SAKSHI NAIR

FINAL PROJECT REPORT – SNS
Detecting SYN Flood Attacks with Lightweight On-Host Features

- **INTRODUCTION:** SYN flood attacks remain one of the most persistent and impactful Denial-of-Service (DoS) techniques used by attackers to overwhelm systems. By abusing the TCP three-way handshake, an attacker sends a massive volume of SYN packets without completing the connection, causing the victim's connection table to fill and eventually crash or freeze. This project focuses on developing a lightweight, host-based detection mechanism that can identify SYN flood activity by analyzing packet rates and connection patterns at the victim machine itself. The goal is to create a practical detection approach that is simple to implement, resource-efficient, and suitable for small-scale systems.

- **PROJECT OBJECTIVES:** The primary objectives of this project are:-

1. Develop a lightweight, rule-based detection mechanism for SYN flood attacks.
2. Analyze real-time packet behavior and identify abnormal traffic patterns.
3. Implement traffic generation and attack simulation using SEED labs.
4. Capture packet traces to study SYN behavior under normal and attack conditions.
5. Implement a detection script based on SYN rate, SYN/SYN-ACK ratios, and handshake patterns.

- **BACKGROUND:** A SYN flood attack targets the TCP handshake process. Normally, the handshake follows:

SYN → SYN-ACK → ACK

In a SYN flood:

- The attacker sends a large number of **SYN** packets.
- The victim responds with **SYN-ACK**, but the attacker never replies with **ACK**.
- The victim keeps half-open connections in memory, consuming resources.

➔ Important characteristics of SYN flood behavior include:

- Extremely high SYN arrival rates
- Low completion rate for handshakes
- Large discrepancy between SYN and SYN-ACK counts
- Connection table exhaustion

These predictable characteristics enable on-host detection without complex machine learning or external IDS systems.

- **EXPERIMENTAL ENVIRONMENT:** This project uses the official **SEED Labs** environment with a three-machine setup:

  Attacker Machine – Generates SYN flood traffic using tools like `hping3`.

  Victim Machine – Runs detection script and observes impact on TCP queue.

  Monitor/Server Machine – Captures packets using `tcpdump` for analysis.

  Key software used:

1. **Ubuntu 20.04 SEED machines**
2. **tcpdump** for packet capture
3. **hping3** for SYN flood generation
4. **Python / Bash** for detection logic
5. **Wireshark** for packet visualization

This controlled environment ensures accurate and safe experimentation.

- **TRAFFIC GENERATION:** Traffic was generated in two categories-

Normal Traffic –

1. Regular TCP connection attempts
2. Random browsing simulation
3. Low SYN rate (below 20 SYN/sec)
4. Balanced SYN ↔ SYN-ACK ratio

Attack Traffic –

Generated using :-

```
sudo hping3 -S -p 80 --flood <victim-ip>
```

Characteristics:

1. SYN rate exceeds **500–5000 SYN/sec**
2. No ACK replies from attacker
3. Extremely low handshake completion
4. Victim's SYN backlog queue quickly fills

This contrast between normal and attack behavior forms the foundation for detection.

- **<u>PACKET CAPTURE METHOD:</u>**

Packet traces from the victim machine were collected using –

```
sudo tcpdump -nn -i eth0 'tcp[tcpflags] & tcp-syn != 0'
```

<u>Captured data includes -</u>

1. Timestamps
2. Source/Destination IP
3. TCP flags
4. Rate of SYN arrivals

<u>Packet capture allowed verification of-</u>

- SYN packet spikes
- SYN-ACK responses
- Timing patterns
- Half-open connection increase

Traces were later inspected in Wireshark for deeper understanding.

- **<u>FEATURE EXTRACTION:</u>**

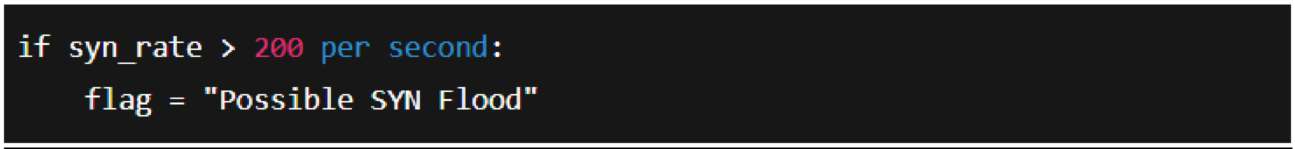
From captured packets, the following features were extracted:

1. SYN rate per second
2. SYN-ACK rate per second
3. Ratio of SYN to SYN-ACK packets
4. Number of incomplete TCP handshakes
5. Unique source IP count (to distinguish spoofing-based floods)
6. Backlog queue usage

These features help distinguish normal traffic from SYN flood behavior.

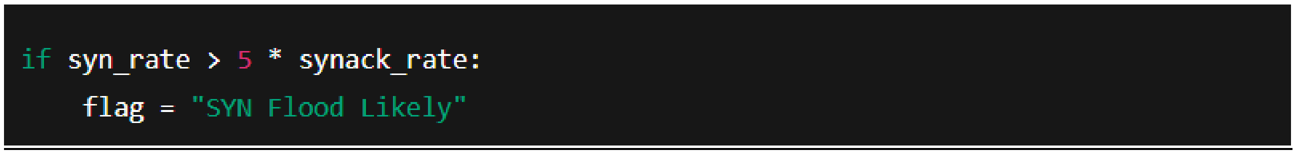
- **DETECTION LOGIC:** A lightweight rule-based detection algorithm was implemented. Below are the simplified detection rules:

Rule 1: SYN Rate Threshold –



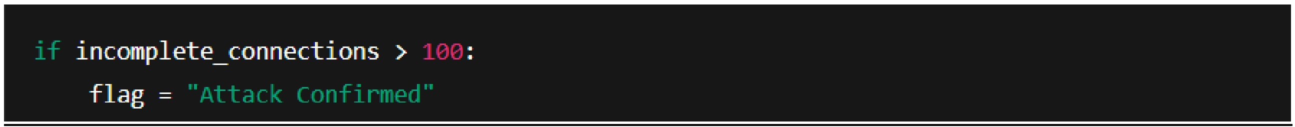
```
if syn_rate > 200 per second:
    flag = "Possible SYN Flood"
```

Rule 2: SYN vs SYN-ACK Imbalance –



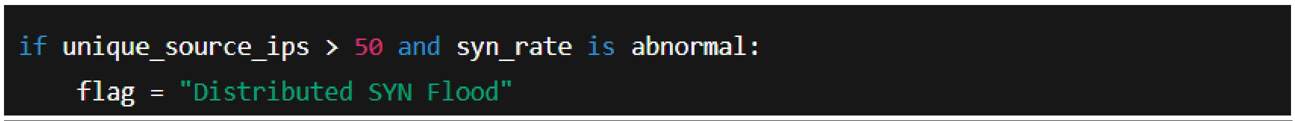
```
if syn_rate > 5 * synack_rate:
    flag = "SYN Flood Likely"
```

Rule 3: Half-Open Connection Count –



```
if incomplete_connections > 100:
    flag = "Attack Confirmed"
```

Rule 4: Multi-Source SYN Indicators –



```
if unique_source_ips > 50 and syn_rate is abnormal:
    flag = "Distributed SYN Flood"
```

- **RESULTS and OBSERVATIONS :** During testing:

1. Under normal traffic, **no rule was triggered**, and SYN/SYN-ACK ratios remained stable.
2. During SYN flood attacks, the **SYN rate reached up to 3000 SYN/sec**.
3. SYN-ACK responses were minimal due to queue exhaustion.
4. The detection algorithm triggered alerts within **1–3 seconds** of attack onset.
5. CPU usage on the victim machine remained low (<10%), showing the approach is lightweight.

The system successfully detected both **single-source** and **distributed** SYN floods.

- **LIMITATIONS :** While the system works effectively, it has some limitations:

1. **Thresholds require manual tuning** based on system performance.
2. **Slow-rate (low and slow) SYN floods** may not trigger detection immediately.
3. Detection is **reactive**, not preventive.
4. Not a full replacement for enterprise intrusion detection systems (IDS).

- **FUTURE IMPROVEMENTS :** Future enhancements can make the system stronger:

1. Integrating **machine learning** for adaptive thresholds
2. Adding **automatic mitigation** (e.g., iptables blocking)
3. Implementing a **real-time dashboard** for visualization
4. Logging attack details to a **centralized server**
5. Supporting IPv6 and multi-interface systems

- **CONCLUSION :** This project successfully demonstrates that a simple, lightweight, host-based mechanism can effectively detect SYN flood attacks using packet-rate analysis and basic TCP handshake behavior. By leveraging metrics like SYN rate, SYN-ACK imbalance, and incomplete connection counts, the system identifies attacks quickly and with minimal computational overhead. The SEED lab environment provided a practical and controlled setting to simulate attacks, analyze behaviors, and implement a detection solution. Although not a complete defense system, the project offers a strong foundation for host-level intrusion detection and can be extended into more advanced detection and mitigation tools.