Comprehensive Test Scenarios and Simulation Framework for Evaluating Snort’s DDoS Detection Module in Mininet

1

2

3

4

5

# Objectives

* Develop detailed test scenarios for SYN Flood, UDP/ICMP Flood, HTTP Flood, and Slowloris attacks, covering low, medium, and high intensities.
* Configure a Mininet-based virtual network with realistic topology and traﬀic pat- terns.
* Define and implement Snort rules to detect each attack type based on the module’s detection mechanisms.
* Simulate attacks and evaluate the module’s performance in terms of detection pre- cision, FPR, detection time, blocking time, and system resource usage.
* Provide a scalable framework for future multi-host attack scenarios and integration with monitoring tools like Prometheus.

# Mininet Environment Setup

A virtual network is constructed in Mininet to emulate a realistic enterprise environment. The topology and configuration are designed to balance complexity and scalability while ensuring accurate simulation of DDoS attacks.

## Network Topology

* + - **Topology**: Tree, depth=1, fanout=5 (1 switch connected to 5 hosts).

### Hosts:

* + - * h1: Victim host running Snort with the DDoS detection module. IP: 10.0.0.1.
      * h2, h3: Attackers for SYN Flood and UDP/ICMP Flood. IPs: 10.0.0.2, 10.0.0.3.
      * h4: Attacker for HTTP Flood and Slowloris. IP: 10.0.0.4.
      * h5: Host generating normal traﬀic. IP: 10.0.0.5.
    - **Switch**: s1, using Open vSwitch (OVS).
    - **Bandwidth**: 10 Mbps per link, with a latency of 1ms to simulate a local network.
    - **Controller**: Default OVS controller in Mininet.

## Mininet Script

The following Python script configures the Mininet topology:

from mininet.topo import Topo from mininet.net import Mininet

from mininet.node import OVSSwitch , Controller from mininet.cli import CLI

from mininet.log import setLogLevel

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 6  7  8  9  10  11 | from mininet.link import TCLink  class DDoSTopo(Topo): def build(self):  s1 = self. addSwitch('s1 ')  h1 = self.addHost('h1 ', ip='10.0.0.1/24 ') # Victim | | | | | |
| 12 | h2 | = self.addHost('h2 ', | ip='10.0.0.2/24 ') | # | Attacker | for |
|  |  | SYN/ UDP Flood |  |  |  |  |
| 13 | h3 | = self.addHost('h3 ', | ip='10.0.0.3/24 ') | # | Attacker | for |
|  |  | SYN/ UDP Flood |  |  |  |  |
| 14 | h4 | = self.addHost('h4 ', | ip='10.0.0.4/24 ') | # | Attacker | for |
|  |  | HTTP/ Slowloris |  |  |  |  |
| 15 | h5 | = self.addHost('h5 ', | ip='10.0.0.5/24 ') | # | Normal tr | affic |
| 16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | self.addLink(h1 , s1 , cls=TCLink , bw=10 , delay='1ms') self.addLink(h2 , s1 , cls=TCLink , bw=10 , delay='1ms') self.addLink(h3 , s1 , cls=TCLink , bw=10 , delay='1ms') self.addLink(h4 , s1 , cls=TCLink , bw=10 , delay='1ms') self.addLink(h5 , s1 , cls=TCLink , bw=10 , delay='1ms')  def run():  topo = DDoSTopo()  net = Mininet(topo=topo , switch=OVSSwitch , controller=Controller  , link=TCLink) net.start()  CLI(net) net.stop()  if name == ' main ': setLogLevel('info') run() | | | | | |

## Victim Host Configuration

Host h1 (the victim) is configured as follows:

* + - **Operating System**: Ubuntu 20.04 LTS.
    - **Snort Version**: 2.9.20, compiled with support for libpcap and daq.
    - **Web Server**: Apache2 (for HTTP Flood and Slowloris tests), listening on port 80.
    - **Monitoring**: Prometheus node exporter for resource usage (CPU, RAM) and custom metrics exported by the DDoS module.

# Snort Rule Configuration

The DDoS detection module integrates with Snort through custom rules that align with its detection mechanisms (statistical analysis, behavioral profiling). Below are the Snort rules for each attack type, designed to trigger alerts based on traﬀic patterns.

## SYN Flood Rule

Detects a high rate of SYN packets without corresponding ACKs (indicative of incomplete TCP handshakes):

alert tcp any any -> 10.0.0.1 80 (msg:" SYN Flood Detected"; flags:S

,12; flow: to\_server; detection\_filter:track by\_dst, count 500 , seconds 1; sid:1000001; rev:1;)

1

This rule triggers if more than 500 SYN packets are sent to port 80 of 10.0.0.1 within 1 second.

## UDP/ICMP Flood Rule

Detects a high volume of UDP traﬀic to a single destination:

alert udp any any -> 10.0.0.1 any (msg:" UDP Flood Detected"; detection\_filter:track by\_dst, count 1000 , seconds 1; sid

:1000002; rev:1;)

1

This rule triggers if more than 1000 UDP packets are sent to 10.0.0.1 within 1 second.

## HTTP Flood Rule

Detects excessive HTTP GET requests to the web server:

alert tcp any any -> 10.0.0.1 80 (msg:" HTTP Flood Detected"; flow: to\_server , established; content:" GET"; http\_method;

detection\_filter:track by\_dst, count 200 , seconds 1; sid:1000003; rev:1;)

1

This rule triggers if more than 200 GET requests are sent to port 80 of 10.0.0.1 within 1 second.

## Slowloris Rule

Detects prolonged HTTP connections (indicative of Slowloris behavior):

alert tcp any any -> 10.0.0.1 80 (msg:" Slowloris Attack Detected"; flow:to\_server , established; content:" GET"; http\_method; flowbits: set, slowloris; flowbits:noalert; sid:1000004; rev:1;)

alert tcp any any -> 10.0.0.1 80 (msg:" Slowloris - Prolonged Connection"; flow:to\_server , established; flowbits:isset, slowloris

; detection\_filter:track by\_dst, count 50 , seconds 10; sid

:1000005; rev:1;)

1

2

The first rule tags connections with a GET request, and the second triggers if 50 or more such connections persist for 10 seconds.

# Test Scenarios

The test scenarios are designed to evaluate the module across normal traﬀic and DDoS attack conditions, with varying intensities to stress-test detection and mitigation capa- bilities.

## Scenario 1: Normal Traﬀic

* + - **Description**: Host h5 generates legitimate TCP/HTTP traﬀic to h1 to simulate normal network activity.
    - **Tool**: iperf for TCP traﬀic, curl for HTTP requests.

### Parameters:

* + - * Protocol: TCP/HTTP, port 80.
      * Rate: 1 Mbps (TCP), 10 requests/second (HTTP).
      * Duration: 300 seconds.

### Sample Commands:

iperf -c 10.0.0.1 -p 80 -b 1M -t 300

curl -s -o / dev/ null http ://10.0.0.1 --max-time 300

1

2

* + - **Expectation**: No DDoS alerts triggered, FPR < 3%.

## Scenario 2: SYN Flood Attack

* + - **Description**: Hosts h2 and h3 send SYN packets to h1 to simulate a SYN Flood attack.
    - **Tool**: hping3.

### Parameters:

* + - * Protocol: TCP, port 80.
      * Intensity Levels:

∗ Low: 500 packets/second per host.

∗ Medium: 1000 packets/second per host.

∗ High: 2000 packets/second per host.

* + - * Spoofed IPs: Random source IPs to simulate distributed attacks.
      * Duration: 60 seconds per intensity.

### Sample Command (medium intensity):

hping3 -S -p 80 --rand -source --faster -c 60000 10.0.0.1

1

* + - **Expectation**: Detection in <1 second, blocking in <2 seconds, precision 93%, FPR < 2%.

## Scenario 3: UDP/ICMP Flood Attack

* + - **Description**: Hosts h2 and h3 send UDP packets to h1 to simulate a UDP Flood attack.
    - **Tool**: hping3.

### Parameters:

* + - * Protocol: UDP, port 53.
      * Intensity Levels:

∗ Low: 1000 packets/second per host.

∗ Medium: 2000 packets/second per host.

∗ High: 5000 packets/second per host.

* + - * Spoofed IPs: Random source IPs.
      * Duration: 60 seconds per intensity.

### Sample Command (medium intensity):

hping3 --udp -p 53 --rand -source --faster -c 120000 10.0.0.1

1

* + - **Expectation**: Detection in <1 second, blocking in <2 seconds, precision 93%.

## Scenario 4: HTTP Flood Attack

* + - **Description**: Host h4 sends HTTP GET requests to h1 to simulate an HTTP Flood attack.
    - **Tool**: wrk.

### Parameters:

* + - * Protocol: HTTP, port 80.
      * Intensity Levels:

∗ Low: 200 requests/second.

∗ Medium: 500 requests/second.

∗ High: 1000 requests/second.

* + - * Concurrent Connections: 100 per intensity.
      * Duration: 60 seconds per intensity.

### Sample Command (medium intensity):

wrk -t 4 -c 100 -d 60s -R 500 http ://10.0.0.1

1

* + - **Expectation**: Detection in <1 second, blocking in <2 seconds, precision 93%, FPR < 2%.

## Scenario 5: Slowloris Attack

* + - **Description**: Host h4 uses Slowloris to maintain prolonged HTTP connections with h1.
    - **Tool**: slowloris.

### Parameters:

* + - * Number of Connections:

∗ Low: 50 connections.

∗ Medium: 100 connections.

∗ High: 200 connections.

* + - * Connection Timeout: 10 seconds.
      * Duration: 120 seconds per intensity.

### Sample Command (medium intensity):

slowloris 10.0.0.1 -port 80 -sockets 100 -tcpto 10

1

* + - **Expectation**: Detection in <5 seconds, blocking in <2 seconds, precision 93%, FPR < 3%.

# Simulation Process

The simulation process is structured to ensure repeatability and accuracy in evaluating the DDoS detection module.

### Environment Setup:

* + Start Mininet with the provided script.
  + Install and configure Snort on h1 with the custom rules.
  + Set up Apache2 on h1 and ensure port 80 is accessible.
  + Install Prometheus node exporter on h1 for resource monitoring.

1. **Baseline Traﬀic Generation**: Use h5 to generate normal traﬀic for 300 seconds to establish a baseline.
2. **Attack Simulation**: Execute each attack scenario (SYN Flood, UDP/ICMP Flood, HTTP Flood, Slowloris) at low, medium, and high intensities.

### Data Collection:

* + Capture Snort alerts and logs.
  + Record network traﬀic using tcpdump on h1: tcpdump -i any -w attack.pcap.
  + Collect Prometheus metrics for CPU, RAM, and custom module metrics.

1. **Analysis**: Process the collected data to compute performance metrics (see Section 6).

# Metrics and Evaluation

The evaluation focuses on both detection performance and system eﬀiciency, aligned with the module’s design goals.

## Performance Metrics

* + - **Detection Precision**: Percentage of attacks correctly identified. Target: 93%.
    - **False Positive Rate (FPR)**: Percentage of normal traﬀic incorrectly flagged as an attack. Target: < 3%.
    - **Detection Time**: Time from attack start to alert generation. Targets:
      * SYN Flood, UDP/ICMP Flood, HTTP Flood: <1 second.
      * Slowloris: <5 seconds.
    - **Blocking Time**: Time from detection to mitigation (e.g., IP blocking via iptables). Target: <2 seconds.

## System Eﬀiciency Metrics

* + - **CPU Usage**: Percentage of CPU utilized by Snort. Target: < 5%.
    - **RAM Usage**: Memory consumed by Snort. Target: < 100MB.
    - **Detection Latency**: Internal processing delay for generating alerts. Target: < 10ms.

## Evaluation Results Table

The following table summarizes expected results based on prior testing:

### Attack Type Detection Time Blocking Time FPR

|  |  |  |  |
| --- | --- | --- | --- |
| SYN Flood | <1s | <2s | <2% |
| UDP/ICMP Flood | <1s | <2s | <2% |
| HTTP Flood | <1s | <2s | <2% |
| Slowloris | <5s | <2s | <3% |

Table 1: Expected Performance Metrics for DDoS Detection Module

# Tools and Dependencies

* **Mininet**: Version 2.3.0, for network emulation.
* **Snort**: Version 2.9.20, with custom DDoS detection module.
* **hping3**: For SYN and UDP/ICMP Flood attacks.
* **wrk**: For HTTP Flood attacks.
* **slowloris**: For Slowloris attacks.
* **tcpdump**: For traﬀic capture and analysis.
* **Prometheus**: For resource and custom metric monitoring.
* **Apache2**: Web server on h1 for HTTP-based attacks.