "Test will run for ${hours}h:${minutes}m:${seconds}s"
eval echo -e "Test will end at `date -d @$end_time`"
echo -e ""

#Start tests
#Loop 1 is on attacks
for attack in "${ATTACKS[@]}"; do
#Loop 2 is on client interaction type
for request in "${REQUESTS[@]}"; do
#Loop 3 is on number of attackers
for ((num_attackers=0; num_attackers < $NUM_ATTACKERS;
num_attackers++)); do
#Loop 4 is on number of clients
for ((num_clients=0; num_clients < $NUM_CLIENTS;
num_clients++)); do

#TESTING BEGIN
curr_time=`date +"%s"`
echo -e "$curr_time\tTest Start: $attack\t$num_attackers
\t$request\t$num_clients"
#TESTING END

#Start up performance tracking
setup_performance_tracking $attack $num_attackers
$request $num_clients

#Start up attackers, starts at first subnet and
#increments through attackers/subnets
if [ $num_attackers -ne 0 ]
then
active_attackers=0
index=0
offset=0
```

```
attacker_count=${SUB_ATTACKERS[0]}
while [ $active_attackers -lt $num_attackers ]; do
if [ $attacker_count -gt 0 ]
then
attacker_count=$(( $attacker_count - 1 ))
setup_machine "attacker" $index $offset
$attack $num_attackers $request $num_clients
active_attackers=$(( $active_attackers + 1))
offset=$(( $offset + 1 ))
else
index=$(( $index + 1 ))
attacker_count=${SUB_ATTACKERS[$index]}
offset=0
fi
done
fi

#Start up clients, starts at first subnet and increments
#through clients/subnets
if [ $num_clients -ne 0 ]
then
active_clients=0
index=0
offset=0
client_count=${SUB_CLIENTS[0]}
while [ $active_clients -lt $num_clients ]; do
if [ $client_count -gt 0 ]
then
client_count=$(( $client_count - 1 ))
setup_machine "client" $index $offset
$attack $num_attackers $request $num_clients
active_clients=$(( $active_clients + 1 ))
offset=$(( $offset + 1 ))
else
index=$(( $index + 1 ))
client_count=${SUB_CLIENTS[$index]}
offset=0
fi
done
fi

#TESTING BEGIN
```

```
curr_time=`date +"%s"`
echo -e "$curr_time\tMaster Sleeping: $attack\t
$num_attackers\t$request\t$num_clients"
#TESTING END

#Sleep this for test time to wait for next test. Extra
#time for good measure
if [ "$attack" == "syn_flood" ]
then
sleep $SYN_FLOOD_SLEEP_TIME
elif [ "$attack" == "dos-new-ip6" ]
then
sleep $DOS_IP6_SLEEP_TIME
else
sleep $SLEEP_TIME
fi
#Reset networking, to counteract attack measures
reset_networking

#TESTING BEGIN
curr_time=`date +"%s"`
echo -e "$curr_time\tTest Complete: $attack\t
$num_attackers\t$request\t$num_clients\n"
#TESTING END

done
done
done
done

eval echo -e "Test end at `date`"
if [[ "$DEFENSE" == "mt6d" ]]
then
eval echo "$DEFENSE $dos_new_ip6 $MT6D_SOURCE $MT6D_PERCENTAGE" |
sendemail -f collector@mail.com -t peter.l.dimarco@gmail.com -u
"Tests Done" -s smtp.vt.edu
else
eval echo "$DEFENSE $dos_new_ip6" | sendemail -f collector@mail.com
-t peter.l.dimarco@gmail.com -u "Tests Done" -s smtp.vt.edu
fi
}
```

```
reset_networking () {
#Resets the machines to counteract any lingering attack effects
curr_time=`date +"%s"`
echo -e "$curr_time\tReseting Networking on all machines"

#Reset Routers/Defense/Server
for (( i=0; i < ${#EVALUATED_MACHINES_IPS[@]}; i++ )); do
if [ ${EVALUATED_MACHINES_NAMES[$i]} == "server" ]
then
#Kill Apache/iPerf, restart the services
sshpass -p $PASSWORD ssh -o StrictHostKeychecking=no
$USER@$SERVER 'killall apache iperf &> /dev/null'
sshpass -p $PASSWORD ssh -o StrictHostKeychecking=no
$USER@$SERVER 'apachectl -k start &> /dev/null'
sshpass -p $PASSWORD ssh -o StrictHostKeychecking=no
$USER@$SERVER 'nohup iperf -Vsu &> /dev/null &'
fi
eval sshpass -p $PASSWORD ssh -o UserKnownHostsFile=/dev/null -o
StrictHostKeychecking=no $USER@${EVALUATED_MACHINES_IPS[$i]}
'service networking restart &> /dev/null; ping6 -c 1 $SERVER &>
/dev/null &'
done
#Reset Attackers
for (( i=0; i < ${#SUB_ATTACKERS[@]}; i++ )); do
local num_attackers=${SUB_ATTACKERS[$i]}
for (( j=0; j < $num_attackers; j++ )); do
local subnet=$(( $i + 1 ))
local offset=$(( $j + 200 ))
eval sshpass -p $PASSWORD ssh -o StrictHostKeychecking=no
$USER@dead:beef:0:$subnet::$offset 'service networking restart
&> /dev/null; ping6 -c 1 $SERVER &> /dev/null &'
done
done
#Reset Clients
for (( i=0; i < ${#SUB_CLIENTS[@]}; i++ )); do
local num_clients=${SUB_CLIENTS[$i]}
for (( j=0; j < $num_clients; j++ )); do
local subnet=$(( $i + 1 ))
local offset=$(( $j + 100 ))
eval sshpass -p $PASSWORD ssh -o StrictHostKeychecking=no
$USER@dead:beef:0:$subnet::$offset 'service networking restart
&> /dev/null; ping6 -c 1 $SERVER &> /dev/null &'
```

```
done
done
}

setup_machine () {
#Starts up the specified attack on the attacker
#1 - attacker or client
#2 - subnet index (important to note it is the index)
#3 - attacker offset
#4 - type of attack
#5 - number of attackers
#6 - type of request
#7 - number of clients
local offset=0
local subnet=0
subnet=$(( $2 + 1 ))
if [ $1 == "client" ]
then
offset=$(( $3 + 100 ))
else
offset=$(( $3 + 200 ))
fi

#TESTING BEGIN
curr_time=`date +"%s"`
echo -e "$curr_time\t$1: $subnet\t$offset\t$4\t$5\t$6\t$7"
#TESTING END

if [[ $offset -gt 199 && "$DEFENSE" == "ipsec" ]]
then
#Attack the ipsec machine instead of server, anyone watching traffic would
#see that address
eval sshpass -p $PASSWORD ssh -o StrictHostKeychecking=no
$USER@dead:beef:0:$subnet::$offset 'nohup /home/pdimarco/$1.sh $4 $5
$6 $7 $DEFENSE_IP $TEST_TIME $MT6D_DEST $MT6D_SOURCE
$MT6D_PERCENTAGE &> /dev/null &'
else
eval sshpass -p $PASSWORD ssh -o StrictHostKeychecking=no
$USER@dead:beef:0:$subnet::$offset 'nohup /home/pdimarco/$1.sh $4 $5
$6 $7 $SERVER $TEST_TIME $MT6D_DEST $MT6D_SOURCE $MT6D_PERCENTAGE
&> /dev/null &'
fi
```

```
}

setup_performance_tracking () {
#Starts up the recorder on the EVALUATED_MACHINES
#1 - type of attack
#2 - number of attackers
#3 - type of request
#4 - number of clients

for (( i=0; i < ${#EVALUATED_MACHINES_NAMES[@]}; i++ )); do

#TESTING BEGIN
curr_time=`date +"%s"`
echo -e "$curr_time\tPerf: ${EVALUATED_MACHINES_NAMES[$i]}\t
${EVALUATED_MACHINES_IPS[$i]}"
#TESTING END

#Changed from using perf to mpstat
eval sshpass -p $PASSWORD ssh -o UserKnownHostsFile=/dev/null -o
StrictHostKeychecking=no $USER@${EVALUATED_MACHINES_IPS[$i]} "nohup
mpstat 1 $TEST_TIME \> /home/pdimarco/
${EVALUATED_MACHINES_NAMES[$i]}[]$1[]$2[]$3[]$4.txt &" 2> /dev/null

done
}

#Execute main routine
main
```

A.2 Collector.sh

This script is to collect all the results from the machines.

```
#!/bin/bash
#
# Run MT6D performance tests - Collector Script
# This script is to run on Collector to gather all files created
#during tests.

#$1 is the defense type
readonly DEFENSE=$1
```

```
if [ "$DEFENSE" != "ipsec" ] && [ "$DEFENSE" != "firewall" ] &&
[ "$DEFENSE" != "mt6d" ] && [ "$DEFENSE" != "none" ]
then
echo "TEST_TYPE_ERROR"
exit
fi
#User settings
readonly USER=root
readonly PASSWORD=ibuypower
#The server who is being attacked/requested
readonly DEFENSE_IP="dead:beef:0:1::301"
#The machines that need to have performance evaluated
#NEEDS TO BE CHANGED PER DEFENSE/SERVER TYPE
readonly EVALUATED_MACHINES_NAMES=( "router1" "router2" "router3" "router4"
"router5" "server" "defense-$DEFENSE" )
readonly EVALUATED_MACHINES_IPS=( "dead:beef:0:1::1" "dead:beef:0:2::1"
"dead:beef:0:3::1" "dead:beef:0:4::1" "dead:beef:0:5::1"
"dead:beef:0:1::300" "dead:beef:0:1::301" )
#Attackers/Clients
#Attackers start on 200 range
readonly SUB_ATTACKERS=( 1 0 1 1 3 )
#Clients start on 100 range
if [ "$DEFENSE" == "mt6d" ] || [ "$DEFENSE" == "ipsec" ]
then
#MT6D/IPSEC Clients (Subnet 5(NAT) doesn't apply)
readonly SUB_CLIENTS=( 1 0 1 1 0 )
else
readonly SUB_CLIENTS=( 1 0 1 1 1 )
fi

main () {
#Check inputs incase of accident
if [ ${#EVALUATED_MACHINES_NAMES[@]} -ne ${#EVALUATED_MACHINES_IPS[@]} ]
then
"Evaluted_machines_Input_Size_don't_match"
exit
fi
if [ ${#SUB_ATTACKERS[@]} -ne ${#SUB_CLIENTS[@]} ]
then
"Subnet_Size_don't_match"
exit
fi
```

```
#Pretest calculations
NUM_ATTACKERS=0
NUM_CLIENTS=0
for ((i=0; i < ${#SUB_ATTACKERS[@]}; i++)); do
NUM_ATTACKERS=$(( $NUM_ATTACKERS + ${SUB_ATTACKERS[$i]} ))
NUM_CLIENTS=$(( $NUM_CLIENTS + ${SUB_CLIENTS[$i]} ))
done

#Clear previous data
#rm -r /home/pdimarco/${EVALUATED_MACHINES_NAMES[6]}
mkdir /home/pdimarco/${EVALUATED_MACHINES_NAMES[6]}

#Gather performance data
gather_performance_tracking

#Gather Attack Data
active_attackers=0
index=0
offset=0
attacker_count=${SUB_ATTACKERS[0]}
while [ $active_attackers -lt $NUM_ATTACKERS ]; do
if [ $attacker_count -gt 0 ]
then
attacker_count=$(( $attacker_count - 1 ))
gather_machine "attacker" $index $offset
active_attackers=$(( $active_attackers + 1 ))
offset=$(( $offset + 1 ))
else
index=$(( $index + 1 ))
attacker_count=${SUB_ATTACKERS[$index]}
offset=0
fi
done

#Gather Client Data
active_clients=0
index=0
offset=0
client_count=${SUB_CLIENTS[0]}
while [ $active_clients -lt $NUM_CLIENTS ]; do
if [ $client_count -gt 0 ]
```

```
then
client_count=$(( $client_count - 1 ))
gather_machine "client" $index $offset
active_clients=$(( $active_clients + 1 ))
offset=$(( $offset + 1 ))
else
index=$(( $index + 1 ))
client_count=${SUB_CLIENTS[$index]}
offset=0
fi
done
}

gather_machine () {
#Gathers up the specified attacker/client data
#1 - attacker or client
#2 - subnet index (important to note it is the index)
#3 - attacker/client offset

local number=0
local offset=0
local subnet=0
subnet=$(( $2 + 1 ))
number=$(( $3 + 1 ))
if [ $1 == "client" ]
then
offset=$(( $3 + 100 ))
else
offset=$(( $3 + 200 ))
fi

echo -e "Gathering $1: $subnet\t$offset"
mkdir /home/pdimarco/${EVALUATED_MACHINES_NAMES[6]}/$1$subnet-$number
eval sshpass -p $PASSWORD scp -o StrictHostKeychecking=no
$USER@[dead:beef:0:$subnet::$offset]:/home/pdimarco/$1$subnet-$number*.txt
/home/pdimarco/${EVALUATED_MACHINES_NAMES[6]}/$1$subnet-$number/
eval sshpass -p $PASSWORD ssh -o StrictHostKeychecking=no
$USER@dead:beef:0:$subnet::$offset
'nohup rm /home/pdimarco/$1$subnet-$number*.txt &> /dev/null &'

}
```

```
gather_performance_tracking () {
#Gathers up the perf recorder data on the EVALUATED_MACHINES

for (( i=0; i < ${#EVALUATED_MACHINES_NAMES[@]}; i++ )); do
#Gather Data from machine
echo -e "Gathering Performance: ${EVALUATED_MACHINES_NAMES[$i]}\t
${EVALUATED_MACHINES_IPS[$i]}"
mkdir /home/pdimarco/${EVALUATED_MACHINES_NAMES[6]}/
${EVALUATED_MACHINES_NAMES[$i]}
eval sshpass -p $PASSWORD scp -o UserKnownHostsFile=/dev/null -o
StrictHostKeychecking=no $USER@[${EVALUATED_MACHINES_IPS[$i]}]:
/home/pdimarco/*\[\]* /home/pdimarco/${EVALUATED_MACHINES_NAMES[6]}/
${EVALUATED_MACHINES_NAMES[$i]}/ 2> /dev/null
eval sshpass -p $PASSWORD ssh -o UserKnownHostsFile=/dev/null -o
StrictHostKeychecking=no $USER@${EVALUATED_MACHINES_IPS[$i]} 'nohup rm
/home/pdimarco/*[]* &> /dev/null &' 2> /dev/null
#Changed from PERF to MPSTAT
# Parse Data files into readable text
# echo -e "Converting to Text: ${EVALUATED_MACHINES_NAMES[$i]}\t
#${EVALUATED_MACHINES_IPS[$i]}"
# filelist=`ls -d /home/pdimarco/${EVALUATED_MACHINES_NAMES[6]}/
#${EVALUATED_MACHINES_NAMES[$i]}/*.data`
# for file in $filelist; do
# if [ -s $file ] #is the file >0 bytes, have seen some 0 byte data files
#(only from server)
# then
# new_file=$(echo $file | sed "s/.data/.txt/g")
# eval perf_3.2.0-29 report --sort comm,dso -i $file > $new_file 2>
#/dev/null
# rm $file
# fi
# done
done
}

#Execute main routine
main
```

Appendix B

Attacker Scripts

B.1 Attacker.sh

This script allows the master to call the attacker with parameters on what attack to start up.

```
#!/bin/bash
#
# Run MT6D performance tests - Attacker Script
# This script is to run attacker work

#1 - type of attack
#2 - number of attackers
#3 - type of request
#4 - number of clients
#5 - server
#6 - test_time

#Running Task PID
PID=0
#MT6D Parameters, Keys must be client#-#
readonly MT6D_KEYS=( "client1-1" "client3-1" "client4-1" )
readonly ROT_BITS=1
readonly MT6D_ROT_TIME=$(( 2**$ROT_BITS ))
#Sleep time between task checks
readonly SLEEP_TIME=0.001
#Attacker variables
readonly DOS_IP6_EXTRA_TIME=180
readonly ATTACK=$1
```

```
readonly NUM_ATTACKERS=$2
readonly REQUEST=$3
readonly NUM_CLIENTS=$4
#Removed readonly from Server to allow mt6d to adjust attacker
INPUT_SERVER=$5
SERVER=$5
PORT=80
SOURCE=`ifconfig eth0 | awk '{print $3}' | grep dead:beef |
sed 's/\(.*\)...\/\1/'`
TEST_TIME=$6
readonly MT6D_DEST=$7
readonly MT6D_SOURCE=$8
readonly MT6D_PERCENTAGE=$9

main () {
#Run dos-new-ip6 for 3 minutes before test to have effects work
#Attack will report running for 3 minutes longer than master stated
if [ "$ATTACK" == "dos-new-ip6" ]
then
TEST_TIME=$(( $TEST_TIME + $DOS_IP6_EXTRA_TIME ))
fi
#Find finishing time
curr_time=`date +"%s"`
end_time=$(( $curr_time + $TEST_TIME ))

#Record starting time
echo -e "STARTED\t$curr_time\t$end_time" >> /home/pdimarco/`hostname`[]
${ATTACK}[]${NUM_ATTACKERS}[]${REQUEST}[]${NUM_CLIENTS}.txt

#Do work for specified time
while [ $curr_time -lt $end_time ]; do
#Calculate MT6D if needed
if [[ $MT6D_DEST -ne 0 ]]
then
number=$(( $RANDOM % 100 ))
if [[ $number -lt $MT6D_PERCENTAGE && $NUM_CLIENTS -ne 0 ]]
then
number=$(( $number % $NUM_CLIENTS ))
key=${MT6D_KEYS[$number]}
SERVER=`/home/pdimarco/mt6d_ip_tool -A $INPUT_SERVER
$ROT_BITS $key`
SERVER="${SERVER%\\n}"
```

```
PORT=`/home/pdimarco/mt6d_ip_tool -P $INPUT_SERVER
$ROT_BITS $key`
PORT="${PORT%\\n}"
if [[ $MT6D_SOURCE -ne 0 ]]
then
#Get source address of client from key
subnet=${key:6:1}
offset=${key:8:1}
#Only add 699 to account for client#-1 is actually 700
(mt6d client start) on subnet
offset=$(( $offset + 699 ))
orig_source="dead:beef:0:$subnet::$offset"
SOURCE=`/home/pdimarco/mt6d_ip_tool -A $orig_source
$ROT_BITS $key`
fi
else
#Attack random address on that subnet
num_1=`printf '%x\n' $RANDOM`
num_2=`printf '%x\n' $RANDOM`
num_3=`printf '%x\n' $RANDOM`
num_4=`printf '%x\n' $RANDOM`
SERVER="dead:beef:0:1:$num_1:$num_2:$num_3:$num_4"
PORT=$RANDOM
if [[ $MT6D_SOURCE -ne 0 ]]
then
#Use real source address
SOURCE=`ifconfig eth0 | awk '{print $3}' |
grep dead:beef | sed 's/\(.*\).../\1/'`
fi
fi
curr_time=`date +"%s"`
echo -e "$curr_time\tMT6D_Address_$SERVER_$PORT" >>
/home/pdimarco/`hostname`[]${ATTACK}[]${NUM_ATTACKERS}[]
${REQUEST}[]${NUM_CLIENTS}.txt
fi
#Call attack
attack_$ATTACK
#Watch attack until it is done
while [ $PID ]; do
sleep $SLEEP_TIME
curr_time=`date +"%s"`
if [[ $curr_time -gt $end_time ]]
```

```
then
kill $PID
fi
PID=$(ps -p $PID | awk '{print $1 }' | grep $PID 2> /dev/null)
done
curr_time=`date +"%s"`
done

#Record ending time
curr_time=`date +"%s"`
echo -e "FINISHED\t$curr_time" >> /home/pdimarco/`hostname`[]${ATTACK}[]
${NUM_ATTACKERS}[]${REQUEST}[]${NUM_CLIENTS}.txt
}

attack_slowloris () {
perl /home/pdimarco/slowloris.pl -dns [$SERVER] -timeout 30 &>
/dev/null &
PID=$!
}

attack_dos-new-ip6 () {
dos-new-ip6 eth0 &> /dev/null &
PID=$!
}

attack_denial6_1 () {
#test_case=$(( `date +"%s"` % 2 ))
#test_case=$(( $test_case + 1 ))
#denial6 eth0 $SERVER $test_case &> /dev/null &
denial6 eth0 $SERVER 1 &> /dev/null &
PID=$!
if [[ $MT6D_DEST -ne 0 ]]
then
sleep $MT6D_ROT_TIME
kill $PID
fi
}

attack_denial6_2 () {
denial6 eth0 $SERVER 2 &> /dev/null &
PID=$!
if [[ $MT6D_DEST -ne 0 ]]
```

```
then
sleep $MT6D_ROT_TIME
kill $PID
fi
}

attack_http_get_flood () {
python /home/pdimarco/get_flooder.py $SERVER &> /dev/null &
PID=$!
if [[ $MT6D_DEST -ne 0 ]]
then
sleep $MT6D_ROT_TIME
kill $PID
fi
}

attack_syn_flood () {
#Confirm ip6tables set to drop RST packets
#ip6tables -A OUTPUT -p tcp --tcp-flags RST RST -j DROP
if [[ $MT6D_DEST -ne 0 ]]
then
#Executing only for a second to check if address changes
#echo -e "$SOURCE $SERVER\n"
python /home/pdimarco/syn_flood.py -s $SOURCE -d $SERVER -p
$PORT -t 0.5 &> /dev/null &
else
python /home/pdimarco/syn_flood.py -s $SOURCE -d $SERVER -p $PORT
-t $TEST_TIME &> /dev/null &
fi
PID=$!
}

#Execute main routine
main
```

Appendix C

Client Scripts

C.1 Client.sh

This script allows the master to call the client with parameters on what communication to start up.

```
#!/bin/bash
#
# Run MT6D performance tests - Client Script
# This script is to run client work

#1 - type of attack
#2 - number of attackers
#3 - type of request
#4 - number of clients
#5 - server
#6 - test_time

#Running Task PID
PID=0
#Sleep time between task checks
SLEEP_TIME=0.001
#Client variables
readonly DOS_IP6_EXTRA_TIME=180
readonly ATTACK=$1
readonly NUM_ATTACKERS=$2
readonly REQUEST=$3
readonly NUM_CLIENTS=$4
readonly SERVER=$5
```

```
readonly TEST_TIME=$6

main () {
#Run dos-new-ip6 for 3 minutes before test to have effects work
#Client will just report starting 3 minutes after master actually started
#him
if [ "$ATTACK" == "dos-new-ip6" ]
then
sleep $DOS_IP6_EXTRA_TIME
fi
#Find finishing time
curr_time=`date +"%s"`
end_time=$(( $curr_time + $TEST_TIME ))

#Completion count
test_count=0

#Record starting time
echo -e "STARTED\t$curr_time\t$end_time" >> /home/pdimarco/`hostname`[]
${ATTACK}[]${NUM_ATTACKERS}[]${REQUEST}[]${NUM_CLIENTS}.txt

#Do work for specified time
while [ $curr_time -lt $end_time ]; do
client_$REQUEST
while [ $PID ]; do
sleep $SLEEP_TIME
curr_time=`date +"%s"`
if [ $curr_time -gt $end_time ]
then
kill $PID
test_count=$(( $test_count - 1 ))
fi
PID=$(ps -p $PID | awk '{print $1 }' | grep $PID 2> /dev/null)
done
test_count=$(( $test_count + 1 ))
curr_time=`date +"%s"`
echo -e "$curr_time\t$test_count" >> /home/pdimarco/`hostname`[]${ATTACK}[]
${NUM_ATTACKERS}[]${REQUEST}[]${NUM_CLIENTS}.txt
done
rm random_* 2> /dev/null
}
```

```
client_iperf_1m_tcp () {
iperf -V -c $SERVER -n 1M &> /dev/null &
PID=$!
}

client_iperf_10m_tcp () {
iperf -V -c $SERVER -n 10M &> /dev/null &
PID=$!
}

client_iperf_100m_tcp () {
iperf -V -c $SERVER -n 100M &> /dev/null &
PID=$!
}

client_iperf_1g_tcp () {
iperf -V -c $SERVER -n 1000M &> /dev/null &
PID=$!
}

client_iperf_500k_udp () {
ping6 -c 1 $SERVER
OUTPUT=$?
if [[ $OUTPUT -eq 0 ]]
then
iperf -V -u -c $SERVER -n 500K &> /dev/null &
PID=$!
else
PID=0
fi
}

client_iperf_1m_udp () {
ping6 -c 1 $SERVER
OUTPUT=$?
if [[ $OUTPUT -eq 0 ]]
then
iperf -V -u -c $SERVER -n 1M &> /dev/null &
PID=$!
else
PID=0
fi
```

```
}

client_iperf_10m_udp () {
#barely gets 1 under 2 min
ping6 -c 1 $SERVER
OUTPUT=$?
if [[ $OUTPUT -eq 0 ]]
then
iperf -V -u -c $SERVER -n 10M &> /dev/null &
PID=$!
else
PID=0
fi
}

client_iperf_100m_udp () {
#Not realistic for 2 minute test
ping6 -c 1 $SERVER
OUTPUT=$?
if [[ $OUTPUT -eq 0 ]]
then
iperf -V -u -c $SERVER -n 100M &> /dev/null &
PID=$!
else
PID=0
fi
}

client_iperf_1g_udp () {
#Not realistic
ping6 -c 1 $SERVER
OUTPUT=$?
if [[ $OUTPUT -eq 0 ]]
then
iperf -V -u -c $SERVER -n 1000M &> /dev/null &
PID=$!
else
PID=0
fi
}

client_wget_500k () {
```

```
wget -6 http://[$SERVER]/random_500k &> /dev/null &
PID=$!
}

client_wget_1m () {
wget -6 http://[$SERVER]/random_1M &> /dev/null &
PID=$!
}

client_wget_10m () {
wget -6 http://[$SERVER]/random_10M &> /dev/null &
PID=$!
}

client_wget_50m () {
wget -6 http://[$SERVER]/random_50M &> /dev/null &
PID=$!
}

client_wget_500m () {
wget -6 http://[$SERVER]/random_500M &> /dev/null &
PID=$!
}

client_wget_1g () {
wget -6 http://[$SERVER]/random_1G &> /dev/null &
PID=$!
}

#Execute main routine
main
```

Bibliography

[1] Attacking the IPv6 Protocol Suite. http://www.thc.org/papers.php, Oct. 2008.

[2] G. Badishi, A. Herzberg, I. Keidar, O. Romanov, and A. Yachin. An Empirical Study of Denial of Service Mitigation Techniques. In *Reliable Distributed Systems, 2008. SRDS '08. IEEE Symposium on*, pages 115 –124, oct. 2008.

[3] M. Barylski. On IPSec performance testing of IPv4/IPv6 IPSec gateway. In *Information Technology, 2008. IT 2008. 1st International Conference on*, pages 1 –4, may 2008.

[4] Terry Benzel, Robert Braden, Dongho Kim, Clifford Neuman, Anthony Joseph, Keith Sklower, Ron Ostrenga, and Stephen Schwab. Design, deployment, and use of the DETER testbed. In *Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test on DETER Community Workshop on Cyber Security Experimentation and Test 2007*, Berkeley, CA, USA, 2007.

[5] José Brustoloni. Protecting electronic commerce from distributed denial-of-service attacks. In *Proceedings of the 11th international conference on World Wide Web*, WWW '02, pages 553–561, New York, NY, USA, 2002. ACM.

[6] Ruiliang Chen, Jung-Min Park, and Randolph Marchany. A divide-and-conquer strategy for thwarting distributed denial-of-service attacks. *IEEE Trans. Parallel Distrib. Syst.*, 18(5):577–588, May 2007.

[7] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998. Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946.

[8] Zhenhai Duan, Peng Chen, Fernando Sanchez, Yingfei Dong, Mary Stephenson, and James Michael Barker. Detecting Spam Zombies by Monitoring Outgoing Messages. *IEEE Trans. Dependable Secur. Comput.*, 9(2):198–210, March 2012.

[9] Matthew Dunlop, Stephen Groat, William Urbanski, Randy Marchany, and Joseph Tront. MT6D: A Moving Target IPv6 Defense. In *MILITARY COMMUNICATIONS CONFERENCE, 2011 - MILCOM 2011*, pages 1321–1326, 2011.

[10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785, 6266, 6585.

[11] R. Gayraud and B. Lourdelet. Network Time Protocol (NTP) Server Option for DHCPv6. RFC 5908 (Proposed Standard), June 2010.

[12] Baha Hebrawi. *Open systems interconnection: upper layer standards and practices.* McGraw-Hill, Inc., New York, NY, USA, 1993.

[13] R. Hinden and S. Deering. IP Version 6 Addressing Architecture. RFC 4291 (Draft Standard), February 2006. Updated by RFCs 5952, 6052.

[14] K. Ingham and S Forrest. A History and Survey of Network Firewalls. `http://www.cs.unm.edu/~treport/tr/02-12/firewall.pdf`, 2002.

[15] P.A. Karger and R.R. Schell. Thirty years later: lessons from the multics security evaluation. In *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pages 119–126, 2002.

[16] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401 (Proposed Standard), November 1998. Obsoleted by RFC 4301, updated by RFC 3168.

[17] Shen Li, V. Sivaraman, A. Krumm-Hellerl, and C. Russell. A Dynamic Stateful Multicast Firewall. In *Communications, 2007. ICC '07. IEEE International Conference on*, pages 1280 –1285, June 2007.

[18] A.X. Liu and M.G. Gouda. Firewall Policy Queries. *Parallel and Distributed Systems, IEEE Transactions on*, 20(6):766–777, 2009.

[19] Giorgia Lodi, Leonardo Querzoni, Roberto Baldoni, Mirco Marchetti, Michele Colajanni, Vita Bortnikov, Gregory Chockler, Eliezer Dekel, Gennady Laventman, and Alexey Roytman. Defending financial infrastructures through early warning systems: the intelligence cloud approach. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*, CSIIRW '09, pages 18:1–18:4, New York, NY, USA, 2009. ACM.

[20] Mohammad M. Masud, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani Thuraisingham. Peer to peer botnet detection for cyber-security: a data mining approach. In *Proceedings of the 4th annual workshop on Cyber security and information intelligence research: developing strategies to meet the cyber security and information intelligence challenges ahead*, CSIIRW '08, pages 39:1–39:2, New York, NY, USA, 2008. ACM.

[21] M. Mathis and J. Heffner. Packetization Layer Path MTU Discovery. RFC 4821 (Proposed Standard), March 2007.

[22] Mixter. "Tribe Flood Network 3000": A theoretical review of what exactly Distributed DOS tools are, how they can be used, what more dangerous features can be implemented in the future, and starting points on establishing Network Intrusion Detection Rules for DDOS. `http://packetstormsecurity.com/files/10525/tfn3k.txt.html`, 2000.

[23] P.V. Mockapetris. Domain names - concepts and facilities. RFC 1034 (INTERNET STANDARD), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592, 5936.

[24] J.C. Mogul and S.E. Deering. Path MTU discovery. RFC 1191 (Draft Standard), November 1990.

[25] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Neighbor Discovery for IP version 6 (IPv6). RFC 4861 (Draft Standard), September 2007. Updated by RFC 5942.

[26] Trung Hai Nguyen, Cao Thanh Doan, Van Quan Nguyen, Thi Huyen Trang Nguyen, and Minh Phuong Doan. Distributed defense of distributed DoS using pushback and communicate mechanism. In *Advanced Technologies for Communications (ATC), 2011 International Conference on*, pages 178–182, 2011.

[27] J. Postel. Transmission Control Protocol. RFC 793 (INTERNET STANDARD), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[28] Supranamaya Ranjan, Ram Swaminathan, Mustafa Uysal, Antonio Nucci, and Edward Knightly. DDoS-shield: DDoS-resilient scheduling to counter application layer attacks. *IEEE/ACM Trans. Netw.*, 17(1):26–39, February 2009.

[29] C. Sun, C. Hu, and B. Liu. SACK2: effective SYN flood detection against skillful spoofs. *Information Security, IET*, 6(3):149 –156, sept. 2012.

[30] S. Thomson, T. Narten, and T. Jinmei. IPv6 Stateless Address Autoconfiguration. RFC 4862 (Draft Standard), September 2007.

[31] Xin Wen, Changqiao Xu, Jianfeng Guan, Wei Su, and Hongke Zhang. Performance investigation of IPSEC protocal over IPv6 network. In *Advanced Intelligence and Awarenss Internet (AIAI 2010), 2010 International Conference on*, pages 174 –177, oct. 2010.

[32] Yi Xie and Shun-Zheng Yu. Monitoring the application-layer DDoS attacks for popular websites. *IEEE/ACM Trans. Netw.*, 17(1):15–25, February 2009.

[33] Xinyu Yang, Ting Ma, and Yi Shi. Typical DoS/DDoS Threats under IPv6. In *Computing in the Global Information Technology, 2007. ICCGI 2007. International Multi-Conference on*, page 55, march 2007.

[34] T. Yatagai, T. Isohara, and I. Sasase. Detection of HTTP-GET flood Attack Based on Analysis of Page Access Behavior. In *Communications, Computers and Signal Processing, 2007. PacRim 2007. IEEE Pacific Rim Conference on*, pages 232 –235, aug. 2007.

[35] Saman Taghavi Zargar, James Joshi, and David Tipper. A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks. *Communications Surveys Tutorials, IEEE*, 15(4):2046–2069, 2013.