Rapporto di Analisi del Codice: Simple Packet Sniffer

Obiettivo :

Creare un programma in python che catturi il socket di rete.

Descrizione generale :

Questo script rappresenta una soluzione fondamentale per l'analisi del traffico di rete. Funziona come un ***"sniffer"*** di rete basilare, la cui funzione principale è l'intercettazione e la cattura di ***pacchetti grezzi*** (raw packets) direttamente dall'interfaccia di rete del sistema su cui è in esecuzione.

Funzionalità Principali:

1. **Cattura di Pacchetti Grezzi:** Lo script opera al livello più basso consentendogli di acquisire i pacchetti nella loro forma originale, prima che vengano processati dallo stack TCP/IP del sistema operativo.
2. **Analisi e Filtro:** Dopo la cattura, lo script esegue un'analisi del contenuto di ciascun pacchetto. Il suo obiettivo specifico è l'identificazione e la verifica di pacchetti che corrispondono a un *indirizzo IP* di destinazione o di origine predefinito (specifico).
3. **Visualizzazione Dettagliata:** Una volta che un pacchetto soddisfa il criterio di corrispondenza IP, lo script procede a estrarre e mostrare i *dettagli* significativi del pacchetto. Questi dettagli includono tipicamente le intestazioni di rete come Indirizzi MAC sorgente/destinazione, Indirizzi IP sorgente/destinazione, protocollo di trasporto, numeri di porta e potenzialmente una porzione del carico utile (payload).

Analisi del codice:

Import delle librerie :

```
import socket
import struct
import os
import sys
```

- Socket : **Crea la connessione** di rete a basso livello per ascoltare tutto il traffico.
- Struct: **Converte i dati binari grezzi** (byte) **in numeri e stringhe leggibili.**

- OS : **Interagisce con il sistema operativo**, qui usato specificamente per controllare se l'utente è amministratore (root).
- Sys : **Gestisce i parametri di sistema**, usato qui per terminare il programma in caso di errore.

Validazione dei Privilegi di Esecuzione:

```
def require_root():
    if os.geteuid() != 0:
        print("[!] Raw sockets require root privileges.")
        sys.exit(1)
```

Questa funzione implementa un controllo preventivo dei permessi a livello del sistema operativo.

Lo script interroga il kernel per ottenere l'identificativo dell'utente, confronta L'id con "0" che nei sistemi Linux è riservato all'account ***"Root"***.
Se l'Id è diverso da "0" lo script esegue ***"sys.exit(1)"*** e termina immediatamente.

Enumerazione e Selezione Automatica dell'Interfaccia di Rete

```
def get_default_interface() -> str:

    for _, iface in socket.if_nameindex():
        if iface != "lo":
            return iface
    raise RuntimeError("No suitable network interface found")
```

Questa funzione ha lo scopo di identificare l'interfaccia di rete fisica da usare per lo sniffing.

- *socket.if_nameindex()* : questo comando è una chiamata di sistema a basso livello, restituendo una lista di tuple contenenti tutte le interfacce di rete disponibili nel kernel.
- *if iface != "lo"*: verifica se l'interfaccia corrente è l'interfaccia di loopback.
- *return iface* : strategia di uscita poichè appena trova che l'interfaccia non è loopback la restituisce e termina la funzione.
- *raise RuntimeError(...)* : per la gestione degli errori

Inizializzazione del Raw Socket e Binding Layer-2

```python
def capture_packets_for_ip(target_ip: str, max_packets: int = 10):
    interface = get_default_interface()

    sock = socket.socket(
        socket.AF_PACKET,
        socket.SOCK_RAW,
        socket.ntohs(0x0003)
    )

    sock.bind((interface, 0))

    print(f"[+] Using interface: {interface}")
    print(f"[+] Filtering packets for IP: {target_ip}\n")

    captured = 0
```

Questo segmento di codice è la **fase di preparazione** poiché sta configurando il canale di ascolto affinché il sistema operativo passi al programma tutto ciò che transita sulla scheda di rete.

- *socket.AF_PACKET* : permette di parlare direttamente con il driver della scheda di rete permettendoci di lavorare al livello 2 del modello OSI.
- *socket.SOCK_RAW* : Permette di rivedere totalmente il pacchetto senza scartare nulla.
- *socket.ntohs(0x0003)* : indispensabile per catturare tutti i protocolli in entrata e uscita.
- *.bind()* : collega il socket software all'interfaccia hardware

Ciclo di Acquisizione e Decodifica del Header IPv4

```
while captured < max_packets:
    raw_data, _ = sock.recvfrom(65535)

    eth_proto = struct.unpack("!H", raw_data[12:14])[0]

    if eth_proto == 0x0800:  # IPv4
        ip_header = raw_data[14:34]
        iph = struct.unpack("!BBHHHBBH4s4s", ip_header)

        src_ip = socket.inet_ntoa(iph[8])
        dst_ip = socket.inet_ntoa(iph[9])

        if src_ip == target_ip or dst_ip == target_ip:
            captured += 1
            print(f"Packet {captured}")
            print(f"  Source IP      : {src_ip}")
            print(f"  Destination IP : {dst_ip}")
            print(f"  Length         : {len(raw_data)} bytes\n")

sock.close()
print("[+] Capture complete")
```

La prima parte dello screen rappresenta il **motore dello script**, poiché tramite il ciclo while si ottengono byte grezzi dal cavo di rete e usa la libreria “Struct” per dare un significato ai byte.

- *raw_data, _ = sock.recvfrom(65535)* : prende tutto il pacchetto fino ad un massimo di 65 KB, dimensione teorica del pacchetto.
- *raw_data[12:14]* : I byte 12 e 13 contengono l'**EtherType**, che ci dice cosa c'è all’interno del pacchetto (IPv4, IPv6, ARP?).

Una volta saputo che è IPv4, saltiamo i primi 14 byte (l'intestazione Ethernet) e leggiamo i successivi 20 byte, che costituiscono l'**Header IP standard**.

Qui avviene la decodifica complessa con la **Regex** *"!BBHHHBBH4s4s"* che mappa i 20 byte dell’Header Ip tramite uno schema specifico.

- *if src_ip == target_ip or dst_ip == target_ip*: se l'IP mittente o l'IP destinatario coincidono con quello che hai digitato all'inizio, il pacchetto viene stampato a schermo e il contatore ***“captured”*** *(visualizzato in precedenza)* aumenta.

Entry Point dell'Esecuzione e Bootstrap dell'Ambiente

```python
def main():
    require_root()
    target_ip = input("Target IP to filter: ").strip()
    capture_packets_for_ip(target_ip)


if __name__ == "__main__":
    main()
```

In questa ultima parte del codice troviamo :

- *require_root()* : Blocca tutto immediatamente se non si viene considerati amministratori;
- *capture_packets_for_ip()* : Avvia il ciclo descritto sopra;

Output del programma :

```
┌──(kali㉿kali)-[~/Desktop/Python-Programs]
└─$ sudo python SocketReteperRelazione.py
[+] Filtering packets for IP: 192.168.1.101

Packet 1
  Source IP      : 192.168.1.101
  Destination IP : 20.42.65.93
  Length         : 66 bytes

Packet 2
  Source IP      : 20.42.65.93
  Destination IP : 192.168.1.101
  Length         : 66 bytes
```

Packet 1 : (Traffico in uscita)

- **Source IP** *(192.168.1.101)*: Il pacchetto parte dalla tua macchina.
- **Destination IP** *(20.42.65.93)*: Il pacchetto è diretto verso un server esterno su Internet.

.

- **Length** *(66 bytes)*: È un pacchetto piccolo. Probabilmente è un pacchetto di controllo TCP (come un ACK o un SYN) o una richiesta DNS, non un trasferimento di file pesante.

Packet 2**:** La Risposta (Traffico in Entrata)

- **Source IP** *(20.42.65.93)*: Il server esterno risponde.
- **Destination IP** *(192.168.1.101)*: Il pacchetto torna alla tua macchina.