Intrusion Prevention System (IPS) Tool

Name: Prateek(166), Dev Sharma(179)

1. Introduction

The Intrusion Prevention System (IPS) tool presented in this report is a Python-based security solution designed to demonstrate the detection and prevention of common network threats. It integrates packet generation, real-time traffic analysis, visualization, and attack simulation to provide a comprehensive proof-of-concept system.

Unlike production-grade IPS solutions, this tool is developed primarily for educational, research, and demonstration purposes. It allows users to explore how network attacks such as ICMP floods, SYN floods, and malicious payload injections can be detected and mitigated.

The tool consists of three main components:

- **Packet Generation Module** for creating benign and malicious PCAP files.
- **Graphical User Interface (GUI)** for monitoring, logging, and visualizing suspicious activity.
- **Attack Simulation Module** for generating real-time attacks against localhost to validate detection.

2. Objectives

- To demonstrate common attack patterns such as ICMP floods, SYN floods, and malicious payloads.
- To provide a graphical monitoring interface for alerts and threat analysis.
- To allow testing both with pre-generated packet captures and real-time traffic.
- To serve as a proof-of-concept (PoC) system for intrusion detection and prevention.

3. Requirements

The following dependencies are required to run the IPS tool (as defined in requirements.txt):

- PyQt5 (>=5.15.10): GUI framework for building the dashboard.
- PyQtChart (>=5.15.6): Optional chart rendering support.
- scapy (>=2.5.0): Packet crafting, sniffing, and manipulation.
- matplotlib (>=3.7.2): Visualization library for charts.
- System Dependency: libpcap-dev (Linux only, required for live sniffing).

4. Components

4.1 generate_pcaps.py

Generates packet capture (PCAP) files for controlled testing:
- normal.pcap: Contains benign traffic (TCP handshakes and HTTP requests).
- malicious.pcap: Contains malicious patterns such as ICMP floods, SYN scans, NULL/FIN/Xmas flag scans, and HTTP requests with SQL injection (SQLi), XSS, and LFI payloads.

4.2 ips_gui.py

Implements the main IPS Graphical User Interface (GUI):
- Provides real-time monitoring of network traffic.
- Detects ICMP floods, SYN floods, and suspicious payloads.
- Displays alerts using line, pie, and bar charts.
- Maintains threat logs, blocklists, and packet analysis reports.
- Supports both PCAP file analysis and live interface monitoring.

4.3 lo_attacker.py

Simulates attack traffic against localhost (127.0.0.1):
- ICMP flood attack.
- TCP SYN flood attack.
- Malicious payload delivery with SQL injection, XSS, and database manipulation strings.

Used to test live detection and prevention mechanisms.

5. Usage Instructions

1. Setup Virtual Environment:

```
  python3 -m venv venv
  source venv/bin/activate   (Linux/macOS)
  venv\Scripts\Activate    (Windows)
```

2. Install Dependencies:

```
  pip install -r requirements.txt
```

3. Generate PCAP Files:

```
  python generate_pcaps.py
```

4. Launch the IPS GUI:

```
  python ips_gui.py
```

5. (Optional) Run Local Attacker Simulation:

```
  python lo_attacker.py
```

6. Demonstration & Proof of Concept

This section is reserved for inserting screenshots of the IPS in action. These may include:
- GUI dashboard with active alerts.
- Charts showing attack distribution.
- Threat logs and blocklist updates.
- Packet analysis reports.

[Insert screenshots/figures here]

7. Limitations

- The IPS is not optimized for production-grade deployments.
- Detection relies on signatures and thresholds, limiting ability to detect zero-day attacks.
- Performance is constrained to small-scale/local environments.
- Live monitoring may require elevated privileges.

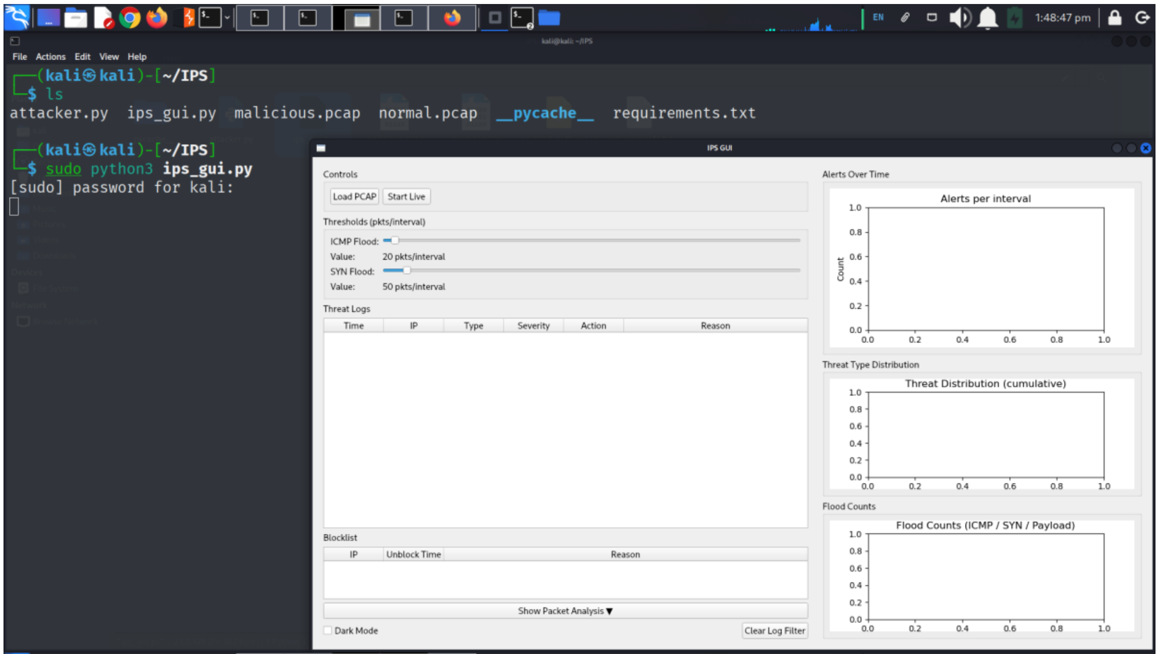
8. Future Enhancements

- Integration with anomaly detection or machine learning models.
- Enhanced visualization and customizable dashboards.
- Distributed monitoring across multiple interfaces or systems.
- Automated reporting and alert notification features.

9. Conclusion

The IPS tool successfully demonstrates how Python can be applied to network security monitoring and intrusion prevention. It integrates packet analysis, visualization, and simulated attack scenarios to highlight detection techniques. Its modular structure allows further experimentation and expansion for research and learning purposes.

10. PoC



```
┌──(kali㉿kali)-[~/IPS]
└─$ ls
attacker.py  ips_gui.py  malicious.pcap  normal.pcap  __pycache__  requirements.txt

┌──(kali㉿kali)-[~/IPS]
└─$ sudo python3 ips_gui.py
[sudo] password for kali:
```

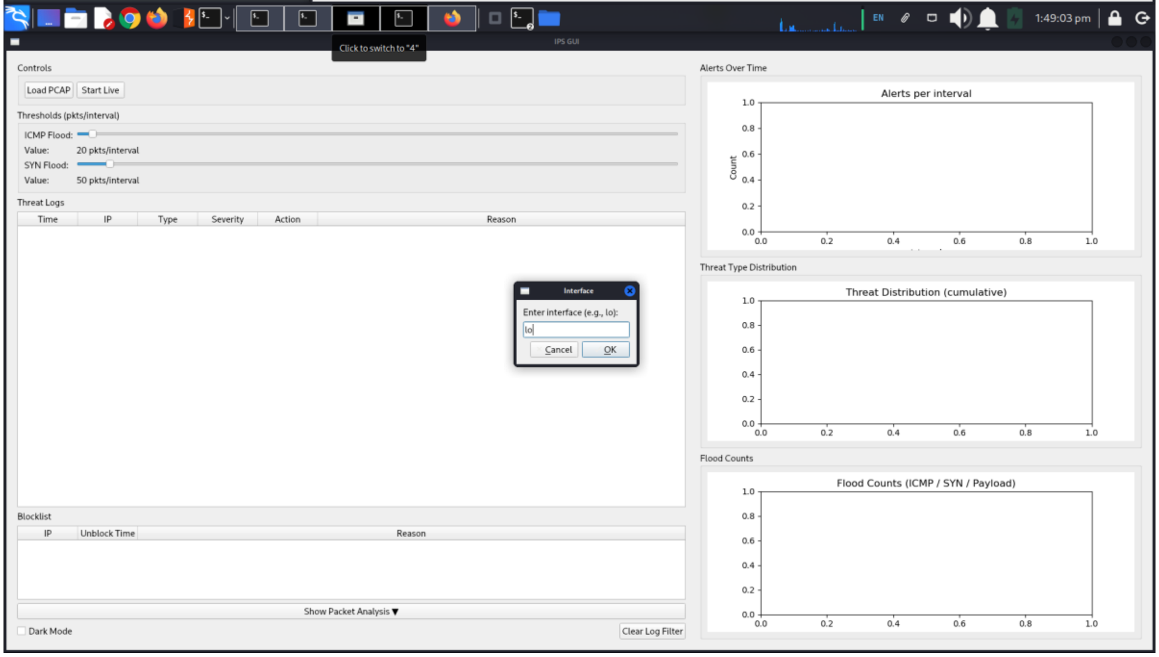
Value: 20 pkts/interval

Value: 50 pkts/interval

Time	IP	Type	Severity	Action	Reason

IP	Unblock Time	Reason

□ Dark Mode



Value: 20 pkts/interval

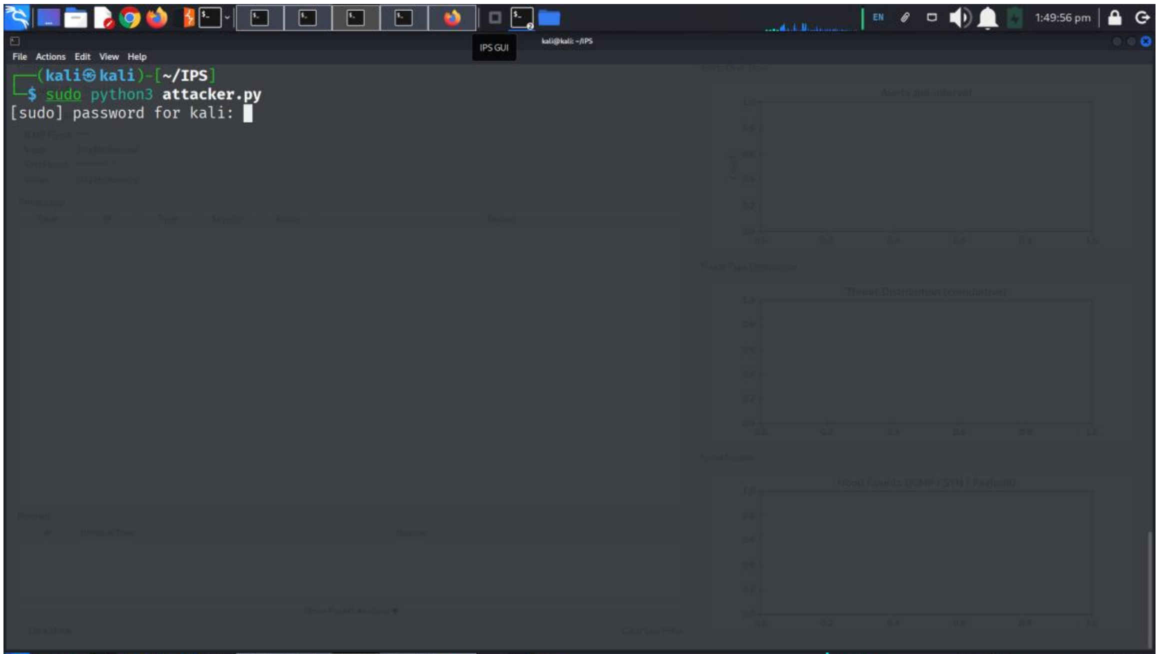
Value: 50 pkts/interval

Time	IP	Type	Severity	Action	Reason

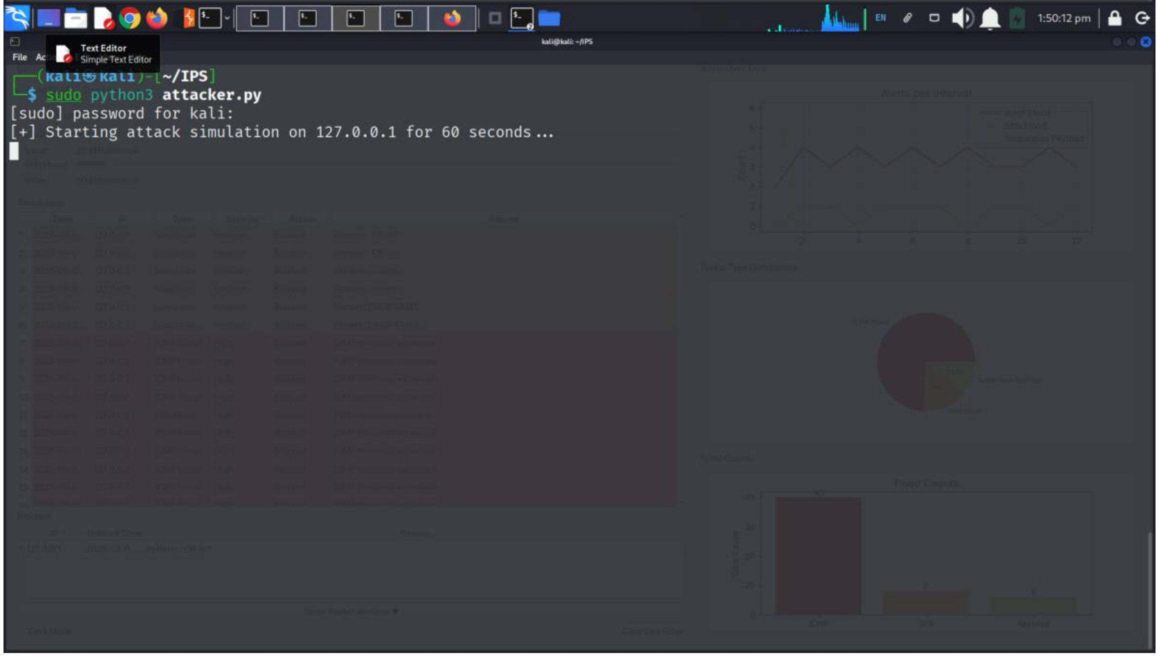
IP	Unblock Time	Reason

Enter interface (e.g., lo): lo

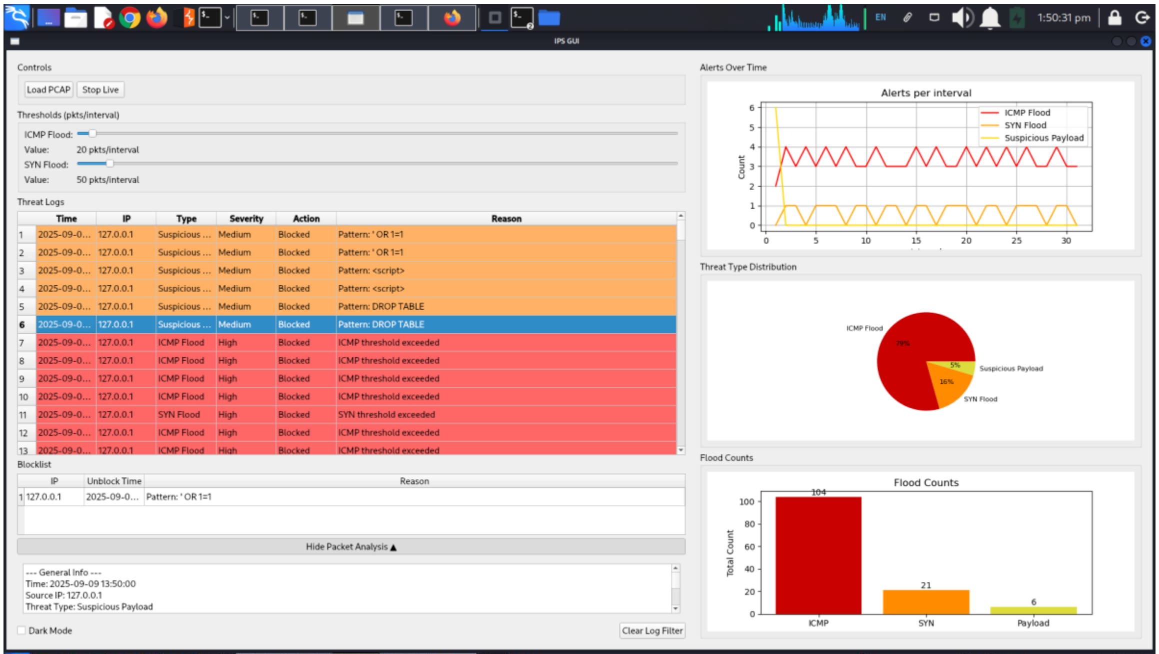
□ Dark Mode



```
┌──(kali㉿kali)-[~/IPS]
└─$ sudo python3 attacker.py
[sudo] password for kali:
```



```
┌──(kali㉿kali)-[~/IPS]
└─$ sudo python3 attacker.py
[sudo] password for kali:
[+] Starting attack simulation on 127.0.0.1 for 60 seconds...
```



Thresholds (pkts/interval)

ICMP Flood:
Value: 20 pkts/interval
SYN Flood:
Value: 50 pkts/interval

Threat Logs

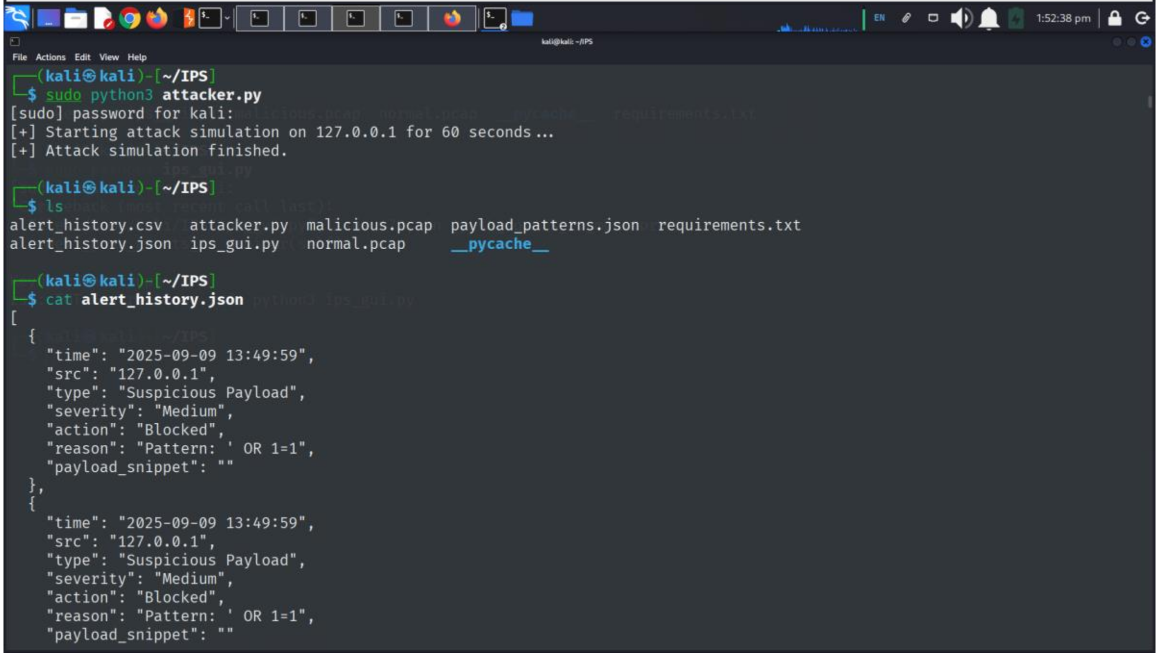
	Time	IP	Type	Severity	Action	Reason
1	2025-09-0...	127.0.0.1	Suspicious ...	Medium	Blocked	Pattern: ' OR 1=1
2	2025-09-0...	127.0.0.1	Suspicious ...	Medium	Blocked	Pattern: ' OR 1=1
3	2025-09-0...	127.0.0.1	Suspicious ...	Medium	Blocked	Pattern: <script>
4	2025-09-0...	127.0.0.1	Suspicious ...	Medium	Blocked	Pattern: <script>
5	2025-09-0...	127.0.0.1	Suspicious ...	Medium	Blocked	Pattern: DROP TABLE
6	2025-09-0...	127.0.0.1	Suspicious ...	Medium	Blocked	Pattern: DROP TABLE
7	2025-09-0...	127.0.0.1	ICMP Flood	High	Blocked	ICMP threshold exceeded
8	2025-09-0...	127.0.0.1	ICMP Flood	High	Blocked	ICMP threshold exceeded
9	2025-09-0...	127.0.0.1	ICMP Flood	High	Blocked	ICMP threshold exceeded
10	2025-09-0...	127.0.0.1	ICMP Flood	High	Blocked	ICMP threshold exceeded
11	2025-09-0...	127.0.0.1	SYN Flood	High	Blocked	SYN threshold exceeded
12	2025-09-0...	127.0.0.1	ICMP Flood	High	Blocked	ICMP threshold exceeded
13	2025-09-0...	127.0.0.1	ICMP Flood	High	Blocked	ICMP threshold exceeded

Blocklist

	IP	Unblock Time	Reason
1	127.0.0.1	2025-09-0...	Pattern: ' OR 1=1

```
--- General Info ---
Time: 2025-09-09 13:50:00
Source IP: 127.0.0.1
Threat Type: Suspicious Payload
```

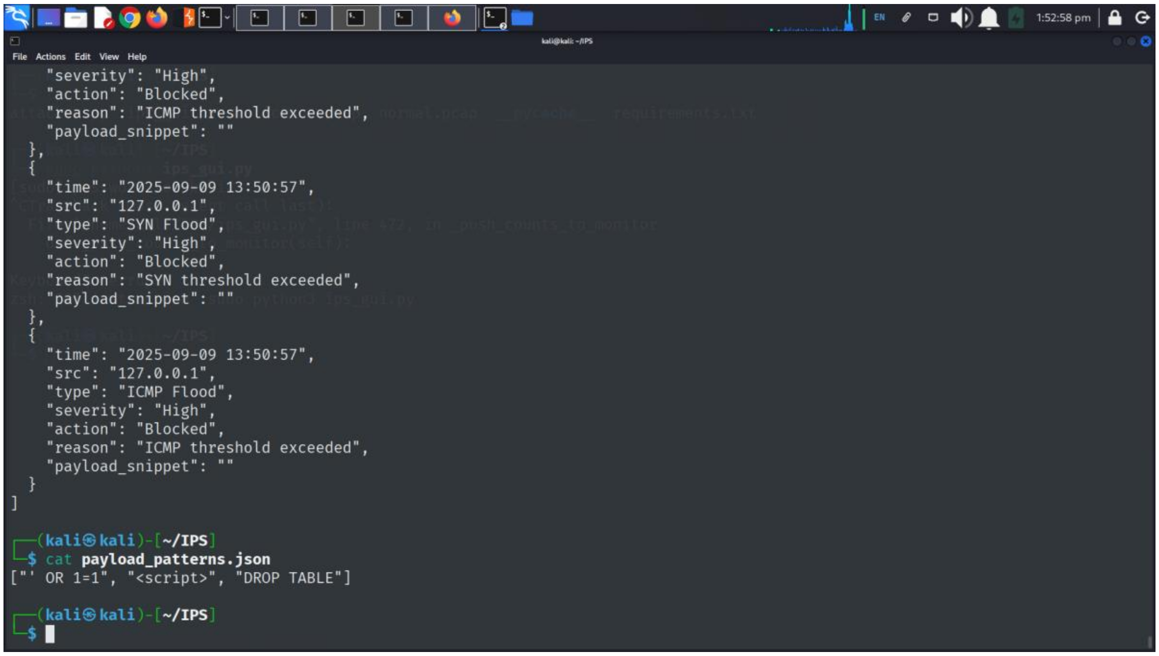
□ Dark Mode



```
┌──(kali㉿kali)-[~/IPS]
└─$ sudo python3 attacker.py
[sudo] password for kali:
[+] Starting attack simulation on 127.0.0.1 for 60 seconds...
[+] Attack simulation finished.

┌──(kali㉿kali)-[~/IPS]
└─$ ls
alert_history.csv   attacker.py  malicious.pcap  payload_patterns.json  requirements.txt
alert_history.json  ips_gui.py   normal.pcap     __pycache__

┌──(kali㉿kali)-[~/IPS]
└─$ cat alert_history.json
[
  {
    "time": "2025-09-09 13:49:59",
    "src": "127.0.0.1",
    "type": "Suspicious Payload",
    "severity": "Medium",
    "action": "Blocked",
    "reason": "Pattern: ' OR 1=1",
    "payload_snippet": ""
  },
  {
    "time": "2025-09-09 13:49:59",
    "src": "127.0.0.1",
    "type": "Suspicious Payload",
    "severity": "Medium",
    "action": "Blocked",
    "reason": "Pattern: ' OR 1=1",
    "payload_snippet": ""
```



```
    "severity": "High",
    "action": "Blocked",
    "reason": "ICMP threshold exceeded",
    "payload_snippet": ""
  },
  {
    "time": "2025-09-09 13:50:57",
    "src": "127.0.0.1",
    "type": "SYN Flood",
    "severity": "High",
    "action": "Blocked",
    "reason": "SYN threshold exceeded",
    "payload_snippet": ""
  },
  {
    "time": "2025-09-09 13:50:57",
    "src": "127.0.0.1",
    "type": "ICMP Flood",
    "severity": "High",
    "action": "Blocked",
    "reason": "ICMP threshold exceeded",
    "payload_snippet": ""
  }
]

┌──(kali㉿kali)-[~/IPS]
└─$ cat payload_patterns.json
["' OR 1=1", "<script>", "DROP TABLE"]

┌──(kali㉿kali)-[~/IPS]
└─$ 
```