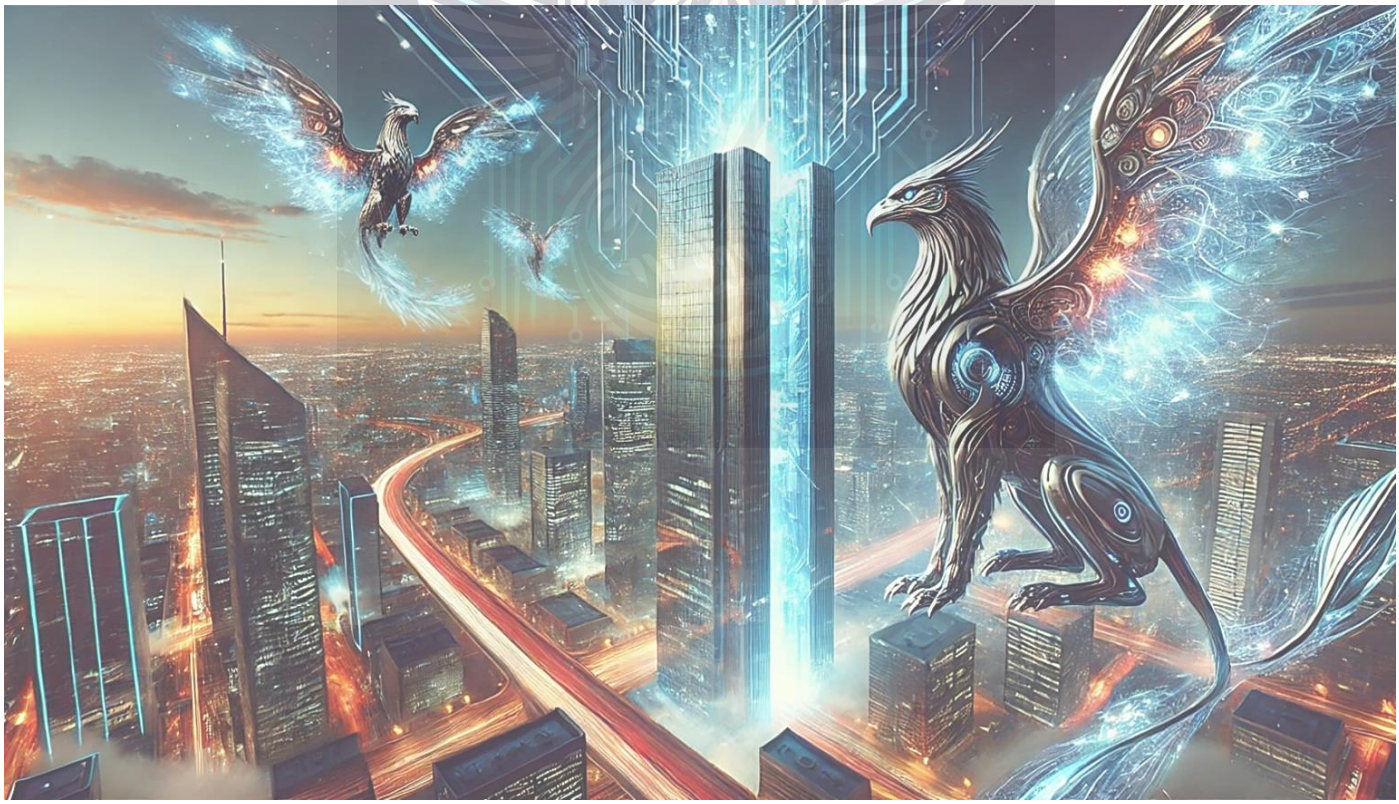
REPORT TECNICO SULLO SCRIPT RICHIESTE HTTP



LE RICHIESTE HTTP

Il codice utilizza diverse librerie python e componenti per effettuare richieste HTTP, salvare i log in file organizzati in una cartella

e gestire gli errori.

LIBRERIE UTILIZZATE:

1.os:

. Questa libreria è utile per interagire con il sistema operativo.

. Funzione utilizzata: os.makedirs()

. Serve per creare la directory (log_dir) in cui salva i file di log.

. L'opzione exist_ok=True garantisce che non venga generato un errore se la directory esiste già

2. requests

. Una libreria molto versatile per effettuare richieste HTTP.

.Funzioni utilizzate:

.requests.get(url),requests.post(url, data), ecc vengono utilizzate per inviare le richieste HTTP.

.Ogni metodo restituisce un oggetto "response", che contiene informazioni come il codice di stato(status_code), la motivazione della risposta (reason)

e gli headers della risposta (headers).

3. datetime

.Permette di lavorare con date e orari

.Funzione utilizzata:datetime.now().strftime()

.Usata per ottenere un timestamp univoci utilizzando data e ora (es. 20241128_101234) che serve per creare nomi univoci per i file di log.

la documentazione riguardante le librerie si può trovare ai seguenti indirizzi:

.os :https://docs.python.org/3/library/os.html#module-os

. requests:https://www.programmareinpython.it/blog/requests-http-gli-esseri-umani-tutorial-python-ita/

. datetime: https://docs.python.org/3/library/datetime.html#module-datetime

STRUTTURA DEL CODICE:

impostazioni della configurazione:

```
# URL del web server da testare
url = input("http://<indirizzo-web-server>: ")

# Lista dei verbi HTTP da testare, inclusi OPTIONS
http_methods = ["OPTIONS", "GET", "POST", "PUT", "DELETE"]
```

.url: chiede all'utente l'indirizzo del server da testare.

.http_methods: contiene i metodi HTTP che lo script testerà uno per uno

Creazione della cartella per i log:

```
# Creazione della cartella dei log
log_dir = "http_requests_logs"
os.makedirs(log_dir, exist_ok=True)  # Crea la cartella se non esiste
```

.log_dir: nome della cartella in cui saranno salvati i log.

.os.makedirs(log_dir, exist_ok=True):

.Se la cartella http_requests_logs non esiste la crea.

.Con exist_ok=True, non genera errori se la cartella esiste già

Creazione del file di log:

Un log è un registro cronologico che documenta eventi, attività o stati di un sistema/applicazione/rete.

Nel nostro caso, la creazione dei file di log è utile per analizzare le risposte delle nostre scansioni in un secondo momento, se necessario.

```
# Creazione del file di log con timestamp
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
log_file_path = os.path.join(log_dir, f"http_requests_log_{timestamp}.txt")

# Creazione del file di log per questa sessione
with open(log_file_path, "w") as log_file:
    log_file.write("== LOG DELLE RICHIESTE HTTP ==\n")
    log_file.write(f"URL testato: {url}\n\n")
```

timestamp: genera una stringa unica basata sulla data e ora attuali

log_file_path: combina il nome della cartella e il nome unico del file

with open(log_file_path, "w")as log_file:

.apre il file in modalità scrittura ("w")

.scrive un'intestazione per indicare che si tratta di un log di richieste HTTP.

Esecuzione delle richieste HTTP:

procediamo alla creazione di un ciclo for per effettuare le varie richieste HTTP:

```
for method in http_methods:
    try:
        # Invia la richiesta corrispondente al metodo
        if method == "GET":
            response = requests.get(url)
        elif method == "POST":
            response = requests.post(url, data={"key": "value"})
        elif method == "PUT":
            response = requests.put(url, data={"key": "value"})
        elif method == "DELETE":
            response = requests.delete(url)
        elif method == "OPTIONS":
            response = requests.options(url)
```

- un ciclo for esegue ogni metodo HTTP specificato nella lista http_methods.
- In base al metodo utilizzato (GET,PUT, ecc) viene invocata la funzione corrispondente della libreria requests come precedentemente già spiegato

(es. per la richiesta GET sarà: requests.get(url))

- data={"key": "value"}: per i metodi POST e PUT, invia dei dati fittizzi al server come payload.

Salvataggio dei Risultati:

Procediamo al salvataggio dopo aver preparato il contenuto del log:

```
# Preparazione del contenuto del log
log_content = f"Metodo: {method}\n"
log_content += f"Status Code: {response.status_code}\n"
log_content += f"Reason: {response.reason}\n"
log_content += f"Headers: {response.headers}\n"
if method == "OPTIONS":
    log_content += f"Allowed Methods: {results[method]['allowed_methods']}\n"
log_content += "-" * 50 + "\n"

# Aggiunta al file di log unico
with open(log_file_path, "a") as log_file:
    log_file.write(log_content)
```

- Preparazione del log:

Crea una stringa (log_content) con i dettagli della risposta: metodo HTTP, codice di stato (status_code), motivo (reason) e intestazioni della risposta (headers).

Se il metodo è OPTIONS, aggiunge i metodi consentiti recuperati dall'header Allow.

- Salvataggio del log:

Apre il file di log in modalità append ("a"), così non sovrascrive i dati esistenti.

Aggiunge il contenuto della richiesta (log_content) al file.

Gestione degli errori:

```
except Exception as e:
    # Gestione degli errori
    error_content = f"Metodo: {method}\nErrore: {str(e)}\n"
    error_content += "-" * 50 + "\n"
    results[method] = {"error": str(e)}

    # Aggiunta dell'errore al file di log unico
    with open(log_file_path, "a") as log_file:
        log_file.write(error_content)
```

- Blocco try-except:

L'implementazione del codice per la gestione degli errori è fondamentale all'interno dello script poiché se qualcosa va storto durante una richiesta, esso ne cattura l'eccezione e salva un messaggio di errore (error_content) nel file di log.

OUTPUT finale:

```
# Output finale nella console
print(f"Log salvato in: {log_file_path}")
print(f"I file di log sono salvati nella cartella: {log_dir}")
```

Il comando "print" avvisa l'utente sulla posizione in cui sono stati salvati i file di log.

Cosa succede in caso di errori:

Se un metodo HTTP fallisce ad esempio se il serve non supporta quel metodo lo script:

1. registra un messaggio di errore nel file di log con il relativo codice di stato

2.continua a processare i metodi successivi senza interrompersi.

Lo script è utile per analizzare i comportamenti di un server HTTP rispetto ai diversi metodi HTTP supportati.

La gestione dei log è strutturata in modo da fornire un report completo di ogni sessione e permette all'utente di salvare le risposte ricevute dalle sue scansione

per poterle ri-utilizzare in un secondo momento se si ha la necessità.

CONSIDERAZIONI FINALI

Attraverso l'uso combinato tra lo script http ed il port scanner, possiamo ottenere una visione dettagliata sia della configurazione del server che della sicurezza dell'intera rete.

Questo approccio non solo facilita il rilevamento di configurazioni errate o di vulnerabilità ma, considerando che le minacce che potrebbero provenire dalla rete esterna siano in continua evoluzione, ci aiuta anche nel comprendere meglio possibili rischi futuri legati all'architettura di rete dell'azienda.