NEED TO CREATE A FRONT-END FOR THIS

Below is a **comprehensive Technical Manual and How-To Guide** for the tool **EAT.py** (an acronym for **Ebcdic, Ascii, TLS** Sniffer). This guide covers:

1. **Introduction**
2. **Requirements & Installation**
3. **How EAT.py Works**
4. **Command-Line Usage**
5. **Feature Details**
6. **Examples**
7. **Troubleshooting & Best Practices**
8. **Appendix A: Changes**

---

1. Introduction

EAT.py stands for **Ebcdic, Ascii, TLS** sniffer. It is a command-line tool that uses the Scapy library to sniff network traffic and decode different protocols:

1. **ASCII**: Primarily plain text traffic (e.g., Telnet, HTTP in clear-text, etc.).
2. **EBCDIC**: Specifically looking for **3270-like** EBCDIC data often used in mainframe terminal environments.
3. **TLS**: Secure traffic on TCP port 443 (TLS/SSL). By default, this will capture encrypted data only, though additional scapy-ssl_tls extensions can help parse some handshake details.

Additionally, EAT.py can monitor **multiple ports** simultaneously, highlight traffic from known **malicious IP addresses**, and save the captured data to a PCAP file.

---

2. Requirements & Installation

1. **Operating System**: Linux or other platforms that support Python 3 and allow packet capture.
2. **Python Version**: Python 3.6+ recommended.
3. **Privileges**: Packet sniffing typically requires **root** privileges on Linux or Administrator on other systems.

4. **Dependencies**:
    - **Scapy** (install via pip install scapy)
    - (Optional) **scapy-ssl_tls** if you want deeper parsing of TLS. (Not strictly required for basic sniffing.)

Installation Steps

1. Ensure Python 3 is installed:
2. python3 --version
3. Install Scapy:
4. pip install scapy
5. Download or place **EAT.py** in a local directory of your choice.
6. (Optional) Make **EAT.py** executable:
7. chmod +x EAT.py
8. Run with root or sudo privileges (e.g., sudo ./EAT.py ...).

---

3. How EAT.py Works

EAT.py leverages Scapy's powerful sniffing capabilities. Internally, EAT.py does the following:

1. **Parses Command-Line Arguments**: Determines whether to sniff EBCDIC, ASCII, or TLS traffic, what target IP to watch, which ports, whether a malicious IP file is provided, etc.
2. **Loads Malicious IPs**: If a JSON file with malicious IPs is specified (e.g., --malicious-ip-file malicious.json), EAT.py loads it into a set for quick membership checks.
3. **Constructs a BPF Filter** (Berkeley Packet Filter) for the target IP and ports, unless TLS mode is enabled.
4. **Starts Sniffing**: EAT.py calls Scapy's sniff() function, passing in the filter and the callback function.
5. **Callback Function**:
    - Identifies the **source IP** of each incoming packet.
    - Prints a **one-time** message when a new IP is detected (highlights in red if it is malicious).

- If in ASCII mode, attempts to decode the payload as ASCII.
- If in EBCDIC mode, attempts to detect 3270 data and decode it using cp500.
- If in TLS mode, simply captures traffic on port 443 and prints a generic message (unless scapy-ssl_tls is used for deeper analysis).
- Appends all captured packets to a PCAP file.

---

4. Command-Line Usage

Below is the **help** message structure for **EAT.py**. You can run ./EAT.py --help for details:

```
usage: EAT.py [-h] [--tls] [--malicious-ip-file MALICIOUS_IP_FILE]
              [-p PORTS [PORTS ...]] [-m MODES [MODES ...]]
              [-i IFACE]
              [TARGET]

Packet Sniffer that can handle ASCII, EBCDIC (3270), or TLS traffic.

positional arguments:
  TARGET                Target IP for traffic sniffing (omit if --tls is used).

optional arguments:
  -h, --help            show this help message and exit
  --tls                 Capture TLS traffic on port 443 instead of EBCDIC/ASCII.
                        If specified, TARGET/PORTS are ignored.
  --malicious-ip-file   Path to JSON file with malicious IPs. (Default: None)
  -p PORTS [PORTS ...]  One or more target ports (Default: 3270).
                        Example: -p 23 3270 80
  -m MODES [MODES ...]  Which traffic types to parse: ascii, ebcdic, or both.
                        Example: -m ascii ebcdic
  -i IFACE              Interface to use (default: tries system default).
```

Key Options

- **TARGET**: The IP address you wish to sniff traffic for (e.g., 10.0.0.1). If you enable --tls, you do **not** need TARGET.
- **-p/--ports**: One or more ports to monitor (e.g., -p 80 23 3270). Defaults to 3270.
- **-m/--modes**: ascii, ebcdic, or both (ascii ebcdic). By default, only ebcdic is activated.
- **--tls**: Switches to TLS sniffing on port 443. This ignores TARGET and PORTS.
- **--malicious-ip-file**: A path to a JSON file containing malicious IPs. The JSON structure is expected to look like:

```
{
 "malicious_ips": [
  "192.168.1.10",
  "10.0.0.7"
 ]
}
```

---

5. Feature Details

5.1 Multiple Ports

EAT.py can sniff multiple ports simultaneously. Internally, the BPF filter constructs a statement such as:

host <TARGET> and tcp and (port <P1> or port <P2> or ...)

Example:

./EAT.py 192.168.1.10 -p 23 3270 80 -m ascii

This will sniff TCP traffic to or from the host 192.168.1.10 on ports 23, 3270, and 80, attempting to decode any payloads as **ASCII**.

5.2 Multiple Modes (ASCII + EBCDIC)

EAT.py offers two parsing modes:

1. **ASCII**: Plain-text data, typically for protocols like Telnet, HTTP, etc.
2. **EBCDIC**: A specialized regex-based detection of 3270 traffic. Decoded using code page 500 (cp500).

You can enable one or both by specifying -m ascii or -m ebcdic, or -m ascii ebcdic.

When both modes are active, EAT.py **attempts to parse** each payload **twice**:

1. First, it tries ASCII decoding.
2. Then, it checks for a 3270-like EBCDIC signature and, if found, decodes it.

5.3 TLS Traffic

With the --tls flag, EAT.py ignores the TARGET and PORTS arguments, defaulting to traffic on **TCP port 443**. By default, these packets are encrypted; if you want to parse handshake details, you'll need additional setup:

- **Install scapy-ssl_tls** (e.g., pip install scapy-ssl_tls)
- Configure environment variables like SSLKEYLOGFILE or run a MITM approach to intercept keys.

In standard usage, EAT.py will simply show a message for each captured TLS packet and log them to tls_traffic.pcap.

5.4 Malicious IP Feature

If you specify --malicious-ip-file <FILE>, EAT.py loads that JSON file at the start. Whenever a new **source IP** is detected, EAT.py checks if that IP is in the malicious list. If so, it prints:

[!] Malicious IP connection: <IP>

in **red**. Otherwise, it prints:

Incoming connection from: <IP>

Crucially, EAT.py will print each **source IP** (malicious or not) **only once** thanks to the recent update (see Appendix). Subsequent packets from the same IP will not re-print the connection message, but you will still see any decoded payload data.

---

6. Examples

6.1 Example 1: Monitoring Telnet ASCII Traffic

Scenario: You want to sniff Telnet traffic (port 23) on host 192.168.1.10. You also have a file listing malicious IPs that might be scanning or connecting.

Command:

sudo ./EAT.py 192.168.1.10 -p 23 -m ascii --malicious-ip-file malicious.json

- **What happens**:

1. EAT.py checks for malicious IPs from malicious.json.
2. A BPF filter is constructed:
3. host 192.168.1.10 and tcp and (port 23)
4. When a new IP sends traffic to or from 192.168.1.10 on port 23, EAT.py prints one of the following:
    - **Malicious**:
    - [!] Malicious IP connection: 10.0.0.5
    - **Non-malicious**:
    - Incoming connection from: 10.0.0.3
5. For each packet's payload, EAT.py prints:
6. [ASCII] (some text data)
7. All packets are saved to traffic.pcap when you press **Ctrl + C**.

6.2 Example 2: Monitoring 3270 EBCDIC Data on Multiple Ports

Scenario: You have a mainframe environment that uses port 3270 and 992 for TN3270 traffic. You'd like to capture both.

Command:

sudo ./EAT.py 10.10.10.10 -p 3270 992 -m ebcdic

- **What happens**:
    1. EAT.py constructs a BPF filter to catch any traffic to/from 10.10.10.10 on 3270 **or** 992.
    2. EAT.py attempts to detect **3270** patterns in each packet. If found, prints:
    3. [EBCDIC] HelloWorld
    4. Source IPs are printed only once.

6.3 Example 3: Sniffing Both ASCII and EBCDIC, Multiple Ports

Command:

sudo ./EAT.py 192.168.1.100 -p 23 3270 80 -m ascii ebcdic

- EAT.py sees if the payload can be decoded as ASCII. Then it also looks for EBCDIC patterns. You might see output like:
- Incoming connection from: 192.168.1.55

- [ASCII] GET / HTTP/1.1
- [EBCDIC] ...

6.4 Example 4: TLS Only

Command:

sudo ./EAT.py --tls

- Ignores any other arguments. Listens on **port 443** for TLS traffic.
- Prints a message like:
- TLS packet captured (likely encrypted).
- Saves them to tls_traffic.pcap.

---

7. Troubleshooting & Best Practices

1. **Permissions**: You must run EAT.py with sufficient permissions to sniff packets (root or sudo on most systems).
2. **Interface Detection**: By default, EAT.py tries to guess the default interface on Linux by reading /proc/net/route. If this fails or if you use an alternate interface (e.g., eth1 or wlan0), specify -i <interface>.
3. **Performance**: Sniffing large amounts of traffic can be resource-intensive. If you need high throughput, consider using filters that are as specific as possible (e.g., narrower subnets, specific ports).
4. **Decryption of TLS**: The default tool does not decrypt TLS traffic. Use scapy-ssl_tls or another approach if you need deeper analysis.
5. **Malicious IP Checking**: Make sure your JSON file is properly formatted. A small syntax error can prevent EAT.py from loading the file.

---

8. Appendix A: Changes

Recent Updates

- **One-Time IP Printing**
    - The script previously printed “Incoming connection from...” or “[!] Malicious IP connection: ...” on **every** packet from that IP.
    - **Now**, we maintain an internal set called seen_ips. Each **source IP** is printed only **once** when it first appears. Subsequent packets from the

same IP do **not** re-print the connection message, though they do still trigger ASCII/EBCDIC decoding messages.

- This change reduces console spam and makes reading the sniff output much more manageable.

Below is a snippet illustrating the core logic in the callback function:

```
def combined_callback(packet, malicious_ips, modes, seen_ips):
  ...
  if src_ip not in seen_ips:
    if src_ip in malicious_ips:
      print(f"{RED}[!] Malicious IP connection: {src_ip}{RESET}")
    else:
      print(f"Incoming connection from: {src_ip}")
    seen_ips.add(src_ip)
  ...
```

With this, **EAT.py** remains a flexible tool for both ASCII/EBCDIC traffic analysis and TLS capturing, now with less repetitive output.