Enhanced Network Security Monitor - Packet Analyzer Report

The provided packet analyzer script has been thoroughly reviewed, and several enhancements have been implemented to boost its features, improve its security detection capabilities, and fulfill the request to extract raw packet data for flagged insecure packets.

1. Summary of Enhancements

The original script was a good foundation for a basic network traffic analyzer. The enhancements focus on three main areas: **Security Feature Boost**, **Code Robustness**, and **Forensic Capability**.

Feature Area	Original Weakness/Feature	Enhancement Implemented
Forensic Capability	No raw data extraction for alerts.	**Raw Packet Data Extraction:** The `flag_alert` method now saves the full raw packet data and a human-readable hex dump to a dedicated directory (`insecure_packet_data/`) when an alert is triggered.
Security Detection	Basic, easily bypassed checks for port scan and SYN flood.	**Improved Security Checks:** Added `alerted_ips` set to prevent repeated alerts for the same IP, improved port scan logic, and introduced new checks for **Ping Flood** and basic **ARP Spoofing**.
HTTP Analysis	Limited attack pattern matching.	**Expanded Attack Signatures:** The HTTP analysis now includes a broader set of regular expressions to detect common web attacks like SQL Injection and Cross-Site Scripting (XSS).
Code Robustness	Lack of configuration and error handling.	**Configuration and Utility:** Added configuration constants for thresholds (`PORT_SCAN_THRESHOLD`,

		`SYN_FLOOD_THRESHOLD`), ensured the raw data directory is created, and improved argument handling for interface selection.

2. Detailed Implementation of Raw Data Extraction

The core requirement was to extract the raw data of packets flagged as insecure. This was achieved by modifying the `flag_alert` method and introducing a new method, `extract_raw_data`.

1. **`extract_raw_data(self, packet, alert_type)`:**
   - Takes the Scapy `packet` object and the `alert_type` (e.g., "HIGH", "MEDIUM").
   - Uses `bytes(packet)` to get the full raw byte string of the packet.
   - Saves the raw bytes to a file with a unique, timestamped name (e.g., `20251125_133406_HIGH_17.raw`).
   - Additionally, it creates a companion `.hex` file containing a human-readable hex dump and metadata (source/destination IP, alert type) for quick inspection without external tools.
2. **`flag_alert(self, message, severity, packet=None)`:**
   - The method signature was updated to accept an optional `packet` argument.
   - If a `packet` is provided, it calls `self.extract_raw_data()` and includes the resulting filename in the console output and the internal `self.alerts` record.

This forensic capability ensures that analysts have the complete, unaltered packet data for any suspicious activity, which is crucial for post-incident analysis and evidence gathering.

3. Enhanced Security Detection Methods

The original script's security checks were enhanced for better real-world performance:

Port Scan Detection

The logic for port scan detection was retained but is now more configurable using the `PORT_SCAN_THRESHOLD` constant. More importantly, a new `self.alerted_ips` set was introduced to prevent the script from generating a flood of identical alerts for the same IP address within a short monitoring window, improving the signal-to-noise ratio.

SYN Flood Detection

Similar to port scanning, the SYN flood detection now uses configurable constants (`SYN_FLOOD_THRESHOLD` and `SYN_FLOOD_WINDOW`) for better tuning. The `alerted_ips` mechanism also applies here to suppress redundant alerts.

New Detection: Ping Flood

A basic rate-limiting check was added within the `analyze_packet` method to detect an excessive rate of ICMP echo requests from a single source IP, which is a common indicator of a Denial-of-Service (DoS) attack known as a **Ping Flood**.

New Detection: Basic ARP Spoofing

A rudimentary check for ARP spoofing was added to the ARP analysis section. It flags an alert if an ARP reply packet has a mismatch between the MAC address in the Ethernet frame header and the MAC address in the ARP payload, a common sign of an attacker attempting to poison the ARP cache.

4. Usage and Execution

The enhanced script, named `enhanced_packet_analyzer.py` , requires root privileges to capture network traffic.

Prerequisites

The script requires the `scapy` library, which was installed during the enhancement process.

Bash

```
sudo pip3 install scapy
```

Execution

The script can be run directly. It will automatically select the most active network interface.

Bash

```
sudo python3 enhanced_packet_analyzer.py
```

To explicitly specify an interface (e.g., for testing on the loopback interface `lo`):

Bash

```
sudo python3 enhanced_packet_analyzer.py lo
```

Raw Data Output

All raw packet data and hex dumps for flagged packets will be saved in the `insecure_packet_data` directory, which is created automatically in the same location where the script is executed.

The enhanced script is attached as `enhanced_packet_analyzer.py`, and a ZIP archive of sample raw data files generated during testing is attached as `insecure_packet_data.zip`.

References

1. Scapy Documentation - Official documentation for the Scapy packet manipulation tool.
2. OWASP Top 10 - Reference for common web application security risks, including SQL Injection and XSS, which informed the attack signature expansion.
3. Network Security Monitoring - General principles of network security monitoring and intrusion detection.