Detailed Explanation of Methods in `enhanced_packet_analyzer.py`

The enhanced script is structured around the `ScapyTrafficAnalyzer` class, which handles the core logic for packet processing, security detection, and reporting. The script also includes utility functions for interface selection and the main execution function.

I. `ScapyTrafficAnalyzer` Class Methods

This class is the heart of the analyzer, managing state, processing packets, and detecting threats.

1. `__init__(self)`

- **Purpose:** Initializes the analyzer's state variables and data structures.
- **Details:**
  - Sets up `defaultdict(int)` for general statistics (`self.stats`).
  - Initializes lists and sets for tracking alerts, suspicious domains, and HTTP requests.
  - Initializes security detection state variables: `self.syn_count` (for SYN flood), `self.port_scan_tracker` (for port scan), and `self.alerted_ips` (to prevent repeated alerts for the same IP).
  - Calculates the initial `self.start_time` for duration tracking.
  - **New Feature:** Ensures the `insecure_packet_data` directory exists to store raw packet data.

2. `analyze_packet(self, packet)`

- **Purpose:** The main callback function for Scapy's `sniff` function, executed for every captured packet.
- **Details:**
  - Increments `total_packets` and updates `last_packet_time` .
  - Checks for the presence of the **IP** layer to extract source (`ip_src`) and destination (`ip_dst`) addresses.
  - **Conditional Analysis:** Based on the presence of the **TCP**, **UDP**, **ICMP**, or **ARP** layers, it calls the relevant specialized analysis and detection methods.
  - For TCP packets, it calls `detect_port_scan` , `detect_syn_flood` , `detect_suspicious_ports` , and `analyze_http` .

- For UDP packets, it calls `detect_suspicious_ports` and `analyze_dns`.
- For ICMP packets, it calls the new `detect_ping_flood`.
- For ARP packets, it calls the new `detect_arp_spoofing`.
- Prints a statistics summary every 50 packets.
- Includes a general `try...except` block to gracefully handle errors during packet processing and increment `error_packets`.

3. `extract_raw_data(self, packet, alert_type )`

- **Purpose: (New Feature)** Extracts the full raw data of a packet and saves it to a file for forensic analysis.
- **Details:**
  - Converts the Scapy `packet` object into raw bytes using `bytes(packet)`.
  - Generates a unique filename based on timestamp, alert severity (`alert_type`), and packet number.
  - Saves the raw bytes to a `.raw` file.
  - Saves a human-readable hex dump of the packet, along with metadata (alert type, packet number, IP addresses), to a companion `.hex` file.
  - Returns the filename of the saved raw data file.

4. `analyze_dns(self, packet)`

- **Purpose:** Analyzes DNS queries for suspicious activity, such as Domain Generation Algorithm (DGA) domains.
- **Details:**
  - Checks for the presence of the **DNS** layer and confirms it is a query (`qr == 0`).
  - **Filtering:** Skips mDNS queries (ending in `.local`) to focus on external traffic.
  - **DGA Detection:** Implements a simple heuristic check: if the first part of the domain name is very long (e.g., > 15 characters) and has high character entropy (e.g., > 10 unique characters), it is flagged as a potential DGA domain.
  - Calls `flag_alert` with the packet if a suspicious domain is detected.

5. `analyze_http(self, packet )`

- **Purpose:** Analyzes HTTP traffic for common web attacks and extracts host/user-agent information.
- **Details:**

- Checks for the **TCP** and **Raw** layers on standard HTTP ports (80, 8080).
- Decodes the raw payload.
- **Information Extraction:** Uses regular expressions to extract the `Host` and `User-Agent` headers for statistical tracking.
- **Web Attack Detection:** Uses an expanded set of regular expressions (`select.+from|union.+select|--|drop|sleep\(|alert\(|onload=|onmouseover=|script>`) to detect common SQL Injection and Cross-Site Scripting (XSS) payloads in the request.
- Calls `flag_alert` with the packet if a web attack is detected.

6. `detect_port_scan(self, src_ip, dst_port)`

- **Purpose:** Detects a potential port scan by tracking the number of unique destination ports an IP address attempts to connect to.
- **Details:**
  - Uses `self.port_scan_tracker` to store a set of unique ports scanned by each `src_ip` .
  - **Threshold:** If the number of unique ports exceeds `PORT_SCAN_THRESHOLD` (default 15), it flags a **HIGH** severity alert.
  - **Alert Suppression:** Uses `self.alerted_ips` to prevent continuous alerts from the same IP address.

7. `detect_syn_flood(self, packet, src_ip)`

- **Purpose:** Detects a SYN flood attack by tracking the rate of SYN packets from a single source IP.
- **Details:**
  - Checks if the TCP flag is set to **SYN** (`tcp.flags == "S"`).
  - Increments the SYN count for the `src_ip` .
  - **Rate Limiting:** Resets the counter every `SYN_FLOOD_WINDOW` (default 10 seconds).
  - **Threshold:** If the SYN count exceeds `SYN_FLOOD_THRESHOLD` (default 20) within the window, it flags a **HIGH** severity alert.
  - **Alert Suppression:** Uses `self.alerted_ips` to prevent continuous alerts.

8. `detect_suspicious_ports(self, dst_port, src_ip, protocol, packet)`

- **Purpose:** Detects traffic destined for or originating from known suspicious ports often associated with malware, backdoors, or unencrypted services.
- **Details:**

- Maintains a dictionary of suspicious ports (e.g., 4444 for Metasploit, 23 for Telnet, 445 for SMB) and their associated threat description.
- If the `dst_port` matches a suspicious port, it flags a **MEDIUM** severity alert.
- Calls `flag_alert` with the packet to ensure raw data is saved.

9. `detect_ping_flood(self, src_ip, packet)`

- **Purpose: (New Feature)** Detects a basic Ping Flood attack by monitoring the rate of ICMP requests.
- **Details:**
  - Tracks the count of ICMP requests per source IP in `self.stats`.
  - **Rate Limiting:** Uses a simple 5-second window for checking the rate.
  - **Threshold:** If an IP sends more than 50 ICMP requests in the 5-second window, it flags a **MEDIUM** severity alert.
  - The time reference for this check is reset every 5 seconds.

10. `detect_arp_spoofing(self, packet)`

- **Purpose: (New Feature)** Performs a basic check for ARP spoofing by looking for inconsistencies in ARP reply packets.
- **Details:**
  - Checks for an ARP reply (`packet.op == 2`).
  - Flags a **HIGH** severity alert if the source MAC address in the Ethernet frame (`packet[Ether].src`) does not match the sender hardware address in the ARP payload (`packet[ARP].hwsrc`), which is a common sign of a malicious or misconfigured ARP reply.

11. `flag_alert(self, message, severity, packet=None)`

- **Purpose:** Records a security alert, prints it to the console, and triggers raw data extraction if a packet is provided.
- **Details:**
  - **Raw Data Trigger:** If `packet` is not `None`, it calls `self.extract_raw_data()` and stores the resulting filename.
  - Records the alert details (timestamp, message, severity, raw data file path) in `self.alerts`.
  - Prints the alert to the console with color-coded emojis and includes the raw data filename for easy reference.

12. `print_stats(self)`

- **Purpose:** Prints a comprehensive summary report of the traffic analysis and security alerts.
- **Details:**
  - Calculates and displays general statistics (duration, total packets, packet rate, error packets, protocol counts).
  - Lists the top 5 most frequent DNS queries.
  - Lists the top 5 most frequent HTTP hosts.
  - Lists the 5 most recent security alerts, including the filename of the extracted raw data if available.

II. Utility and Main Functions

1. `find_active_interface()`

- **Purpose:** Automatically scans available network interfaces and selects the one with the most traffic for sniffing.
- **Details:**
  - Uses Scapy's `get_if_list()` to get a list of interfaces.
  - Iterates through non-loopback interfaces, performing a brief `sniff(count=3, timeout=2)` to count packets.
  - Selects the interface that captured the most packets.
  - Provides fallbacks to the `'any'` interface or the first available interface if no traffic is detected.

2. `live_capture_scapy()`

- **Purpose:** The main function that orchestrates the packet capture process.
- **Details:**
  - **Interface Selection:** Determines the target interface, either from a command-line argument (`sys.argv[1]`) or by calling `find_active_interface()` .
  - Initializes the `ScapyTrafficAnalyzer` class.
  - Starts the packet capture using Scapy's `sniff` function, passing `analyzer.analyze_packet` as the callback function (`prn`).
  - Uses `store=0` to prevent Scapy from storing packets in memory, saving resources.

- Includes robust error handling for `PermissionError` (requires `sudo`) and `KeyboardInterrupt` (Ctrl+C) to ensure a final summary report is printed.

3. `if __name__ == "__main__":`

- **Purpose:** Standard Python entry point.
- **Details:** Calls `live_capture_scapy()` to start the program when the script is executed directly.