# ****Network Intrusion Prevention System (IPS) — Weekly Task Report****

**Team:** SkullFaced  
**Duration:** 1 week  
**Date: 20.08.2025**

Name: Ishan Chowdhury

## ****1. Objective****

The goal of this task was to design and implement a lightweight Intrusion Prevention System (IPS) capable of detecting and blocking malicious traffic in real time. Specifically, the IPS should:

Block ICMP ping floods.

Drop repeated TCP SYN floods or half-open connections.

Prevent simple scan patterns (SYN/NULL/FIN scans, repeated port attempts).

Enforce basic application-level rules (e.g., block suspicious HTTP payloads such as SQL injection patterns).

## ****2. IPS Design & Prevention Logic****

Our IPS consists of a Python-based engine (IPSCore) that processes normalized network events (timestamp, IPs, ports, protocol, TCP flags, payload). For offline testing, it runs against PCAP files using **Scapy**.

### ****Implemented Detection Rules****

**ICMP Flood Prevention**

Tracks per-source ICMP echo requests in a 1-second sliding window.

Blocks a source if it exceeds **100 packets per second** (configurable).

**SYN Flood & Half-Open Connection Handling**

Counts pure SYN packets (SYN=1, ACK=0) per source in a 1-second window.

Blocks if a source exceeds **200 SYNs per second**.

Tracks incomplete handshakes (SYN without ACK).

Blocks if backlog exceeds **500 half-open connections** to a destination.

**Port Scan Detection**

Records unique destination ports contacted by a source in a 5-second window.

Blocks if a host probes **20 or more ports** rapidly using suspicious flags (SYN, NULL, FIN).

**Suspicious HTTP Payloads (SQL Injection Signatures)**

Inspects up to 2KB of TCP payload for SQLi patterns.

Blocks if regex detects common malicious strings (e.g., ’ OR 1=1--, UNION SELECT, sleep(), benchmark(), etc.).

## ****3. False Positive Handling****

**Threshold Tuning:** All rule thresholds (ICMP rate, SYN rate, backlog size, scan ports) are configurable via CLI flags or rules.py.

**Conservative Signatures:** SQLi regex looks for strong attack indicators only, not generic SQL keywords.

**Tested on Normal Traffic:** Ran against a normal browsing PCAP; no false blocks were observed.

**Future Enhancements:** Add IP allow-lists for monitoring tools and an “alert-only” mode to further reduce false positives.

## ****4. Demo Results****

We ran the IPS against two PCAPs:

**Normal PCAP (benign traffic):**

Total: ~120 packets

Blocked: 0

Allowed: 120

**Malicious PCAP (generated attacks: ICMP flood, SYN scan, SQLi HTTP request):**

Total: ~350 packets

Blocked: 27

ICMP Flood: 10

SYN Flood: 12

Port Scan: 3

SQLi Payloads: 2

## ****5. Unit / Integration Testing****

Unit tests validated:

ICMP flood detection logic

SYN flood detection

Port scan rule triggering

SQL injection regex match on payloads

**Command to run tests:**

pytest -q

All tests passed, confirming the correctness of core prevention logics

## ****6. Limitations & Future Improvements****

**Kernel Enforcement:** Current implementation logs block decisions but does not drop packets. Integration with iptables, nftables, or eBPF would enable real blocking.

**Performance:** Python is not optimal for high-throughput traffic. A compiled or eBPF-based IPS would handle production load better.

**Protocol Support:** Extend beyond ICMP/TCP/UDP to cover TLS parsing, application-layer signatures, and evasion resistance.

**Telemetry:** Add centralized alerting and dashboards for visibility.

## ****7. Conclusion****

The implemented IPS successfully detects and blocks ICMP floods, SYN floods, port scans, and simple HTTP-based exploits such as SQL injection. With tunable thresholds and clear prevention logic, it provides a strong foundation for real-time intrusion prevention and can be extended into a production-ready system with kernel integration and performance optimizations.