RELAZIONE TESTING DI RETE

06 SETTEMBRE 2024

PANTHER PWNERS

Progetto 1: Port Scanner

Il lavoro svolto ha prodotto un programma utilizzato per scansionare le porte su un indirizzo IP specificato dall'utente, fornendo un report delle porte aperte e dei servizi in ascolto su tali porte.

Il programma inizia importando tutte le funzioni e classi dal modulo socket, che consente la comunicazione di rete tra computer, e successivamente richiede all'utente di inserire l'indirizzo IP del target da scansionare insieme ad un intervallo di porte, specificando la porta iniziale e quella finale.

Il programma entra quindi in un ciclo che scansiona ogni porta compresa nell'intervallo specificato, dove per ogni porta crea un oggetto socket per la comunicazione TCP tentando di connettersi alla porta del target per verificare che la connessione abbia successo. Se tale connessione riesce, indicando che la porta è aperta, il programma determina il servizio associato alla porta utilizzando la funzione 'getservbyport' e aggiunge un messaggio alla lista che indica che la porta è aperta specificandone il servizio, mentre se la connessione fallisce la porta è considerata chiusa, e come tale non viene stampata sul terminale di modo da alleggerire l'output.

Durante la scansione il programma stampa quale porta sta attualmente scansionando, di modo da fornire all'utente un aggiornamento in tempo reale del suo avanzamento, e dopo aver fornito l'elenco di porte aperte e servizi associati, si conclude con un messaggio che ne indica la terminazione.

Perché utilizzare questo programma

L'utilizzo di questo programma è consigliato per migliorare la sicurezza e gestire le risorse di rete, mantenere la rete aziendale efficiente e conforme alle normative di sicurezza, in particolare:

1. Verifica della sicurezza della rete: il programma può essere usato per identificare le porte aperte su un server o un dispositivo di rete, valutando così i potenziali rischi di sicurezza e fornendo un modo per garantire che solo le porte necessarie siano accessibili, riducendo così la superficie di attacco.
2. Gestione delle risorse di rete: fornendo una panoramica delle porte aperte e dei servizi associati, il programma consente agli amministratori di rete di monitorare quali servizi sono in esecuzione sui loro dispositivi, aiutando a gestire le risorse di rete e garantendo che solo i servizi autorizzati siano attivi.
3. Troubleshooting e diagnosi: elencando quali porte e servizi sono aperti, il programma può essere utilizzato nell'aiuto della diagnosi di eventuali problemi di rete.
4. Conformità alle normative di sicurezza: utilizzare un programma di scansione delle porte può aiutare a garantire la conformità a normative di suicurezza e best practice richieste all'azienda nel verificare regolarmente le configurazioni di rete.
5. Prevenzione di accessi non autorizzati: il programma facilita l'individuazione di potenziali vulnerabilità nella rete, elencando le porte non necessarie di modo da prevenire accessi non autorizzati alla rete.

Codice completo commentato

```
# Importare tutte le funzioni e classi dal modulo 'socket' che consente
la comunicazione tra computer in una rete
from socket import *

# Chiedere all'utente di inserire l'indirizzo IP da scansionare e il
range delle porte
target = input("Inserisci l'indirizzo IP da scansionare: ")
porta_iniziale = input("Inserisci il numero di porta da cui iniziare la
scansione: ")
porta_finale = input("Inserisci il numero di porta con cui terminare la
scansione: ")

# Convertire gli input da str a int
inizio = int(porta_iniziale)
fine = int(porta_finale) + 1

# Inizializzare una lista per lo stato delle porte
stato_porte = []

# Iniziare un ciclo for dalla prima all'ultima porta selezionata
for i in range(inizio, fine):

    # Creare un oggetto socket usando IPv4 (AF_INET) e TCP
(SOCK_STREAM) per ogni porta
    s = socket(AF_INET, SOCK_STREAM)
    s.settimeout(0.2)

    # Connettersi all'IP del target alla porta corrente
    risultato = s.connect_ex((target, i))

    # Determinare lo stato della porta e l'eventuale servizio associato
    if risultato == 0:
        service = getservbyport(i)
        stato_porte.append(f"Porta {i}: Aperta | Servizio: {service}")
    else:
        stato_porte.append

    # Chiudere il socket per evitare problemi di connessione successiva
    s.close()

    # Stampare il numero di porta attualmente in fase di scan
    print("Attualmente in scan: porta ", i)

# Stampare l'elenco delle porte aperte
print("\nRisultati della scansione:")
for risultato in stato_porte:
    print(risultato)
```

```
# Stampare il termine della scansione
print("\nLa scansione è terminata.")
```

Esempio di output

```
Inserisci l'indirizzo IP da scansionare: 192.168.1.248
Inserisci il numero di porta da cui iniziare la scansione: 20
Inserisci il numero di porta con cui terminare la scansione: 80
Attualmente in scan: porta  20
Attualmente in scan: porta  21
Attualmente in scan: porta  22
Attualmente in scan: porta  23
Attualmente in scan: porta  24
Attualmente in scan: porta  25
Attualmente in scan: porta  26
Attualmente in scan: porta  27
Attualmente in scan: porta  28
Attualmente in scan: porta  29
Attualmente in scan: porta  30
Attualmente in scan: porta  31
Attualmente in scan: porta  32
Attualmente in scan: porta  33
Attualmente in scan: porta  34
Attualmente in scan: porta  35
Attualmente in scan: porta  36
Attualmente in scan: porta  37
Attualmente in scan: porta  38
Attualmente in scan: porta  39
Attualmente in scan: porta  40
Attualmente in scan: porta  41
Attualmente in scan: porta  42
Attualmente in scan: porta  43
Attualmente in scan: porta  44
Attualmente in scan: porta  45
Attualmente in scan: porta  46
```

```
Attualmente in scan: porta  47
Attualmente in scan: porta  48
Attualmente in scan: porta  49
Attualmente in scan: porta  50
Attualmente in scan: porta  51
Attualmente in scan: porta  52
Attualmente in scan: porta  53
Attualmente in scan: porta  54
Attualmente in scan: porta  55
Attualmente in scan: porta  56
Attualmente in scan: porta  57
Attualmente in scan: porta  58
Attualmente in scan: porta  59
Attualmente in scan: porta  60
Attualmente in scan: porta  61
Attualmente in scan: porta  62
Attualmente in scan: porta  63
Attualmente in scan: porta  64
Attualmente in scan: porta  65
Attualmente in scan: porta  66
Attualmente in scan: porta  67
Attualmente in scan: porta  68
Attualmente in scan: porta  69
Attualmente in scan: porta  70
Attualmente in scan: porta  71
Attualmente in scan: porta  72
Attualmente in scan: porta  73
Attualmente in scan: porta  74
Attualmente in scan: porta  75
Attualmente in scan: porta  76
Attualmente in scan: porta  77
Attualmente in scan: porta  78
Attualmente in scan: porta  79
```

```
Attualmente in scan: porta  80


Risultati della scansione:
Porta 21: Aperta | Servizio: ftp
Porta 22: Aperta | Servizio: ssh
Porta 23: Aperta | Servizio: telnet
Porta 25: Aperta | Servizio: smtp
Porta 53: Aperta | Servizio: domain
Porta 80: Aperta | Servizio: http


La scansione è terminata.
```

Descrizione dettagliata del codice

```
# Importare tutte le funzioni e classi dal modulo 'socket' che consente la comunicazione tra computer in una rete
from socket import *
```

- Questa prima linea importa tutte le funzioni dal modulo socket, che permette di creare e gestire connessioni di rete consentendo la comunicazione tra computer.

```
# Chiedere all'utente di inserire l'indirizzo IP da scansionare e il range delle porte
target = input("Inserisci l'indirizzo IP da scansionare: ")
porta_iniziale = input("Inserisci il numero di porta da cui iniziare la scansione: ")
porta_finale = input("Inserisci il numero di porta con cui terminare la scansione: ")
```

- Queste tre righe chiedono all'utente di inserire l'indirizzo IP del computer da scansionare (target) e il range di porte da scansionare (porta_iniziale e porta_finale).
- input() raccoglie l'input dell'utente come stringa.

```
# Convertire gli input da str a int
inizio = int(porta_iniziale)
fine = int(porta_finale) + 1
```

- Questa parte del codice converte i valori delle porte da stringhe (str) a numeri interi (int), poiché le porte devono essere numeri interi mentre l'input dell'utente viene registrato come stringa.
- fine = int(porta_finale) + 1 aggiunge 1 al valore di porta_finale per includere l'ultima porta nel ciclo for successivo, poiché range() esclude il valore finale.

```
# Inizializzare una lista per lo stato delle porte
stato_porte = []
```

- Qui viene creata una lista vuota chiamata stato_porte, che verrà utilizzata per memorizzare i risultati della scansione delle porte.

```
# Iniziare un ciclo for dalla prima all'ultima porta selezionata
for i in range(inizio, fine):
```

- Il ciclo for itera attraverso ogni porta nell'intervallo specificato.

```
# Creare un oggetto socket usando IPv4 (AF_INET) e TCP (SOCK_STREAM) per ogni porta
s = socket(AF_INET, SOCK_STREAM)
s.settimeout(0.2)
```

- s = socket(AF_INET, SOCK_STREAM) crea un nuovo oggetto socket s per ogni porta, utilizzando IPv4 (AF_INET) e TCP (SOCK_STREAM).
- s.settimeout(0.2) imposta un timeout di 0,2 secondi per l'operazione di connessione, in modo che la scansione non resti bloccata su una porta per troppo tempo.

```
# Connettersi all'IP del target alla porta corrente
risultato = s.connect_ex((target, i))
```

- risultato = s.connect_ex((target, i)) tenta di connettersi all'indirizzo IP target alla porta i.
- connect_ex() restituisce 0 se la connessione ha successo indicando che la porta è aperta.

```
# Determinare lo stato della porta e l'eventuale servizio associato
 if risultato == 0:
     service = getservbyport(i)
     stato_porte.append(f"Porta {i}: Aperta | Servizio: {service}")
 else:
     stato_porte.append
```

- if risultato == 0: verifica se la porta è aperta (se risultato è 0).
    - Se la porta è aperta, service = getservbyport(i) tenta di ottenere il nome del servizio associato a quella porta.
    - stato_porte.append(f"Porta {i}: Aperta | Servizio: {service}") aggiunge alla lista stato_porte una stringa che indica che la porta è aperta e specifica il servizio associato.
- Se la porta è chiusa, stato_porte.append non aggiunge stringhe per evitare ridondanza di informazioni e facilitare la lettura dei risultati.

```
# Chiudere il socket per evitare problemi di connessione successiva
s.close()
```

- Questa linea chiude il socket s per evitare problemi di connessione nelle iterazioni successive, liberando risorse e prevenendo errori di connessione.

```
# Stampare il numero di porta attualmente in fase di scan
```

```
 print("Attualmente in scan: porta ", i)
```

- Questa linea stampa sullo schermo la porta attualmente in fase di scansione, fornendo un feedback in tempo reale all'utente sul progresso della scansione.

```
# Stampare l'elenco delle porte aperte

print("\nRisultati della scansione:")

for risultato in stato_porte:

   print(risultato)
```

- print("\nRisultati della scansione:") stampa un'intestazione per i risultati della scansione.
- Il ciclo for itera attraverso la lista stato_porte e stampa lo stato di ciascuna porta aperta.

```
# Stampare il termine della scansione

print("\nLa scansione è terminata.")
```

- Questa linea stampa un messaggio finale per informare l'utente che la scansione è terminata.

Progetto 2: Interrogatore verbi HTTP

Il lavoro svolto ha prodotto un programma utilizzato per elencare e interagire con i metodi HTTP supportati da un URL o indirizzo IP fornito dall'utente, in particolare GET, POST, PUT e DELETE.

Il programma inizia, subito dopo aver importato i moduli necessari per la sua corretta esecuzione, chiedendo all'utente di inserire l'URL o l'indirizzo IP target che desidera interrogare e definendo una lista di metodi HTTP andando a controllare quali di questi metodi sono supportati dal target. Per ogni metodo, il programma invia una richiesta HTTP e verifica che il codice di stato della risposta sia inferiore a 400, indicandone la validità, aggiungendo tale metodo alla lista di metodi abilitati.

Dopo aver identificato i metodi supportati, il programma stampa l'elenco di questi metodi. Se nessun metodo è supportato, informa l'utente che non sono stati trovati metodi HTTP supportati per il target specificato.

Successivamente, il programma entra in un loop che presenta un menù interattivo all'utente, mostrando le opzioni basate sui metodi HTTP che sono risultati abilitati. L'utente può selezionare uno dei verbi HTTP ritornati digitando il numero associato a ciascun verbo. Se il metodo è supportato, il programma esegue una richiesta utilizzando quel metodo e stampa il codice di stato della risposta ricevuta; se l'utente digita un'opzione non supportata viene stampato un messaggio di di errore, mentre se l'utente sceglie di terminare il programma il loop si interrompe e il programma si termina.

Perché utilizzare questo programma

Un'azienda potrebbe trovare utile questo programma per diversi motivi legati alla gestione e alla sicurezza delle proprie applicazioni web e server, in particolare:

1. Verifica dei Metodi HTTP supportati: il programma permette di identificare quali metodi HTTP sono abilitati su un server, garantendo che solo i metodi necessari e sicuri siano esposti, aiutando così a prevenire potenziali vulnerabilità.
2. Sicurezza e conformità: testare i metodi HTTP supportati può aiutare a identificare configurazioni non sicure o inadeguate che potrebbero esporre l'applicazione a rischi di sicurezza. Ad esempio, potrebbe essere pericoloso avere il metodo DELETE accessibile pubblicamente se questo non fosse necessario. Assicurarsi che solo i metodi necessari siano abilitati è una pratica di sicurezza fondamentale.
3. Troubleshooting e diagnosi: in caso di problemi con un'applicazione web, il programma può aiutare a diagnosticare se i metodi HTTP che ci si aspetta siano abilitati e funzionanti.

Codice completo commentato

```
# Importare il modulo 'request' per gestire le richieste HTTP
import requests

# Importare il modulo 'time' per gestire il countdown del menù
import time

# Chiedere all'utente di inserire l'indirizzo da interrogare
target = input("Inserisci l'URL o l'IP del target da interrogare: ")

# Creare la lista dei metodi HTTP da controllare
metodi = ['GET', 'POST', 'PUT', 'DELETE']
metodi_abilitati = []

# Controllare quali metodi HTTP sono abilitati sull'indirizzo dato,
considerandoli tale solo se ritorna un codice di stato inferiore a 400
for metodo in metodi:
    risposta = requests.request(metodo, f"http://{target}")
    if risposta.status_code < 400:
        metodi_abilitati.append(metodo)

# Stampare l'output dei metodi HTTP abilitati
if metodi_abilitati:
    print(f"Metodi HTTP abilitati per {target}:")
    for metodo in metodi_abilitati:
        print(f"- {metodo}")
else:
    print(f"Nessun metodo HTTP supportato trovato per {target}.")

# Iniziare un loop per dare molteplici opzioni all'utente fin quando
non decide di terminare il programma,
# mostrando il menù all'utente per scegliere un verbo HTTP basato sui
metodi abilitati trovati

while True:
    print("\nScegli un verbo HTTP:")
    if 'GET' in metodi_abilitati:
        print("Digita 1 per GET;")
    if 'POST' in metodi_abilitati:
        print("Digita 2 per POST;")
    if 'PUT' in metodi_abilitati:
        print("Digita 3 per PUT;")
    if 'DELETE' in metodi_abilitati:
        print("Digita 4 per DELETE;")
    print("Digita 0 per terminare il programma.")

    # Inserire la scelta dell'utente
    scelta = int(input("\nLa tua scelta: "))
```

```
    # Eseguire l'operazione in base alla scelta dell'utente e ai metodi
abilitati
    if scelta == 1 and 'GET' in metodi_abilitati:
        risposta = requests.get(f"http://{target}")
        print("\nRisposta web:", risposta.status_code)

    elif scelta == 2 and 'POST' in metodi_abilitati:
        risposta = requests.post(f"http://{target}")
        print("\nRisposta web:", risposta.status_code)

    elif scelta == 3 and 'PUT' in metodi_abilitati:
        risposta = requests.put(f"http://{target}")
        print("\nRisposta web:", risposta.status_code)

    elif scelta == 4 and 'DELETE' in metodi_abilitati:
        risposta = requests.delete(f"http://{target}")
        print("\nRisposta web:", risposta.status_code)

    elif scelta == 0:
        print("\nIl programma si termina.")
        break

    else:
        print("\nInput non valido. Riprova.")

    # Creare un countdown di tre secondi prima di riproporre il menù
    for i in range(0, 3):
        print(".")
        time.sleep(1)
```

Esempio di output

```
Inserisci l'URL o l'IP del target da interrogare: 192.168.1.248
Metodi HTTP abilitati per 192.168.1.248:
- GET
- POST
- PUT
- DELETE

Scegli un verbo HTTP:
Digita 1 per GET;
Digita 2 per POST;
Digita 3 per PUT;
Digita 4 per DELETE;
Digita 0 per terminare il programma.

La tua scelta: 1

Risposta web: 200
.
.
.
Scegli un verbo HTTP:
Digita 1 per GET;
Digita 2 per POST;
Digita 3 per PUT;
Digita 4 per DELETE;
Digita 0 per terminare il programma.

La tua scelta: 2

Risposta web: 200
.
.
```

```
.
Scegli un verbo HTTP:
Digita 1 per GET;
Digita 2 per POST;
Digita 3 per PUT;
Digita 4 per DELETE;
Digita 0 per terminare il programma.

La tua scelta: 3

Risposta web: 200
.
.
.
Scegli un verbo HTTP:
Digita 1 per GET;
Digita 2 per POST;
Digita 3 per PUT;
Digita 4 per DELETE;
Digita 0 per terminare il programma.

La tua scelta: 4

Risposta web: 200
.
.
.
Scegli un verbo HTTP:
Digita 1 per GET;
Digita 2 per POST;
Digita 3 per PUT;
Digita 4 per DELETE;
Digita 0 per terminare il programma.
```

```
La tua scelta: 5


Input non valido. Riprova.
.
.
.
Scegli un verbo HTTP:
Digita 1 per GET;
Digita 2 per POST;
Digita 3 per PUT;
Digita 4 per DELETE;
Digita 0 per terminare il programma.

La tua scelta: 0


Il programma si termina.
```

Descrizione dettagliata del codice

```
# Importare il modulo 'request' per gestire le richieste HTTP

import requests
```

- Il modulo ‘request’ permette di inviare e gestire richieste HTTP.

```
# Importare il modulo 'time' per gestire il countdown del menù

import time
```

- Il modulo ‘time’ fornisce funzioni per lavorare con il tempo, come sleep, che verrà usato in seguito nel ciclo del menù per fare una pausa per un certo numero di secondi prima di riproporre le opzioni di scelta.

```
# Chiedere all'utente di inserire l'indirizzo da interrogare

target = input("Inserisci l'URL o l'IP del target da interrogare: ")
```

- Questa riga chiede all’utente di inserire l’URL o l'indirizzo IP del server che si vuole interrogare salvando lìinput nella variabile ‘target’.

```
# Creare la lista dei metodi HTTP da controllare

metodi = ['GET', 'POST', 'PUT', 'DELETE']

metodi_abilitati = []
```

- Nella prima riga viene creata una lista contenente i metodi HTTP da testare (GET, POST, PUT, DELETE) mentre nella seconda viene creata una lista vuota dove verranno memorizzati i metodi HTTP che risulteranno supportati dal server.

```
# Controllare quali metodi HTTP sono abilitati sull'indirizzo dato, considerandoli tale solo se ritorna un codice di stato inferiore a 400

for metodo in metodi:

    risposta = requests.request(metodo, f"http://{target}")

    if risposta.status_code < 400:

        metodi_abilitati.append(metodo)
```

- In questo blocco viene creata un’iterazione ‘for’ tale per cui per ogni ‘metodo’ nella lista ‘metodi’ viene inviata una richiesta HTTP usando ‘requests.request’ con il metodo corrente. Si verifica poi il codice di stato della risposta, e solo se è inferiore a 400 (indicando quindi successo nella risposta) il metodo viene aggiunto alla lista ‘metodi_abilitati’.

```
# Stampare l'output dei metodi HTTP abilitati

if metodi_abilitati:

    print(f"Metodi HTTP abilitati per {target}:")

    for metodo in metodi_abilitati:

        print(f"- {metodo}")
```

```
else:
    print(f"Nessun metodo HTTP supportato trovato per {target}.")
```

- Se ci sono metodi HTTP che risultano abilitati, questi vengono stampati uno dopo l'altro, mentre se non ne rislutano viene stampato un messaggio che indica che nessun metodo HTTP è abilitato dal server.

```
# Iniziare un loop per dare molteplici opzioni all'utente fin quando non decide di terminare il programma, mostrando un menù all'utente per scegliere un verbo HTTP basato sui metodi abilitati trovati
while True:
    print("\nScegli un verbo HTTP:")
    if 'GET' in metodi_abilitati:
        print("Digita 1 per GET;")
    if 'POST' in metodi_abilitati:
        print("Digita 2 per POST;")
    if 'PUT' in metodi_abilitati:
        print("Digita 3 per PUT;")
    if 'DELETE' in metodi_abilitati:
        print("Digita 4 per DELETE;")
    print("Digita 0 per terminare il programma.")
```

- Viene visualizzato un menù che permette all'utente di scegliere quale metodo HTTP utilizzare, basato sui metodi abilitati precedentemente trovati. Se un metodo è supportato, viene mostrata l'opzione corrispondente.

```
# Inserire la scelta dell'utente
    scelta = int(input("\nLa tua scelta: "))
```

- Input dell'utente: L'utente inserisce la propria scelta nel menù, convertendo il valore inserito in un intero e salvato nella variabile 'scelta'. python

```
# Eseguire l'operazione in base alla scelta dell'utente e ai metodi abilitati
    if scelta == 1 and 'GET' in metodi_abilitati:
        risposta = requests.get(f"http://{target}")
        print("\nRisposta web:", risposta.status_code)

    elif scelta == 2 and 'POST' in metodi_abilitati:
        risposta = requests.post(f"http://{target}")
        print("\nRisposta web:", risposta.status_code)
```

```
    elif scelta == 3 and 'PUT' in metodi_abilitati:
        risposta = requests.put(f"http://{target}")
        print("\nRisposta web:", risposta.status_code)

    elif scelta == 4 and 'DELETE' in metodi_abilitati:
        risposta = requests.delete(f"http://{target}")
        print("\nRisposta web:", risposta.status_code)

    elif scelta == 0:
        print("\nIl programma si termina.")
        break

    else:
        print("\nInput non valido. Riprova.")
```

- In base alla scelta dell'utente e ai metodi abilitati, viene inviata una richiesta HTTP corrispondente (GET, POST, PUT, DELETE), stampando la risposta web (status code).
- Se l'utente sceglie 0, viene stampato un messaggio di terminazione e il ciclo while si interrompe, chiudendo il programma.
- Infine, se l'utente sceglie un'opzione non valida viene stampato un messaggio di errore e l'utente viene invitato a riprovare.

```
# Creare un countdown di tre secondi prima di riproporre il menù
    for i in range(0, 3):
        print(".")
        time.sleep(1)
```

- Quest'ultimo blocco implementa un'iterazione che serve a fare una pausa di tre secondi tra la stampa dell'output richiesto nelll'ultima operazioni e la riproposta del menù, di modo da facilitare l'utilizzo del programma da parte dell'utente.

Progetto Bonus: Rilevatore socket di rete

Il lavoro svolto ha prodotto un programma volto a catturare e analizzare socket di rete in tempo reale visualizzando informazioni sulle connessioni di rete attive sul sistema.

Il programma inizia importando i moduli necessari quali 'socket', 'psutil', e 'struct', necessari rispettivamente per gestire le connessioni di rete, ottenere informazioni sulle connessioni di rete attive, e decodificare le instestazioni IP e TPC dei pacchetti di rete.

Successivamente si struttura definendo tre funzioni per gestire e analizzare i pacchetti di rete.

1. La funzione 'parse_ip_header': questa funzione si occupa di estrarre le informazioni dall'intestazione IP di un pacchetto, come la versione IP, la lunghezza dell'intestazione, il TTL (Time To Live), il protocollo, e gli indirizzi IP di origine e destinazione.

2. La funzione 'parse_tcp_header': questa funzione analizza l'intestazione TCP del pacchetto, estraendo le porte di origine e destinazione.

3. La funzione 'print_packet_info': questa funzione prende come oggetto le due precedenti funzioni di parsing per visualizzare le informazioni ottenute, stampando i dettagli sull'intestazione IP e TCP se il pacchetto è TCP, o in alternativa stampando solo i dettagli dell'intestazione IP.

Il programma poi recupera le connessioni di rete attive usando il modulo 'psutil', che fornisce un elenco dettagliato delle connessioni in corso, comprese le informazioni come indirizzi locali e remoti, stato della connessione, PID (Process ID), e nome del processo associato.

Successivamente, il codice crea un socket raw per catturare i pacchetti TCP e inizia un ciclo infinito per ricevere e analizzare i pacchetti. Ogni pacchetto ricevuto viene passato alla funzione di stampa, che visualizza le informazioni dettagliate sul pacchetto stesso. Inoltre, per ogni connessione attiva, viene stampata una riga con i dettagli della connessione, incluso il protocollo, l'indirizzo locale e remoto, lo stato della connessione, il PID, e il nome del processo associato.

Il programma si conclude infine gestendo le eccezioni in caso di interruzione da parte dell'utente tramite 'KeyboardInterrupt', chiudendo il socket e terminando l'esecuzione del programma in modo ordinato.

Perché utilizzare questo programma

L'utilizzo di questo programma è consigliato per mantenere la rete aziendale sicura, efficiente e ben gestita, facilitando la risoluzione dei problemi e il monitoraggio delle risorse di rete, in particolare:

1. Monitoraggio della rete: permette di monitorare il traffico di rete in tempo reale, identificando i pacchetti TCP e analizzando le connessioni di rete, aiutando così a diagnosticare problemi di rete e identificare traffico anomalo.

2. Gestione delle connessioni: fornendo un quadro dettagliato delle connessioni di rete attive, inclusi gli indirizzi IP e le porte, lo stato delle connessioni, e i processi che le utilizzano, è possibile capire quali applicazioni o servizi stanno utilizzando la rete e per gestire meglio le risorse di rete.

3. Sicurezza: monitorando i pacchetti di rete, il programma può aiutare a rilevare attività sospette o non autorizzate, come connessioni inusuali o traffico non previsto, migliorando la sicurezza della rete aziendale.
4. Analisi delle prestazioni: consente di analizzare le prestazioni della rete, monitorando il TTL dei pacchetti e altre metriche che possono indicare problemi di latenza o congestione.
5. Diagnosi dei problemi: in caso di problemi di rete, il programma fornisce informazioni dettagliate che possono essere utilizzate per il troubleshooting e per identificare la causa dei problemi.

Codice completo commentato

```
# Importare tutti i moduli necessari per l'esecuzione del programma
import socket
import psutil
import struct

# Creazione di una funzione parse_ip_header
def parse_ip_header(packet):
    # Estrazione semplificata dei campi specifici dell'intestazione IP
    ip_header = packet[:20]
    iph = struct.unpack('!BBHHHBBH4s4s', ip_header)
    version_ihl = iph[0]
    version = version_ihl >> 4
    ihl = version_ihl & 0xF
    ttl = iph[5]
    protocol = iph[6]
    src_ip = socket.inet_ntoa(iph[8])
    dest_ip = socket.inet_ntoa(iph[9])
    ip_header_length = ihl * 4

    # Restituire le informazioni elencate
    return version, ihl, ttl, protocol, src_ip, dest_ip,
ip_header_length

# Creazione di una funzione parse_tcp_header
def parse_tcp_header(packet, ip_header_length):

    # Estrarre i successivi 20 byte come intestazione TCP
    tcp_header = packet[ip_header_length:ip_header_length + 20]
    tcph = struct.unpack('!HHLLBBHHH', tcp_header)
    src_port = tcph[0]
    dest_port = tcph[1]

    # Restituire la porta sorgente e porta di destinazione
    return src_port, dest_port

# Creazione di una funzione print_packet_info
def print_packet_info(packet):
```

```
    version, ihl, ttl, protocol, src_ip, dest_ip, ip_header_length =
parse_ip_header(packet)

    # Controllare se il protocollo è TCP e stamapre relative
informazioni
    if protocol == 6:
        src_port, dest_port = parse_tcp_header(packet,
ip_header_length)
        print(f"IP Header:")
        print(f"  Version: {version}")
        print(f"  IHL: {ihl}")
        print(f"  TTL: {ttl}")
        print(f"  Protocol: TCP")
        print(f"  Source IP: {src_ip}")
        print(f"  Destination IP: {dest_ip}")

        print(f"TCP Header:")
        print(f"  Source Port: {src_port}")
        print(f"  Destination Port: {dest_port}")

    # Stampare le informazioni se il pacchetto non è TCP
    else:
        print(f"IP Header:")
        print(f"  Version: {version}")
        print(f"  IHL: {ihl}")
        print(f"  TTL: {ttl}")
        print(f"  Protocol: Other ({protocol})")
        print(f"  Source IP: {src_ip}")
        print(f"  Destination IP: {dest_ip}")

# Main script: recupero delle connesioni di rete
connessioni = psutil.net_connections(kind='inet')

# Stampare un'intestazione formattata per visualizzare le colonne delle
informazioni di rete
print(f"{'Protocol':<10} {'Local Address':<30} {'Remote Address':<30}
{'Status':<15} {'PID':<6} {'Process Name':<20}")

# Creare un socket per la cattura dei pacchetti
s = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_TCP)

# Creare un ciclo while per catturare pacchetti dal socket
try:
    while True:
        packet = s.recvfrom(65535)
        raw_packet = packet[0]  # Access the raw packet data
        print(f"Pacchetto ricevuto: {raw_packet}")
```

```
        print_packet_info(raw_packet)

        # Creare un'iterazione per le connessioni attive
        for connessione in connessioni:
            # Costruire l'indirizzo locale (laddr) e l'indirizzo remoto
(raddr)
            laddr = f"{connessione.laddr.ip}:{connessione.laddr.port}"
            raddr = f"{connessione.raddr.ip}:{connessione.raddr.port}"

            # Determinare il protocollo
            if connessione.type == socket.SOCK_STREAM:
                protocollo = "TCP"
            else:
                protocollo = "UDP"

            # Determinare lo stato e il PID della connessione
            status = connessione.status
            pid = connessione.pid

            # Stampare il nome del PID associato se disponibile
            try:
                nome_processo = psutil.Process(pid).name()
            except (psutil.NoSuchProcess, psutil.AccessDenied):
                nome_processo = "N/A"

            # Stampare le informazioni della connessione
            print(f"{protocollo:<10} {laddr:<30} {raddr:<30}
{status:<15} {pid:<6} {nome_processo:<20}")

# Gestire le eccezioni del blocco 'try'
except KeyboardInterrupt:
    print("Interrotto dall'utente.")
finally:
    s.close()
```

Esempio di output

```
Protocol   Local Address          Remote Address         Status       PID    Process Name

Pacchetto ricevuto:
b'E\x00\x004I\x7f@\x00m\x06\xac\x89\x14*AU\xc0\xa8\x01\x94\x01\xbb\xe2\x88)\x97\xf4z
\xe6\xe9\x995\x80\x10@\x01\x0cL\x00\x00\x01\x01\x08\n\x10\x98\x9e\xc1\x9d\xb5C0'

IP Header:

 Version: 4

 IHL: 5

 TTL: 109

 Protocol: TCP
```

```
 Source IP: 20.42.65.85
 Destination IP: 192.168.1.148
TCP Header:
 Source Port: 443
 Destination Port: 57992
UDP     192.168.1.148:68          192.168.1.1:67          NONE        603   NetworkManager
TCP     192.168.1.148:57992       20.42.65.85:443         ESTABLISHED 1635  code
```

Descrizione dettagliata del codice

```
# Importare tutti i moduli necessari per l'esecuzione del programma
import socket
```

- Questa prima linea importa tutte le funzioni dal modulo 'socket', che permette di creare e gestire connessioni di rete consentendo la comunicazione tra computer.

```
import psutil
```

- Il modulo 'psutil' serve per il monitoraggio di sistemi e processi e viene qui utilizzato per ottenere informazioni sulle connessioni e sui processi di rete.

```
import struct
```

- Il modulo 'struct' consente di lavorare con dati binari, e qui viene usato per decodificare le intestazioni di pacchetti di rete, attraverso la funzione 'unpack'.

```
# Creazione di una funzione 'parse_ip_header'
def parse_ip_header(packet):
```

- La funzione 'parse_ip_header(packet)' viene creata per estrarre e interpretare i campi dell'intestazione header di un pacchetto IP in formato raw. L'intestazione IP è la parte iniziale di un pacchettto di rete e contiene informazioni cruciali come indirizzo IP sorgente e destinazione, TTL, versione e tipo di protocollo utilizzato e altri paramentri di controllo.

```
#Estrazione semplificata dei campi specifici dall'intestazione IP
    ip_header = packet[:20]
    iph = struct.unpack('!BBHHHBBH4s4s', ip_header)
    version_ihl = iph[0]
    version = version_ihl >> 4
    ihl = version_ihl & 0xF
    ttl = iph[5]
    protocol = iph[6]
    src_ip = socket.inet_ntoa(iph[8])
```

```
    dest_ip = socket.inet_ntoa(iph[9])

    ip_header_length = ihl * 4
```

- La riga ‘packet[:20]’ estrae i primi 20 byte del pacchetto, che contengono l’intestazione IP.
- La riga ‘struct.unpack('!BBHHHBBH4s4s', ip_header)’ decodifica l’intestazione IP in una tupla di valori, con il formato '!BBHHHBBH4s4s' che definisce il layout del pacchetto IP (versione, lunghezza, TTL, protocollo, ecc.).
    - IHL – Internet Header Lenght: il primo byte contiene sia la versione del protocollo IP che la lunghezza dell'intestazione IP.
    - version = version_ihl >> 4: estrae i primi 4 bit per ottenere la versione del protocollo IP (IPv4).
    - ihl = version_ihl & 0xF: ottiene i secondi 4 bit per la lunghezza dell'intestazione IP.
    - TTL: il Time to Live è un contatore che limita la vita di un pacchetto sulla rete per evitare intasamenti.
    - protocol = iph[6]: contiene il numero del protocollo (TCP ha valore 6).
    - src_ip = socket.inet_ntoa(iph[8]): converte l'indirizzo sorgente IP in formato leggibile.
    - dest_ip = socket.inet_ntoa(iph[9]): converte l'indirizzo destinazione IP in formato leggibile.
    - ip_header_length = ihl * 4: calcola la lunghezza effettiva dell'intestazione IP in byte (ogni unità IHL è 4 byte).

```
# Restituire le informazioni elencate

return version, ihl, ttl, protocol, src_ip, dest_ip, ip_header_length
```

- return: Restituisce le informazioni estratte dall'intestazione IP.

```
# Creazione di una funzione ‘parse_tcp_header’

def parse_tcp_header(packet, ip_header_length):
```

- La funzione ‘parse_tcp_header()’ viene create per estrarre e interpretare i campi dell’intestazione header TCP da un pacchetto di rete una volta che l’intestazione IP è già stata elaborata. L’intestazione TCP contiene informazioni critiche come la porta sorgetne e la porta di destinazione, il numero di sequenza e conferma, il checksum, e altri dati che gestiscono il controllo della comunicazione tramite il protocollo TCP.

```
    # Estrazione semplificata dei successivi 20 byte come intestazione TCP

    tcp_header = packet[ip_header_length:ip_header_length + 20]

    tcph = struct.unpack('!HHLLBBHHH', tcp_header)

    src_port = tcph[0]

    dest_port = tcph[1]
```

```
    return src_port, dest_port
```

- ‘packet[ip_header_length:ip_header_length + 20]’ estrae i 20 byte successivi a quelli dell’intestazione IP, corrispondenti all’intestazione TCP.
- ‘struct.unpack('!HHLLBBHHH', tcp_header)’ decodifica l'intestazione TCP, estraendo informazioni come le porte sorgente e destinazione.
    - src_port: la porta sorgente da cui è stato inviato il pacchetto.
    - dest_port: la porta destinazione del pacchetto.

```
# Restituire le porta sorgente e porta di destinazione

return src_port, dest_port
```

- La funzione restituisce le porte sorgente e destinazione per permettere al programma principale di stampare o utilizzare queste informazioni.

```
# Creazione di una funzione ‘print_packet_info’

def print_packet_info(packet):
```

- La funzione ‘print_packet_info’ viene creata per estrarre, interpretare e stampare in maniera leggibile le informazioni principali contenute in un pacchetto di rete, sia a livello di instestazione IP che TCP.

```
# Estrazione dell’intestazione IP

version, ihl, ttl, protocol, src_ip, dest_ip, ip_header_length = parse_ip_header(packet)
```

- Questa riga serve per estrarre i campi fondamentali quali versione IP, lunghezza dell’intestazione, TTL, protocollo, IP sorgente e IP destinazione.

```
    # Controllare se il protocollo è TCP ed estrarne dettagli dell’intestazione

    if protocol == 6:

        src_port, dest_port = parse_tcp_header(packet, ip_header_length)
```

- Se il protocollo è TCP (numero 6) vengono estratti anche i dettagli dell’intestazione TCP attraverso la funzione ‘parse_tcp_header’.

```
    # Stampare le informazioni del pacchetto TCP:

        print(f"IP Header:")

        print(f"  Version: {version}")

        print(f"  IHL: {ihl}")

        print(f"  TTL: {ttl}")

        print(f"  Protocol: TCP")

        print(f"  Source IP: {src_ip}")

        print(f"  Destination IP: {dest_ip}")

        print(f"TCP Header:")

        print(f"  Source Port: {src_port}")

        print(f"  Destination Port: {dest_port}")
```

- Se il pacchetto è TCP vengono stampati i dettagli relativi sia all'intestazione IP che all'intestazione TCP.

```
# Stampare le informazioni se il pacchetto non è TCP
    else:
        print(f"IP Header:")
        print(f"  Version: {version}")
        print(f"  IHL: {ihl}")
        print(f"  TTL: {ttl}")
        print(f"  Protocol: Other ({protocol})")
        print(f"  Source IP: {src_ip}")
        print(f"  Destination IP: {dest_ip}")
```

- Se il pacchetto non è TCP, vengono stampati solo i dettagli dell'intestazione IP.

```
# Main script: recupero delle connessioni di rete
connessioni = psutil.net_connections(kind='inet')
```

- Questa riga utilizza il modulo 'psutil' per ottenere una lista di connessioni di rete attive. La funzione 'psutil.net_connections' con l'argomento 'kind=inet' restituisce una lista di connessioni di rete per i protocolli IP

```
# Stampare un'intestazione formattata per visualizzare le informazioni di rete
print(f"{'Protocol':<10} {'Local Address':<30} {'Remote Address':<30} {'Status':<15} {'PID':<6} {'Process Name':<20}")
```

- Stampa un'intestazione formattata per visualizzare le colonne delle informazioni delle connessioni.

```
# Creare un socket per la cattura dei pacchetti
s = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_TCP)
```

- Questa linea crea un socket con le seguenti caratteristiche:
    - socket.AF_INET: specifica che il socket sarà usato per gli indirizzi Ipv4;
    - socket.SOCK_RAW: crea un socket di tipo RAW che consente di accedere ai pacchetti di rete a livello di protocollo. I socket RAW sono comunemente utilizzati per attività come packet sniffig e analisi di rete;
    - socket.IPPROTO_TCP: indica che il socket cattura pacchetti TCP.

```
# Creare un ciclo while per catturare pacchetti dal socket
try:
```

- Il blocco 'try' serve per includere una parte di codice che potrebbe generare eccezioni di modo da correggerle tramite la funzione 'except'.

```
    while True:
```

- Questa riga crea un loop infinito in cui il programma continuerà ad eseguire quanto richiesto finché non esplicitamente interrotto.

```
packet = s.recvfrom(65535)
```

- Questa riga richiama al metodo 'recvfrom', che riceve un pacchetto di rete dal socket 's' (il paramentro 65535 specifica che i pacchetti sono ricevibili fino alla dimensione massima di 65535 bytes)

```
raw_packet = packet[0]  # Access the raw packet data
```

- Con questa riga viene estratto il pacchetto RAW dai dati ricevuti, cioè la parte effettiva contenente il pacchetto di rete.

```
print(f"Pacchetto ricevuto: {raw_packet}")
```

- Stampa il pacchetto RAW ricevuto in formato byte string, contenente i dati grezzi del pacchetto come intestazione IP e TCP.

```
print_packet_info(raw_packet)
```

- Questa riga chiama la funzione 'print_packet_info' con il pacchetto RAW come argomento, la quale estrae le informazioni dell'intestazione IP e le stampa in formato leggibile fornendo dettagli come:
    - versione IP;
    - lunghezza dell'intestazione;
    - TTL;
    - Protocollo di trasporto;
    - Indirizzo IP sorgente e destinazione;

```
# Creare un'iterazione sulle connessioni attive

for connessione in connessioni:

   laddr = f"{connessione.laddr.ip}:{connessione.laddr.port}"

   raddr = f"{connessione.raddr.ip}:{connessione.raddr.port}"
```

- Questa parte di codice itera sulle connessioni attive ottenute costruendo stringhe per gli indirizzi locali e remoti.

```
# Determinare il protocollo

   if connessione.type == socket.SOCK_STREAM:

      protocollo = "TCP"

   else:

      protocollo = "UDP"
```

- Questa parte del codice verifica il tipo di protocollo utilizzato dalla connessione di rete e assegna una stringa corrispondente TCP alla variabile 'protocollo' se la connessione corrente risulta vera per SOCK_STREAM, altrimenti esegue il blocco 'else' assegnando una stringa UDP.

```
# Estrarre lo stato e il PID della connessione

   status = connessione.status

   pid = connessione.pid
```

- Queste due righe di codice estraggono lo stato e il PID della connessione di rete

```
    # Stampare il nome del PID associato se disponibile
        try:
            nome_processo = psutil.Process(pid).name()
```

- Questa parte di codice tenta di ottenere il nome del processo associato a una connessione di rete usando il PID, gestendo eventuali eccezioni. psutil.Process(pid) crea un processo con l'ID fornito e consente di ottenere diverse informazioni sul processo come il nome, tramite il paramentro .name().

```
        except (psutil.NoSuchProcess, psutil.AccessDenied):
            nome_processo = "N/A"
```

- Il blocco 'except' intercetta due possibili eccezioni:
  - NoSuchProcess: quest'eccezione viene sollevata se il processo con quel PID non esiste più.
  - AccessDenied: quest'eccezione viene sollevata se il programma non ha i permessi necessari per accedere alle informazioni del processo.
- Se una delle eccezioni viene sollevata la variabile 'nome_processo' viene impostato su N/A.

```
# Stampare le informazioni delle connessioni
        print(f"{protocollo:<10} {laddr:<30} {raddr:<30} {status:<15} {pid:<6} {nome_processo:<20}")
#Gestire le eccezioni del blocco 'try'
except KeyboardInterrupt:
print("Interrotto dall'utente.")
```

- Questo blocco 'except' intercetta l'eccezione KeyboardInterrupt, che viene sollevata quando l'utente interrompe manualmente il programma, ed esegue il codice dentro lo stesso blocco, ossia stampare il messaggio "interrotto dall'utente".

```
finally:
    s.close()
```

- Questa ultima parte del codice viene eseguita sempre ('finally') chiudendo il il socket di modo da non occupare risorse di sitema o causando problemi con altre connessioni.