Appendix E:

Simulation - Testing and Results

1 Introduction

The aim of a Distributed Denial of Service attack is to disrupt the connection between a service, such as an application server or DNS server, and that services users.

Denial of Service (DoS) attacks typically exploited vulnerabilities with network or application layer protocols, for example, Syn Floods or HTTP Floods. With these types of attacks, spoofed IP addresses were typically used, both to mask the IP address of the attacker, and to take advantages of vulnerabilities of the protocols being used.

As DoS mitigation techniques improved, and attacks using IP spoofing became easier to defend against, attackers began utilising botnets to perform amplification attacks. Distributed Denial of Service (DDoS) attacks utilise large number of hosts to perform attacks. These types of attacks prove more difficult to defend against than typical Denial of Service attacks.

In order to extensively evaluate the impact of Distributed Denial of Service attacks, and the role which IoT devices play in modern DDoS attacks, simulations can be used. Simulations can show the impact to a network, and the disruption caused by these attacks, without having to use real network devices.

1.1 Simulation Software

There were a number of simulation software packages considered for the tests contained in this document. Each of these packages offer their own set of advantages and disadvantages.

1.1.1 ns-3

The most commonly used network simulation software is "ns-3"; a "discrete-event network simulator for Internet systems"[1]. This package is commonly used for research purposes, and numerous examples of D/DoS simulations implemented using ns-3 can be found.

1.1.2 Cooja

Another simulation software package considered was "Cooja". This package simulates Wireless Sensor Networks built on the Contiki IoT operating system[2]. While the underlying system being simulated is an IoT device, there was little information to be found on implementing the types of testing required, and as such this software package was ruled out.

1.1.3 Mininet

The final software package considered was "Mininet". Mininet is a network simulator which uses Linux containers to emulate network devices[3]. As the network devices are emulated using containers, each host uses real Linux Kernel code.

As the Mininet hosts run Linux Kernel code, Linux applications can be easily run. This is advantageous as common attack tools can be installed and executed. Each host can be assigned a unique IP address allowing the attacking host to address it's target as in

a real attack situation. As each host is implemented as a container, a terminal can be open for each host for manual command execution.

Mininet implements Software Defined Networking using the OpenFlow protocol. By default, Mininet uses the OpenFlow reference controller, however, it also allows for the use of remote controllers. The "Floodlight" controller will be used for this purpose, as it has a GUI application which can display the network topology.

Mininet virtualizes the network links between each host or switch in the network. These links can be assigned parameters such as a delay time on the link, or a set bandwidth.

The Python API implemented by Mininet allows for the topology and the terminal commands to be created and executed programmatically. The Python API will be used in order to execute repeatable tests with reproducible results.

Due to the advantages offered by the Mininet simulation software package, this will be the simulator of choice for testing. There are however a number of disadvantages which must be noted[4].

1. Containers share the file system of the host, i.e. the desktop or VM on which Mininet is being run
2. A network cannot exceed the bandwidth of a single server
3. Non-Linux-compatible OpenFlow switches and applications are not supported

While these limitations must be kept in mind, they will not prove to be a hindrance for the purposes of these simulations.

2 Attack Types

For testing purposes, a number of different attack types were be simulated. SYN Flood attacks and ICMP Flood attacks were be simulated.

2.1 SYN Flood

SYN floods are an attack which exploit the TCP protocols TCP handshake. The attacking node sends TCP SYN packets to the target using spoofed IP addresses. The target device will reply with a SYN-ACK, and wait for the initiator to reply with a final ACK packet. As the attacking node is using a spoofed IP address, it never responds to the SYN-ACK, and so the target node will wait indefinitely for the ACK packet[5].

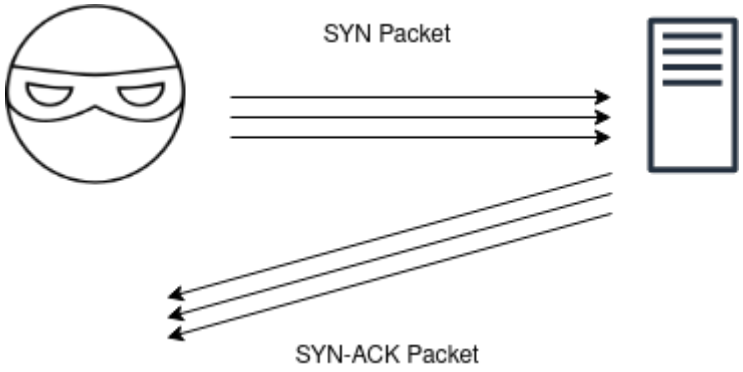

Figure 1: SYN Flood Attack

In order to perform a SYN Flood attack, the "hping" tool is used. Hping is a tool for creating and transmitting TCP, UDP or ICMP packets. This tool has multiple functions, such as port scanning, and firewall mapping, but can also be utilised as a Denial of Service tool. By utilising the tools "rand-source" and "flood" flags, the tool can execute a SYN flood, sending TCP SYN packets to the target address via spoofed IP addresses.

2.2 ICMP Flood

An ICMP flood is a distributed denial of service attack, performed using a botnet[6]. ICMP requests take server resources to process and reply to. By sending large numbers of ICMP requests, the servers resources can be exhausted.

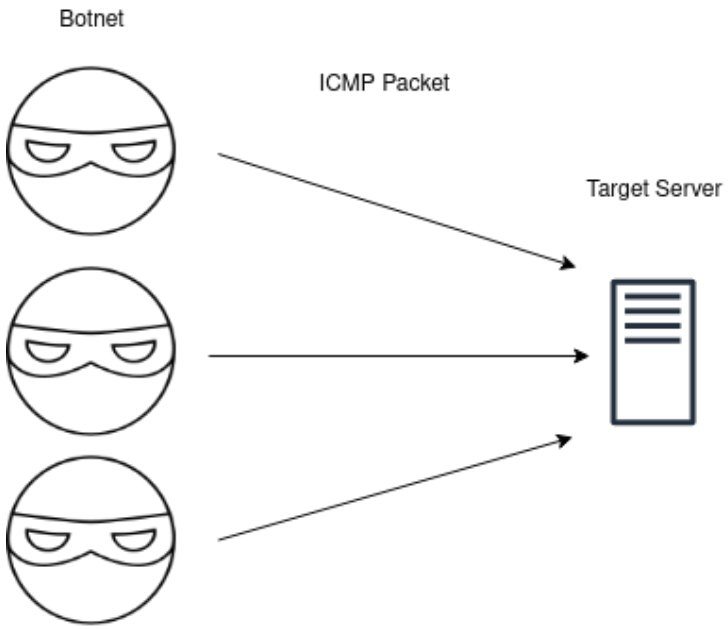

Figure 2: ICMP Flood Attack

The "ping" tool can be used to execute an ICMP flood. By using the "-f" flag, the ping tool can be used to launch an ICMP flood from a single attacking node.

3 Denial of Service Simulation

For the initial Denial of Service simulation, the network topology is as shown in Figure 3. This network consists of an attacking traffic source (shown on the left), a legitimate user traffic source (shown on the right), and a server (shown on the bottom).

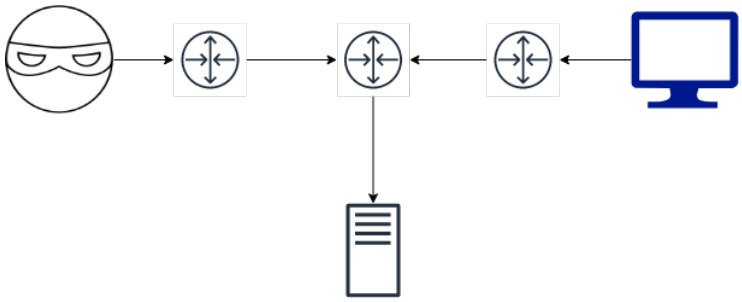

Figure 3: Denial of Service Network Topology

This topology is implemented using the Mininet Python API. Each of the traffic sources is implemented as a "host", and each switch as a "switch". The links between each node in the network must be defined. The completed topology configuration is as follows:

```
class MyTopo( Topo ):
        "Dos Topology"

        def __init__( self ):
                "Create custom topo"

                # Initialize topology
                Topo.__init__(self)

                # Add hosts and switches
                h1 = self.addHost( 'h1' )
                h2 = self.addHost( 'h2' )
                h3 = self.addHost( 'h3' )
                s1 = self.addSwitch( 's1' )
                s2 = self.addSwitch( 's2' )
                s3 = self.addSwitch( 's3' )

                # Add links
```

```
        self.addLink( h1, s1 )
        self.addLink( s1, s2 )
        self.addLink( s2, h3 )
        self.addLink( s2, s3 )
        self.addLink( s3, h2 )
```

Listing 1: DoS Simulation Network Topology

3.1 SYN Flood

3.1.1 Attack

For the SYN Flood, the malicious traffic source launches the attack using the following "hping" command:

```
$ hping -S -p <TargetPort> --rand-source --flood <
    TargetIP>
```

Listing 2: SYN Flood DoS Command

This command can be broken down as follows:

- The -S flag denotes setting the SYN flag for the transmission.
- The -p flag sets the destination port number for the transmit packet, in this case the target server.
- The –rand-source flag sets the IP address to spoofed values for each packet that is transmit.
- The "–flood" flag sends packets as fast as is possible.

3.1.2 Mitigation

3.2 ICMP Flood DoS Attack

3.2.1 Attack

For the ICMP Flood, the malicious traffic source launches the attack using the following "ping" command:

```
$ sudo ping -f -l 100000000000000 -s 1500
```

Listing 3: ICMP Flood DoS Command

This command can be broken down as follows:

- The -f flag denotes a flood, as such the tool outputs "ECHO_REQUEST" packets as fast as is possible.
- The -l flag denotes the number of packets to send without waiting for a reply. The value associated with this flag is set to an arbitrarily high number.
- The -s flag denotes the packet size, the value for which can be determined by observing the output of the "ifconfig eth0". For the DoS network, this value was set to 1500 bytes.

3.2.2 Mitigation

4 Distributed Denial of Service Simulation

For the Distributed Denial of Service simulation, an expanded version of the topology used in Figure 3, consisting of a significantly larger number of attacking sources is used. This reflects the increase in attackers, which is common in DDoS attacks involving IoT botnets. It was chosen to use 8 switch nodes, each connected to 8 host

nodes, and one switch and host connected as the target server to these networks. The complete topology can be seen in Figure 4.

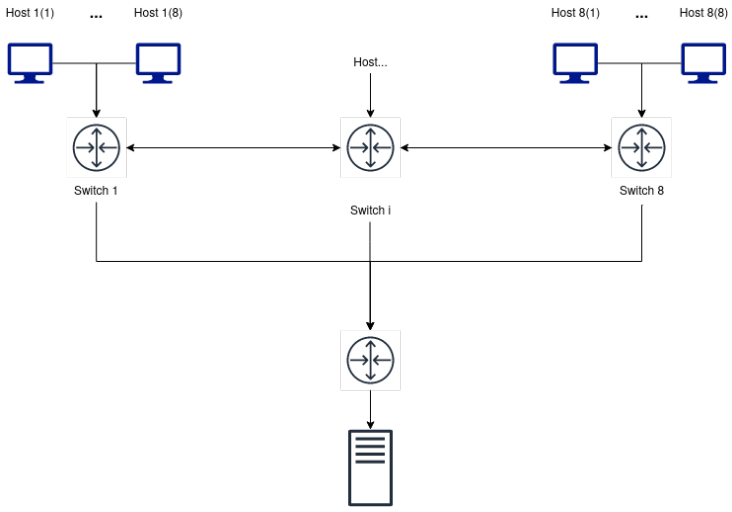

Figure 4: Distributed Denial of Service Network Topology

Within this topology, any number of hosts can be used to launch the distributed denial of service attack. The topology is again implemented using the Mininet Python API as follows:

```
class LineTopo( Topo ):
    "Linear topology example."

    def __init__( self ):
        "Create linear topology"
        super(LineTopo, self).__init__()
        h1 = []
        h2 = []
        h3 = []
        h4 = []
        h5 = []
```

```
        h6 = []
        h7 = []
        h8 = []
        s = []          # list of switches
        server = []

        M=9
        N=8
        # add N hosts  h1..hN
        for i in range(1,N+1):
            h1.append(self.addHost('h1' + str(i)))
            h2.append(self.addHost('h2' + str(i)))
            h3.append(self.addHost('h3' + str(i)))
            h4.append(self.addHost('h4' + str(i)))
            h5.append(self.addHost('h5' + str(i)))
            h6.append(self.addHost('h6' + str(i)))
            h7.append(self.addHost('h7' + str(i)))
            h8.append(self.addHost('h8' + str(i)))

        # add N switches s1..sN
        for i in range(1,M+1):
            s.append(self.addSwitch('s' + str(i)))

        # add target server
        server.append(self.addHost('server'))

        # Add links from hi to si
        for i in range(N):
            self.addLink(h1[i], s[0])
            self.addLink(h2[i], s[1])
```

```
            self.addLink(h3[i], s[2])
            self.addLink(h4[i], s[3])
            self.addLink(h5[i], s[4])
            self.addLink(h6[i], s[5])
            self.addLink(h7[i], s[6])
            self.addLink(h8[i], s[7])

        # Add links from target server to s8
        self.addLink(server[0], s[8])

        # link switches
        for i in range(M-2):
            self.addLink(s[i],s[i+1])

        # link all switches to target switch
        for i in range(M-2):
            self.addLink(s[i], s[M-1])


topos = { 'mytopo': (lambda: LineTopo() ) }
```

Listing 4: DDoS Simulation Network Topology

5 Results

Packet analysis for each of the simulations was performed using the Wireshark tool. This tool can be run on the PC running Mininet, and configured to monitor the loopback address for OpenFlow packets, as described in the "Start Wireshark" section here [7], or can be run on one of the Mininet host nodes.

5.1 Denial of Service Results

The Denial of Service topology was first verified within the Floodlight web server. Figure 5 displays the results of this:

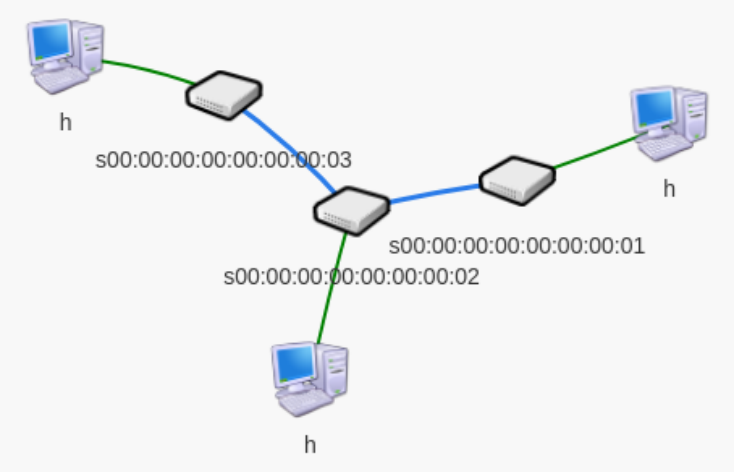

Figure 5: Denial of Service Floodlight Topology

5.1.1 SYN Flood

Using the Wireshark packet analyzer tool. test est

5.1.2 ICMP Flood

5.2 Distributed Denial of Service

The Distributed Denial of Service topology was verified within the Floodlight web server. Figure 6 displays the results of this, with 8 switches with 8 hosts each, and a final switch connected to the single target server node.

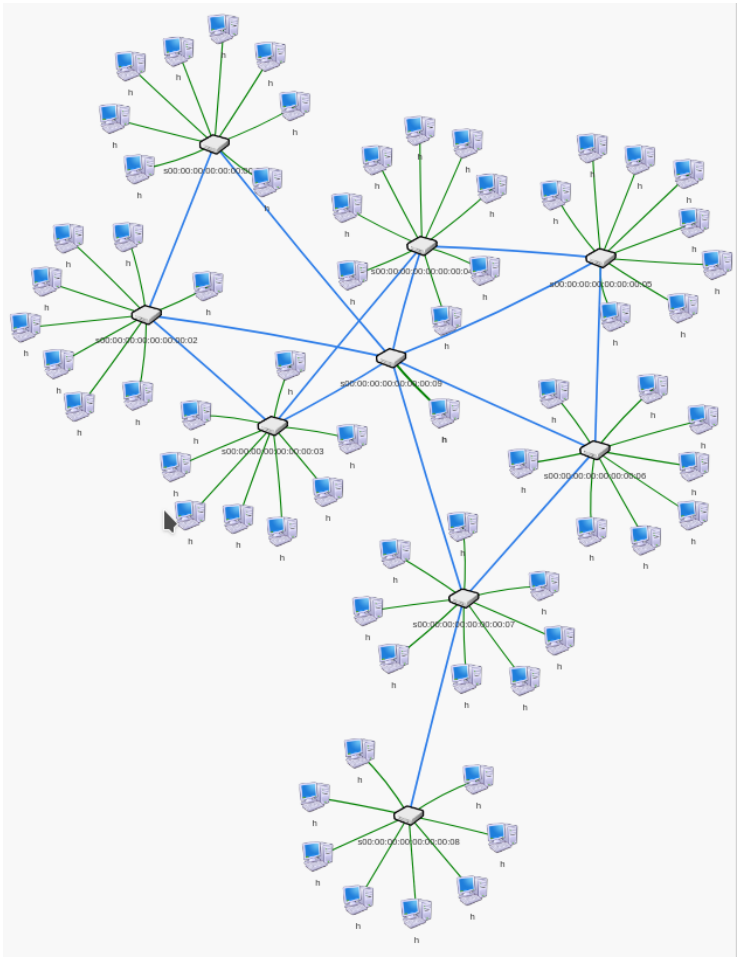

Figure 6: Distributed Denial of Service Floodlight Topology

5.2.1 SYN Flood

5.2.2 ICMP Flood

6 Resources

The following list outlines both the hardware and software resources which were utilised in the execution of these simulations:

6.1 Hardware:

As these simulations utilised Linux containers, only a single PC was required to execute the simulations. The specifications of the laptop PC are as follows:

- HP Envy 17
  - Intel Core i7-6500U (Dual Core)
  - 12GB RAM

6.2 Software:

For these simulations, the Linux operating system was used for it's ease of installation of software packages, and also for the availability of containers. Note that a Linux virtual machine would be a suitable replacement.

6.2.1 Operating System:

- Manjaro Linux
  - Kernel Version 5.4.52-1-MANJARO
  - OS Type: 64-bit

6.2.2 Simulation Software:

- Mininet
  - Version: 2.3.0d6
- Floodlight Controller
  - Version: 1.2

7 Reproduction

The following steps must be performed in order to reproduce the achieved simulations. As the Manjaro Linux distribution was used for these simulations, all package manager instructions will be given for Manjaro.

7.1 Mininet

Mininet must first be installed. This can be done according to the Mininet documentation **??**. For Debian based systems, Mininet can be installed via the "apt" package manager, using the command:

```
$ apt-get install mininet
```

Listing 5: Debian-based Distro Mininet Install

For Arch based distributions, Mininet may be installed using the Arch User Repository using the following command:

```
$ yay -S mininet-git
```

Listing 6: Arch-based Distro Mininet Install

To test the installation, execute the command:

```
$ sudo mn --test pingall
```

Listing 7: Mininet installation test

If the error "ovs-vsctl: unix:/run/openvswitch/db.sock database connection failed" occurs, the Open vSwitch must be started using the command:

```
$ sudo /usr/share/openvswitch/scripts/ovs-ctl start
```

Listing 8: Open vSwitch service start command

7.2 Floodlight OpenFlow Controller

The Floodlight OpenFlow Controller can be installed via GitHub**??**. For version 1.2 (as used for these simulation), the Java 8 development kit must be installed:

```
$ sudo pacman -S openjdk8-src
```

Listing 9: openjdk8 installation

Floodlight has a number of dependencies which can be installed via the following command:

```
$ sudo pacman -S git ant maven python-dev
```

Listing 10: Floodlight Dependencies

To download and build Floodlight, execute the following commands:

```
$ git clone git://github.com/floodlight/floodlight.git
$ cd floodlight
$ git submodule init
$ git submodule update
$ ant
```

```
# If the ant build fails, Floodlight can be built using
    the following Maven command
$ sudo mvn package
```

Listing 11: Floodlight installation commands

7.3 Source Code

The source code for the simulations can be found on GitHub, and can be downloaded using the following command:

```
$ git clone https://github.com/mLenehanDCU/MininetCode.
    git
```

Listing 12: Source code download

8 Future Work

For work built upon these simulations, the first recommendation is to use either the Mininet VM image, or a Linux distribution such as Ubuntu. While Manjaro is based on Arch Linux, which has access to a large number of packages, Mininet, along with a number of the packages used for sending and monitoring traffic flow were required to be installed from the Arch User Repositories (AUR). For comparison, Mininet, is available from Ubuntus default "apt" package manager, along with all of the other software packages used.

9 Conclusion

References

[1] Nsnam, 20AD. [Online]. Available: https://www.nsnam.org/.

[2] B. Thébaudeau, *An introduction to cooja*, Sep. 2019. [Online]. Available: https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja.

[3] 2018. [Online]. Available: http://mininet.org/.

[4] *Mininet overview*, 2018. [Online]. Available: http://mininet.org/overview/.

[5] 2020. [Online]. Available: https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/.

[6] 2020. [Online]. Available: https://www.cloudflare.com/learning/ddos/ping-icmp-flood-ddos-attack/.

[7] M. Team, *Mininet walkthrough*. [Online]. Available: http://mininet.org/walkthrough/.