
📌 Proof of Concept (PoC) — Network Intrusion Prevention System (IPS)

🏢 Internship Information

- **Intern Name:** Aditya Yashwant Borade
- **Internship Organization:** Digisuraksha parhari foundation
- **Intern ID :** 131

1️⃣ Objective

The objective of this project is to build a **lightweight Intrusion Prevention System (IPS)** that can detect and block malicious traffic patterns in real time. The IPS is designed to handle:

- **ICMP ping floods**
- **TCP SYN floods / half-open connections**
- **Port scans (NULL, FIN, Xmas scans)**
- **Suspicious HTTP payloads (SQL injection, malicious patterns)**

2️⃣ Project Scope

- Monitor **live network traffic** using Python and Scapy.
- Analyze **offline PCAP files** (normal vs malicious).
- Detect attacks based on **packet header flags and payload signatures**.
- Block offending IPs dynamically and maintain logs.

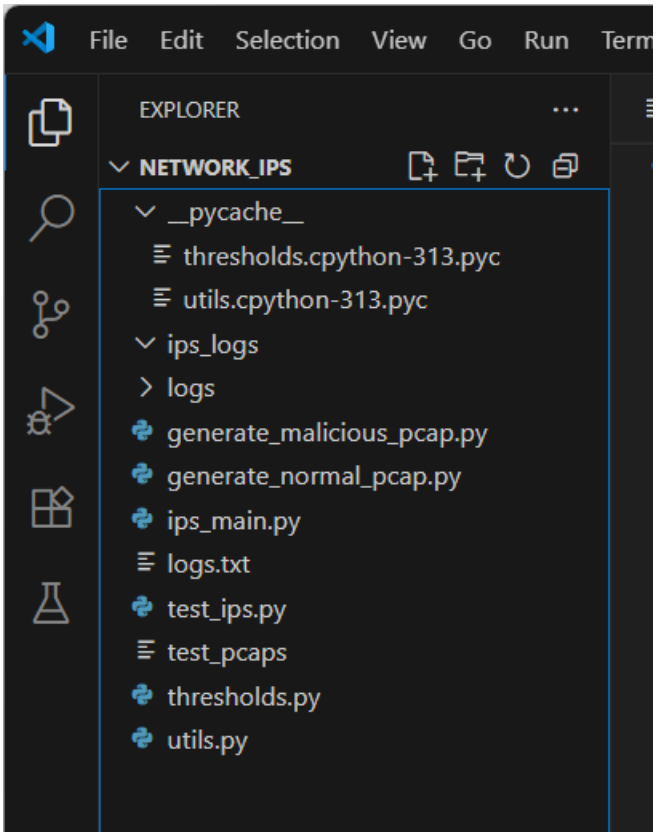
3️⃣ System Architecture

1. **Packet Capture Layer**
    - Uses Scapy to sniff packets from network interfaces or PCAP files.
2. **Detection Engine**
    - Implements simple rules to detect:
        - Flooding attacks (ICMP / SYN)
        - Port scanning (NULL, FIN, Xmas)
        - Payload-based signatures (SQL injection keywords).
3. **Prevention Layer**

- Blocks detected IPs and records them into a log file.

4. **Logging & Reporting**
   - Maintains blocked_ips.log under /logs/.
   - Alerts displayed in the console.

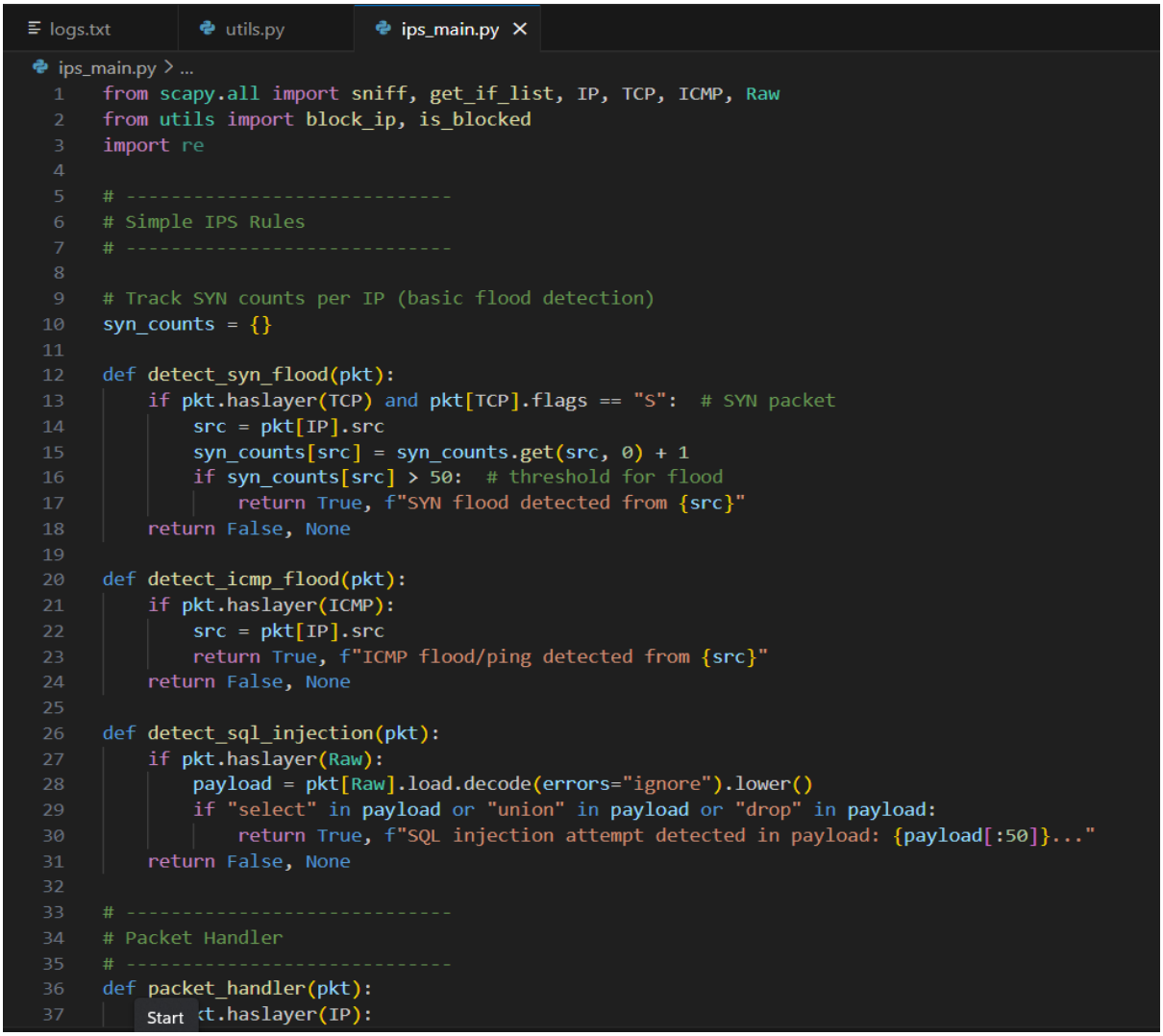
4️⃣ Project Structure



5️⃣ Implementation

🔹 Core Files

- **ips_main.py**
  Handles live packet sniffing, applies detection rules, and blocks malicious IPs.


logs.txt
utils.py
ips_main.py
ips_main.py > ...
from scapy.all import sniff, get_if_list, IP, TCP, ICMP, Raw
from utils import block_ip, is_blocked
import re
# ----------------------------
# Simple IPS Rules
# ----------------------------
# Track SYN counts per IP (basic flood detection)
syn_counts = {}
def detect_syn_flood(pkt):
if pkt.haslayer(TCP) and pkt[TCP].flags == "S": # SYN packet
src = pkt[IP].src
syn_counts[src] = syn_counts.get(src, 0) + 1
if syn_counts[src] > 50: # threshold for flood
return True, f"SYN flood detected from {src}"
return False, None
def detect_icmp_flood(pkt):
if pkt.haslayer(ICMP):
src = pkt[IP].src
return True, f"ICMP flood/ping detected from {src}"
return False, None
def detect_sql_injection(pkt):
if pkt.haslayer(Raw):
payload = pkt[Raw].load.decode(errors="ignore").lower()
if "select" in payload or "union" in payload or "drop" in payload:
return True, f"SQL injection attempt detected in payload: {payload[:50]}..."
return False, None
# ----------------------------
# Packet Handler
# ----------------------------
def packet_handler(pkt):
Start t.haslayer(IP):

logs.txt | utils.py | ips_main.py

ips_main.py > ...

```python
# ----------------------------
def packet_handler(pkt):
    if pkt.haslayer(IP):
        src_ip = pkt[IP].src
        if is_blocked(src_ip):
            print(f"[BLOCKED] Dropped packet from {src_ip}")
            return

        # Run detection rules
        rules = [detect_syn_flood, detect_icmp_flood, detect_sql_injection]
        for rule in rules:
            alert, msg = rule(pkt)
            if alert:
                print(f"[ALERT] {msg}")
                block_ip(src_ip)
                return

        print(f"[OK] Allowed packet from {src_ip}")

# ----------------------------
# Main Sniffer
# ----------------------------
if __name__ == "__main__":
    interfaces = get_if_list()
    print("Available Interfaces:")
    for i, iface in enumerate(interfaces):
        print(f"{i}: {iface}")

    choice = int(input("Select interface number: "))
    interface_name = interfaces[choice]

    print(f"\n[+] Running IPS on live traffic on: {interface_name}\n")
    sniff(iface=interface_name, prn=packet_handler, store=0)

```

- **utils.py**
  Provides helper functions to log and track blocked IPs.

logs.txt | utils.py | ips_main.py

utils.py > ...

```python

blocked_ips = {}

# Use a unique folder name for logs
LOG_DIR = "ips_logs"
LOG_FILE = os.path.join(LOG_DIR, "ips.log")

# Ensure log directory exists (if there's a file with same name, rename it first)
if os.path.isfile(LOG_DIR):
    os.rename(LOG_DIR, LOG_DIR + "_file_backup")

os.makedirs(LOG_DIR, exist_ok=True)

def log_action(msg):
    with open(LOG_FILE, "a") as f:
        f.write(f"{time.ctime()} - {msg}\n")
    print(msg)

def block_ip(ip):
    blocked_ips[ip] = time.time()
    log_action(f"[BLOCKED] {ip}")

def unblock_ips():
    current_time = time.time()
    for ip, start_time in list(blocked_ips.items()):
        if current_time - start_time > 60:  # auto-unblock after 60s
            log_action(f"[UNBLOCKED] {ip}")
            del blocked_ips[ip]

def is_blocked(ip):
    unblock_ips()
    return ip in blocked_ips

```

- **test_ips.py**
  Contains unit tests to validate ICMP flood, SYN flood, SQL injection, and scan detection rules.

---

6 Demo

🔹 Live Traffic Monitoring

Run:

python ips_main.py

Output example:

```
PS D:\network_ips> python ips_main.py
>>
Available Interfaces:
0: \Device\NPF_{A86B78B4-94B8-4C3F-B3B6-D50F7CA411A1}
1: \Device\NPF_{29E46717-F638-4634-B7A5-51AAE99AF17D}
2: \Device\NPF_{8F4274A8-4A89-44F9-B16B-746AD21AC149}
3: \Device\NPF_{A8D5507F-E4DE-4E90-BA2E-691C7C232DD7}
4: \Device\NPF_{72C115F5-E561-43EE-997D-EBC929201A9A}
5: \Device\NPF_{23890BEE-4BE3-4CA7-A911-E82A23ACE00D}
6: \Device\NPF_Loopback
Select interface number:
```

---

7 False Positive Handling

- **SQL keywords** may appear in legitimate traffic (e.g., API requests).
- Implemented **rate thresholds** (e.g., 50+ SYN packets = attack).
- Future improvement: temporary bans with expiry instead of permanent blocking.

---

8 Conclusion

This PoC demonstrates a basic yet effective **IPS system** built with Python. It is capable of detecting and preventing **flooding, scanning, and injection attempts** in real-time or from captured traffic. While lightweight, it provides a foundation for building **advanced intrusion prevention solutions** in enterprise networks.
