Relazione sul Codice per la Verifica dei Verbi HTTP

Obbiettivo:

Verifica dei Verbi HTTP Scriveremo un programma in Python per inviare richieste HTTP GET, POST, PUT, DELETE) al web server e verificare le risposte.

Descrizione generale:

Questo programma sarà uno strumento di ricognizione web automatizzata con lo scopo di interrogare un server web specifico per determinare quali azioni o metodi HTTP sono consentite in una determinata risorsa.

Definizione del target :

Relazione sul funzionamento del programma Python con l'obiettivo di andare a verificare quali verbi sono ammessi e quali no su un particolare Target.
Il target sarà composto da 2 parti :

- **Base URL** (e.g., http://192.168.50.101/):
- **Path** (e.g., /dvwa/ or /phpMyAdmin/):

Analisi del codice:

Librerie Utilizzate

```
import requests
from urllib.parse import urljoin
```

Import di queste due librerie :

- **Requests**
  - Usata per inviare richieste HTTP al server e ricevere determinate risposte

- **Urljoin**

- Usata specificamente per la funzione urljoin, utile per unire correttamente l'URL di base e il percorso (path) senza errori di formattazione

Preparazione e Sanitizzazione dell'URL

```python
def test_methods(base_url: str, path: str):
    url = urljoin(base_url.rstrip("/") + "/", path.lstrip("/"))

    methods = ["OPTIONS", "GET", "POST", "PUT", "DELETE"]
    results = {}
```

Questa è la fase di preparazione della funzione in cui il programma si assicura che l'indirizzo in input sia corretto e si preparano tutti gli strumenti necessario per la fase successiva.

Nella riga: *"url = urljoin(base_url.rstrip("/") + "/", path.lstrip("/"))"* si vanno ad individuare e correggere eventuali errori comuni di formattazione come la mancanza degli *"/"* all'inizio dell'URL o del Path.

Definizione dei Metodi e Payload

Come si può vedere sono definite 4 metodi HTTP che verranno testati:

- OPTIONS
- GET
- POST
- PUT
- DELETE

Lo script userà questa lista per eseguire un ciclo e provare questi metodi uno per uno.
In questa determinata funzione si esegue la scansione vera e propria dell'URL target

Ricognizione Iniziale (Metodo OPTIONS)

```
try:
    r = requests.options(url, timeout=5, allow_redirects=False)
    results["OPTIONS"] = (r.status_code, r.headers.get("Allow", ""), r.headers.get("Server", ""))
except requests.RequestException as e:
    results["OPTIONS"] = ("ERR", str(e), "")

payload = {"test": "1"}
```

Primo tentativo di contatto con il server, tramite infatti il metodo *“OPTIONS”* si va a esplorare chiedendo al server quali metodi sono abilitati su questo URL, restituendo in molti casi l’Header con i metodi permessi *(Allow)*.

Nel caso in cui invece la pagina venga reindirizzata su un'altra lo script avrà istruzione di non seguirla poichè noi vogliamo che l’analisi venga fatta sull’URL inserito e non su un’altro.

Il tutto verrà salvato in una tupla con 3 informazioni:

- Lo stato della risposta;
- L’header Allow
- Il software del server usato

La riga del payload invece è un piccolo pacchetto di dati finti che servirà per verificare se il server accetta l’invio di dati oppure no.

Ciclo di Scansione Automatica

```
for m in ["GET", "POST", "PUT", "DELETE"]:
    try:
        if m == "GET":
            r = requests.get(url, timeout=5, allow_redirects=False)
        elif m == "POST":
            r = requests.post(url, data=payload, timeout=5, allow_redirects=False)
        elif m == "PUT":
            r = requests.put(url, data=payload, timeout=5, allow_redirects=False)
        elif m == "DELETE":
            r = requests.delete(url, timeout=5, allow_redirects=False)

        results[m] = (r.status_code, r.headers.get("Allow", ""), r.headers.get("Server", ""))
    except requests.RequestException as e:
        results[m] = ("ERR", str(e), "")
```

Arriviamo al core del programma, tramite questo ciclo ***"for"*** si automatizza il processo andando a raccogliere la lista dei metodi inizializzata sopra ed eseguirla. La variabile ***"m"*** cambia in base al valore della lista.

Essendo metodi differenti non tutti porteranno il payload inizializzato prima, ad esempio il *"GET"* serve solo per leggere quindi senza il trasporto dei dati mentre il post servirà a mandare dei dati usando quindi il payload.

Inoltre è presente anche un *"except"* utile per gestire i vari casi di errore.
Infine tutti i risultati vengono salvati nel dizionario ***"results"*** usando ***"m"*** come chiave.

Analisi dei Risultati e Logica di Validazione

```
print(f"\nTarget: {url}")
print("-" * 70)
for method, (code, allow, server) in results.items():
    allow_txt = f" Allow={allow}" if allow else ""
    server_txt = f" Server={server}" if server else ""
    print(f"{method:7} -> {code}{allow_txt}{server_txt}")


enabled = []
for method, (code, _, _) in results.items():
    if code == "ERR":
        continue

    if code != 405:
        enabled.append(method)

print("\nLikely handled/enabled methods:", ", ".join(enabled) if enabled else "None / unclear")
```

Questo è il blocco finale cioè la fase di **reporting** e **analisi**, andando a presentare i dati di output in 2 parti: la parte tabellare e l'analisi logica.

L'ultimo print va a verificare come la variabile *"Allow"* contenga dati oppure no.
Se la variabile contiene dati si andranno a visualizzare nell'output altrimenti si lascia la stringa vuota.

L'analisi logica invece (seconda parte dello screen) va a individuare quali metodi sono effettivamente attivi.

L'if è la parte fondamentale poiché va a verificare se il metodo è consentito oppure no.
La logica dell'if e che **se il codice è diverso da 405 allora probabilmente il metodo è abilitato**.
Il tutto poi viene stampato in una frase leggibile.

Interfaccia Utente e Entry Point

```python
def main():
    base_url = input("Base URL (e.g., http://192.168.50.101/): ").strip()
    path = input("Path (e.g., /dvwa/ or /phpMyAdmin/): ").strip()

    if not base_url:
        print("Base URL is required.")
        return

    if not path:
        path = "/"

    test_methods(base_url, path)

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

Questo blocco finale indica l'interfaccia utente utile per dare un output leggibile, si mettono insieme le varie funzioni e si dà la possibilità di interagire con l'utente.

La funzione *"main()"* gestisce il dialogo e prepara i dati prima di passarli al core del programma

L'*If __name__ == "__main__"*: è la chiave di accensione del programma.

Output del programma:

```
Target: http://192.168.1.102/phpMyAdmin/themes/original/img/logo_right.png
----------------------------------------------------------------------
OPTIONS -> 200 Allow=GET,HEAD,POST,OPTIONS,TRACE Server=Apache/2.2.8 (Ubuntu) DAV/2
GET     -> 200 Server=Apache/2.2.8 (Ubuntu) DAV/2
POST    -> 200 Server=Apache/2.2.8 (Ubuntu) DAV/2
PUT     -> 405 Allow=GET,HEAD,POST,OPTIONS,TRACE Server=Apache/2.2.8 (Ubuntu) DAV/2
DELETE  -> 405 Allow=GET,HEAD,POST,OPTIONS,TRACE Server=Apache/2.2.8 (Ubuntu) DAV/2

Likely handled/enabled methods: OPTIONS, GET, POST
```

Nell'output del programma possiamo vedere :

- Target
- Risultati dettagliati:
    - **OPTIONS** -> 200 (successo);
    - **Metodi accettati**;
    - **Nome e Software del server**, utile per ulteriori analisi.