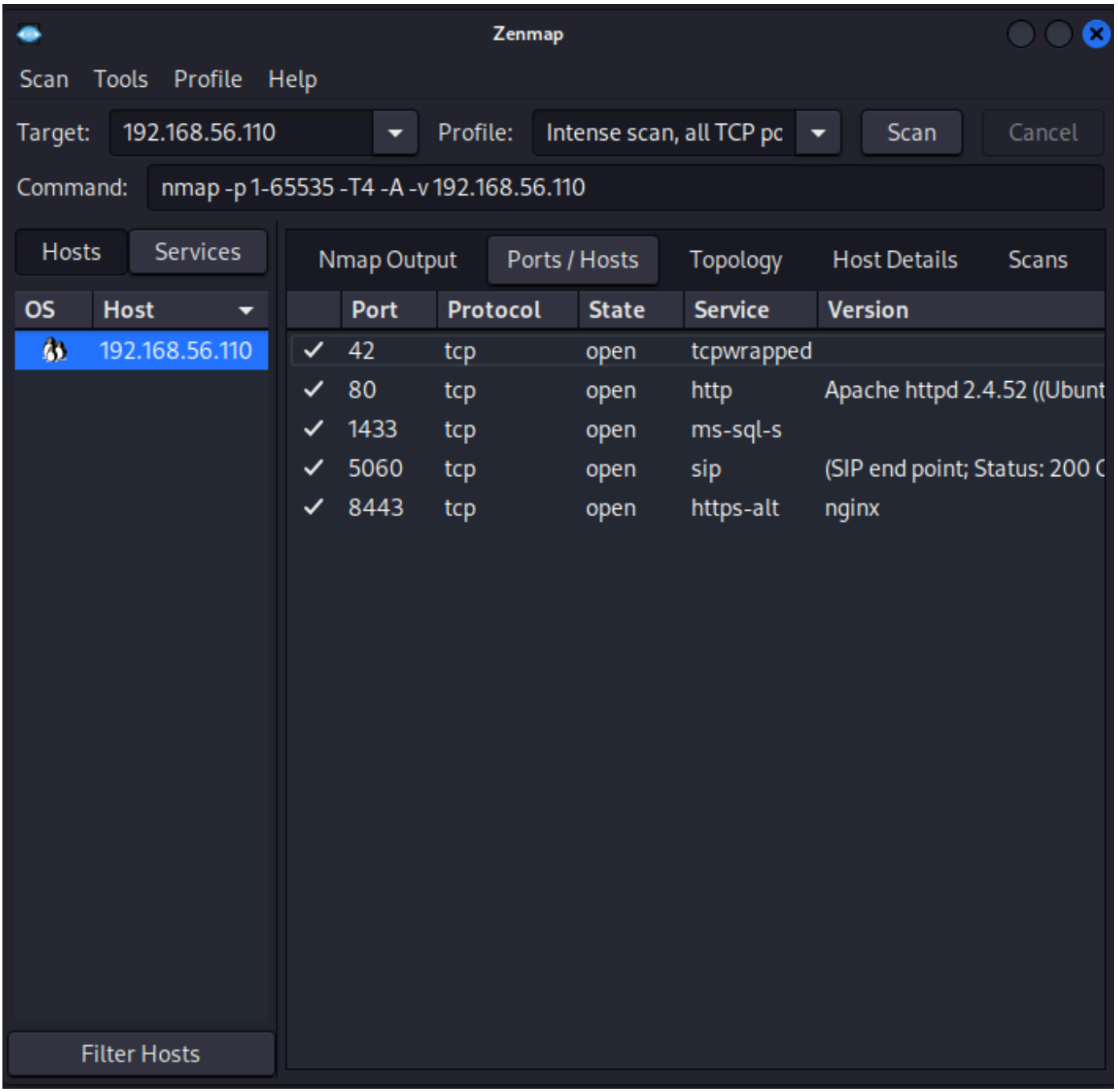
# Epicode - Harry P.

## Introduzione

In questo esercizio, abbiamo affrontato la compromissione di una macchina virtuale a tema Harry Potter, progettata per testare le capacità di penetration testing e analisi forense. La macchina, denominata Theta, è stata sabotata da un dipendente infedele di nome Luca, che ha modificato configurazioni critiche e introdotto vulnerabilità nel sistema. L'obiettivo era riprendere il controllo del server compromesso, analizzando servizi esposti, vulnerabilità web, e utilizzando tecniche avanzate come SQL Injection, steganografia e decodifica Brainfuck. Durante l'esercizio, è stato necessario enumerare porte e servizi, eseguire attacchi su specifici vettori di vulnerabilità e sfruttare correlazioni tra dati ottenuti per ricostruire l'accesso completo alla macchina con privilegi amministrativi.

### Fase di Information Gathering con Zenmap

Per iniziare abbiamo tentato una scansione con Zenmap




ed abbiamo proceduto come da prassi.

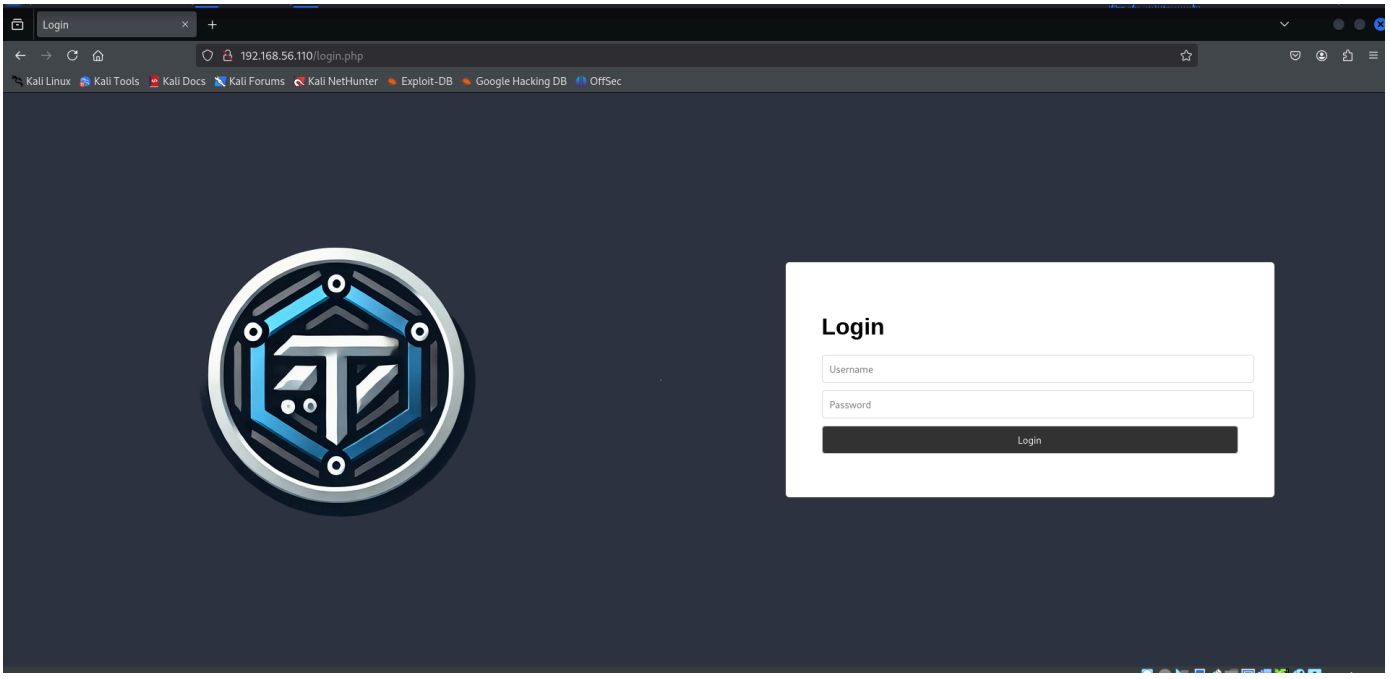
### Esplorazione Iniziale del Sito Web

Dopo aver identificato la porta 80 aperta durante la scansione con Nmap, abbiamo avviato un'analisi manuale del sito web ospitato sulla macchina target.

1. Visito l'url:

`http://192.168.56.110/login.php`

Abbiamo aperto il browser e osservato la pagina di login disponibile su questa porta.



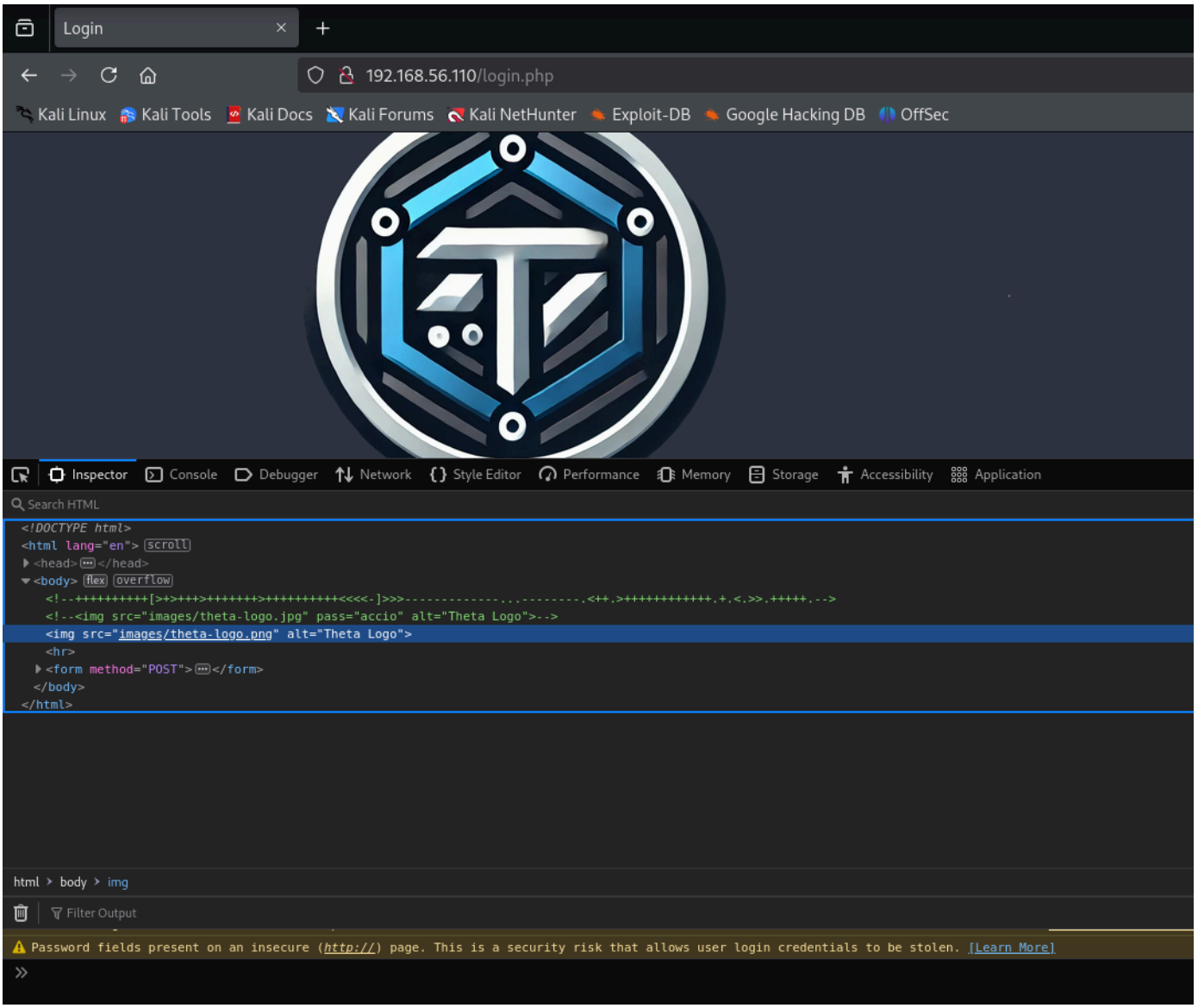

# Risultati dell’Analisi

## Struttura del Sito:

- La pagina principale presenta un classico form di login, con campi per username e password, suggerendo la presenza di un sistema di autenticazione.
- Dal punto di vista visivo, non sono state notate funzionalità o elementi particolari che potrebbero fornire ulteriori informazioni.
  Codice Sorgente:
- Analizzando il codice sorgente della pagina (view-source), abbiamo trovato commenti che contenevano un codice Brainfuck. Questo indizio ha diretto la nostra attenzione verso una possibile stegano-analisi o ulteriori decodifiche.

## Analisi dei Commenti e Decodifica del Brainfuck

Proseguendo nell’esplorazione del sito, abbiamo analizzato il codice sorgente della pagina web utilizzando view-source. Nei commenti del codice abbiamo individuato un’informazione significativa: una stringa di codice Brainfuck.




## Codice Brainfuck

Il commento conteneva il seguente codice:

```
++++++++++[>+>+++>+++++++>++++++++++<<<<-]>>>------------.,.--------.
<++.>++++++++++++.+.<.>>.+++++.+++.
```

Abbiamo decodificato la stringa utilizzando il sito decode.fr, ottenendo il risultato:

**9991 → di**

Questo valore si è rivelato una parte di un possibile schema numerico correlato ad altre informazioni trovate nel sito.

Altri Indizi nei Commenti
Nel codice sorgente, abbiamo trovato un ulteriore commento:

```
<!-- <img src="images/theta-logo.jpg" pass="accio" alt="Theta Logo"> -->
```

Il commento suggeriva l'esistenza di un'immagine (theta-logo.jpg) protetta da una passphrase, presumibilmente "accio".

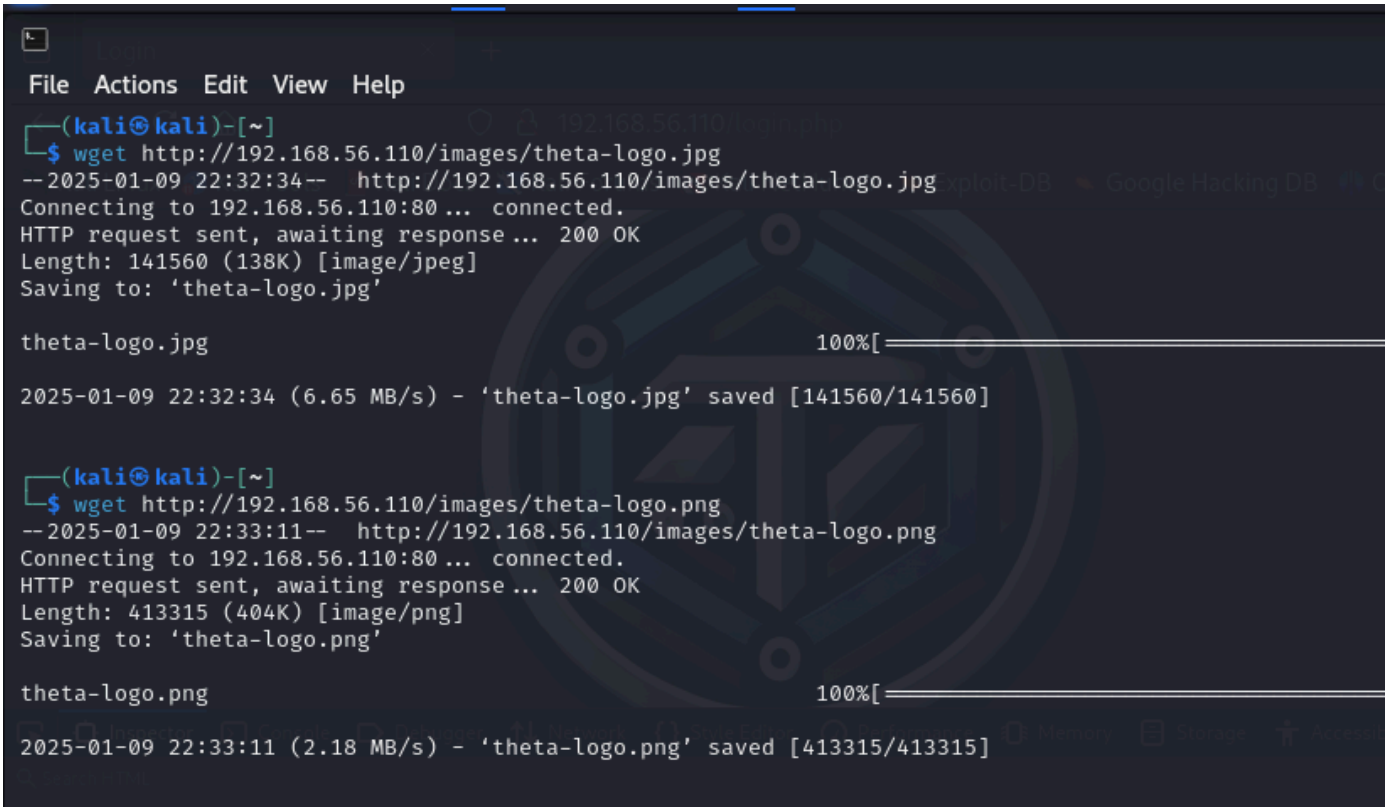
### Download delle Immagini

Seguendo questo indizio, abbiamo utilizzato il comando wget per scaricare le immagini menzionate nel sito:

```
wget http://192.168.56.110/images/theta-logo.jpg

wget http://192.168.56.110/images/theta-logo.png
```

Questo passaggio ci ha permesso di ottenere i file necessari per un'analisi successiva, indicando che erano stati nascosti ulteriori indizi o informazioni critiche nelle immagini.




## Estrazione di Informazioni con Steghide

Dopo aver scaricato le immagini theta-logo.jpg e theta-logo.png, abbiamo utilizzato Steghide, un tool specifico per l'analisi di file contenenti dati nascosti tramite tecniche di steganografia.

### Comandi Utilizzati

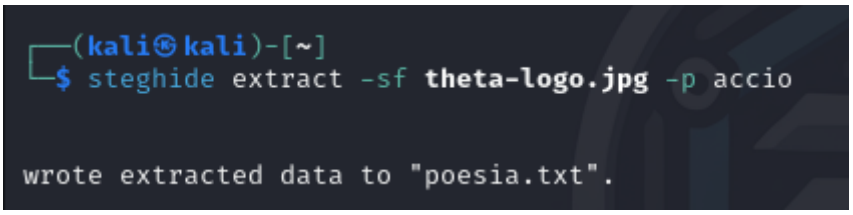
Per analizzare le immagini, abbiamo utilizzato i seguenti comandi:

```
steghide extract -sf theta-logo.jpg -p "accio"
```

e

`steghide extract -sf theta-logo.png -p "accio"`

Entrambi i comandi indicano a Steghide di estrarre eventuali dati nascosti nei file specificati. La passphrase accio, trovata nei commenti del codice sorgente della pagina web, è stata utilizzata per decrittare i dati nascosti.



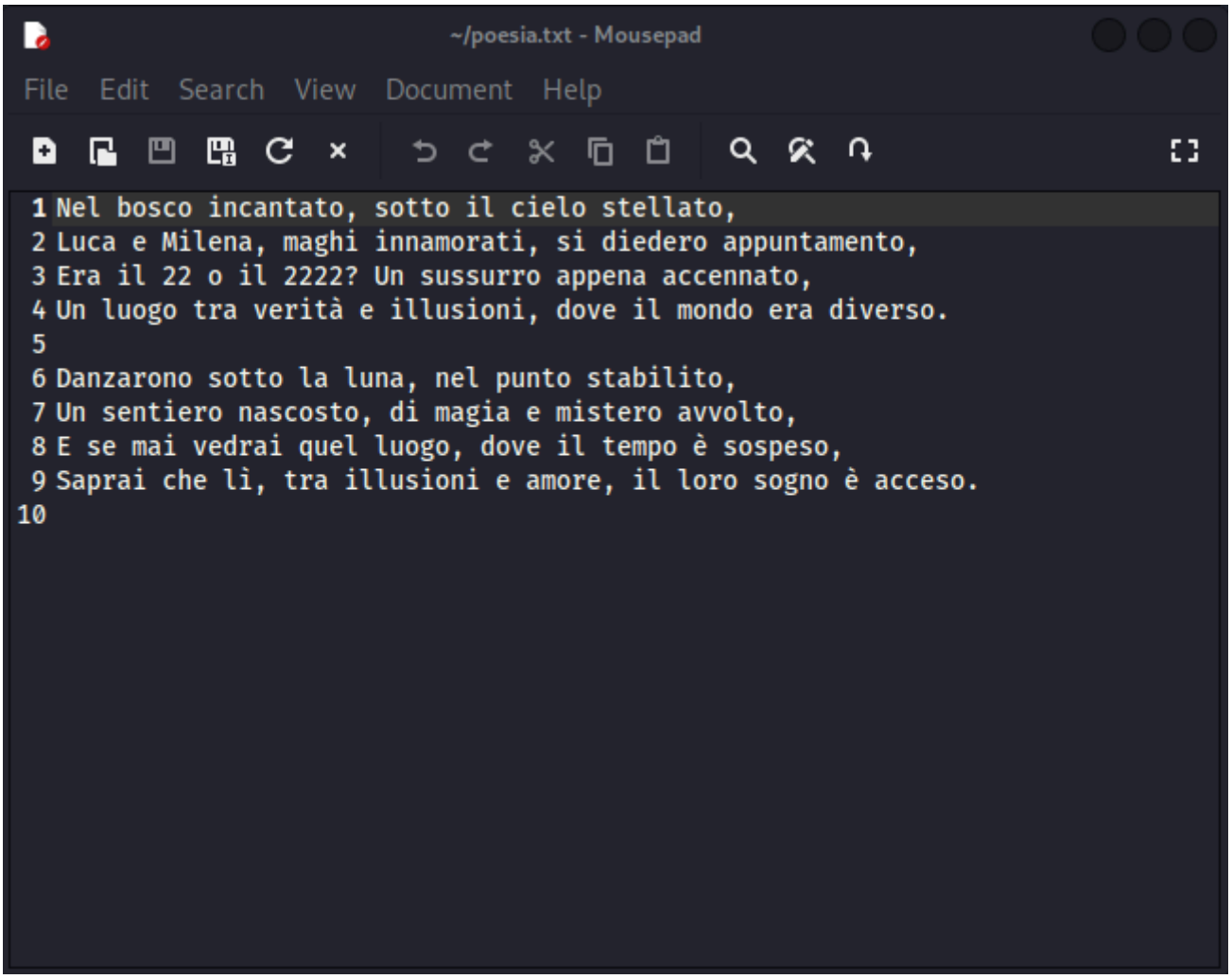

### Risultati

**Immagine theta-logo.jpg**:

- Steghide ha rilevato un file nascosto e lo ha estratto correttamente.
- Il file estratto si chiamava poesia.txt.
  Immagine theta-logo.png:
- Non sono stati trovati dati nascosti o il formato non era supportato per l'estrazione.

### Esame del File Estratto

Dopo l'estrazione, abbiamo analizzato il contenuto del file `poesia.txt` utilizzando il comando cat. Questo file conteneva ulteriori indizi rilevanti per il proseguimento dell'esercizio, confermando l'utilità della steganografia per nascondere informazioni critiche.




Abbiamo utilizzato Gobuster, uno strumento di enumerazione delle directory, per cercare percorsi nascosti o directory non documentate nel sito web ospitato su http://192.168.56.110. Questo tipo di analisi è fondamentale per individuare file o directory potenzialmente interessanti che potrebbero contenere vulnerabilità, credenziali, o informazioni sensibili.

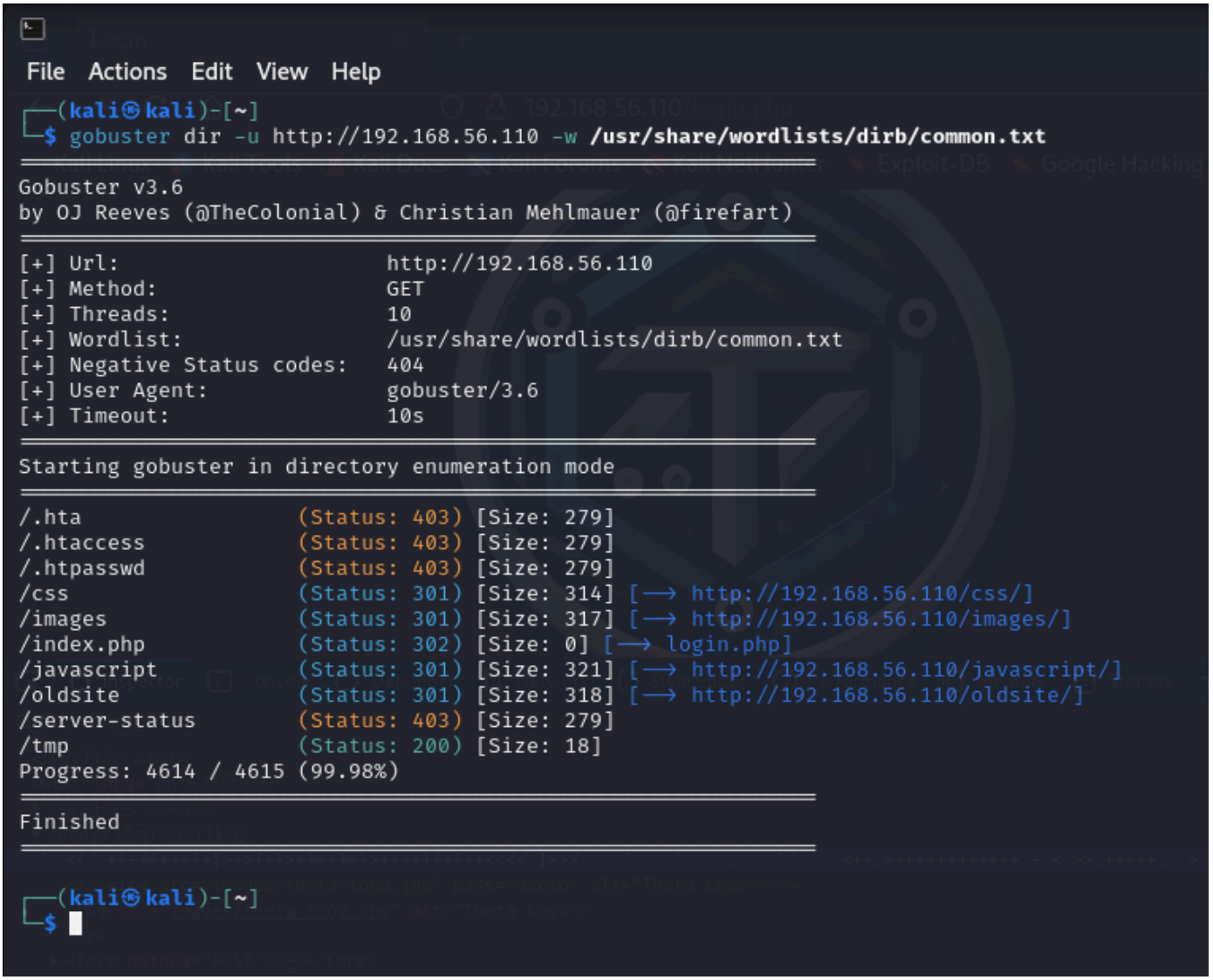
Il comando eseguito è stato:

```
gobuster dir -u http://192.168.56.110 -w /usr/share/wordlists/dirb/common.txt
```

● -u http://192.168.56.110: Specifica l'URL del target, ovvero l'indirizzo IP del sito da analizzare.
● -w /usr/share/wordlists/dirb/common.txt: Indica il percorso del dizionario utilizzato. Questo file contiene una lista di nomi di directory comuni da testare.
Risultati
Gobuster ha identificato diverse directory e file interessanti, tra cui:

● /css: Contiene file CSS, probabilmente per la formattazione del sito.
● /images: Directory di immagini, utile per analisi steganografiche o di metadati.
● /javascript: Contiene file JS, potenzialmente utili per analizzare la logica del client-side.
● /oldsite: Una directory particolarmente interessante che potrebbe contenere una versione precedente del sito web, spesso più vulnerabile.
● /tmp: Una directory temporanea, possibile deposito di file utili o sensibili.

Questa scansione ha ampliato il nostro campo d'azione, fornendoci nuovi percorsi da esplorare per ulteriori analisi e potenziali punti d’ingresso.




Per esplorare ulteriormente i percorsi trovati con Gobuster, abbiamo aperto la directory /oldsite nel browser utilizzando l'URL:

`http://192.168.56.110/oldsite/login.php`

### Descrizione

Questa directory conteneva una pagina di login, simile a quella presente nella directory principale del sito. Tuttavia, trattandosi di una versione precedente del sito ("oldsite"), potrebbe contenere vulnerabilità non ancora corrette. Le versioni legacy dei siti web spesso non vengono aggiornate regolarmente, rappresentando un target ideale per attacchi.

### Osservazioni

Accedendo alla pagina /oldsite/login.php, abbiamo trovato una schermata di login identica a quella della directory principale, ma con la possibilità di testare vulnerabilità come SQL Injection o altre tecniche d'attacco. Questa scoperta ci ha fornito un nuovo terreno per tentare attacchi manuali e sfruttare eventuali debolezze specifiche di questa versione del sito.

Per analizzare il codice sorgente della pagina di login nella directory /oldsite, abbiamo utilizzato il comando:

```
view-source:firefox http://192.168.56.110/oldsite/login.php
```

### Descrizione

Questo comando permette di visualizzare il codice sorgente HTML di una pagina web, un passaggio essenziale durante l'enumerazione e l'analisi di vulnerabilità. Esaminare il codice sorgente aiuta a individuare commenti nascosti, riferimenti a file, parametri significativi o altre informazioni utili che potrebbero essere sfruttate in attacchi futuri.

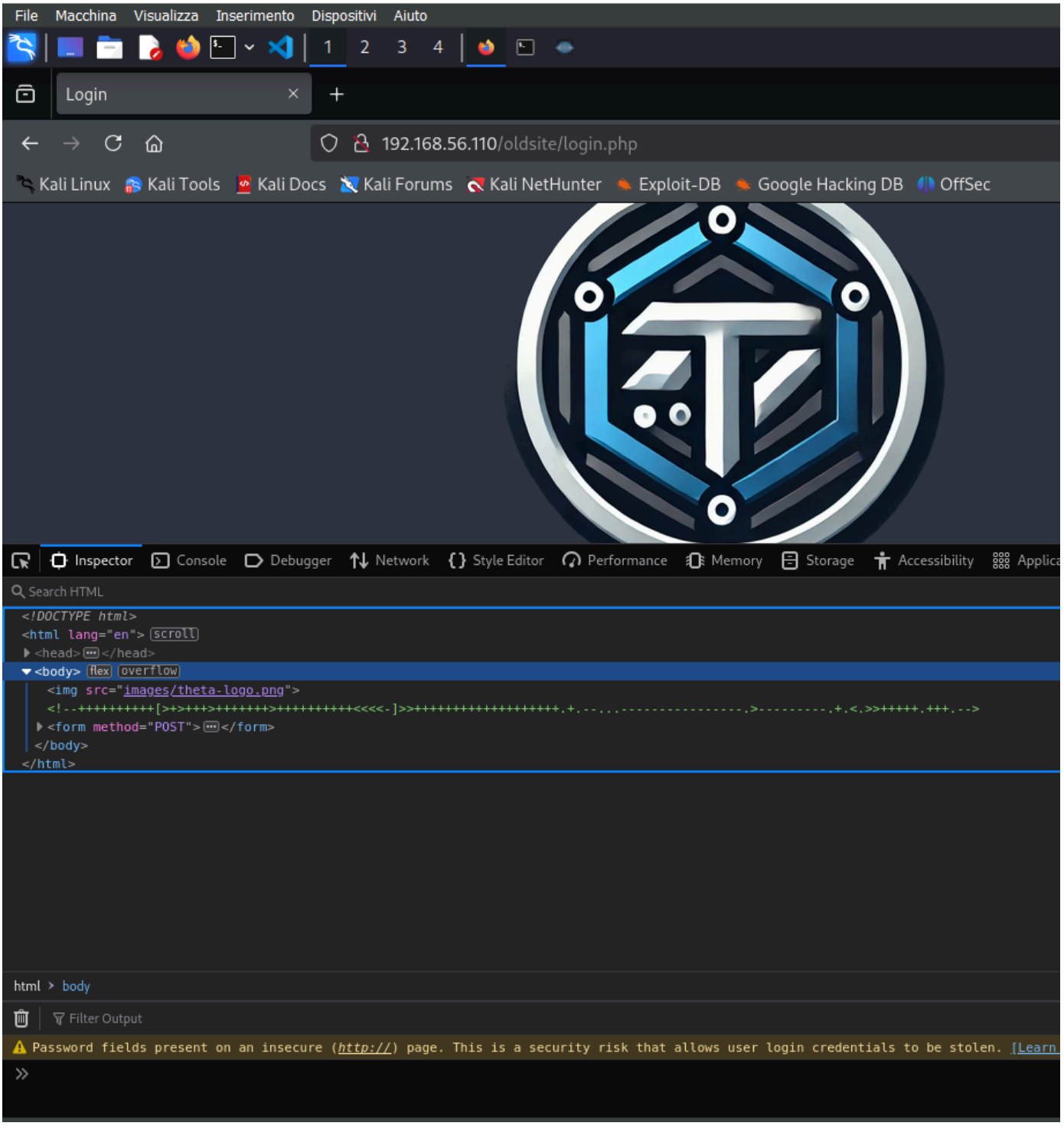
Utilizzando il comando, è possibile accedere alla struttura completa della pagina, incluse le informazioni che non sono immediatamente visibili dall'interfaccia utente. Questi dettagli possono contenere indizi critici, come credenziali, chiavi di accesso o codice potenzialmente sfruttabile.

Nel codice sorgente della pagina /oldsite/login.php, abbiamo individuato un commento contenente un frammento di codice scritto nel linguaggio Brainfuck:

```
++++++++++[>+>+++>+++++++>++++++++++<<<<-]>>++++++++++++++++++.+.--...----------
---.>---------.+.<.>>+++++.+++.
```

### Descrizione

Il linguaggio Brainfuck è un linguaggio esoterico basato su un sistema di puntatori e comandi minimalisti, spesso utilizzato per nascondere informazioni o per sfide tecniche. La presenza di questo codice nei commenti è un indizio che potrebbe contenere un messaggio o una chiave utile per il proseguimento dell’esercizio.




Per decodificare il messaggio, abbiamo utilizzato un decodificatore online, come decode.fr, che consente di interpretare e tradurre Brainfuck in testo leggibile.

### Risultato

Il codice è stato decodificato con successo, restituendo il seguente valore:

12000 => il

### Analisi

Il valore decodificato "12000 => il" suggerisce che il numero potrebbe essere correlato a una porta di rete o a un'informazione chiave. La parola "il" può far parte di un messaggio più ampio, il che implica

che ulteriori indizi saranno necessari per ricostruire il significato completo.

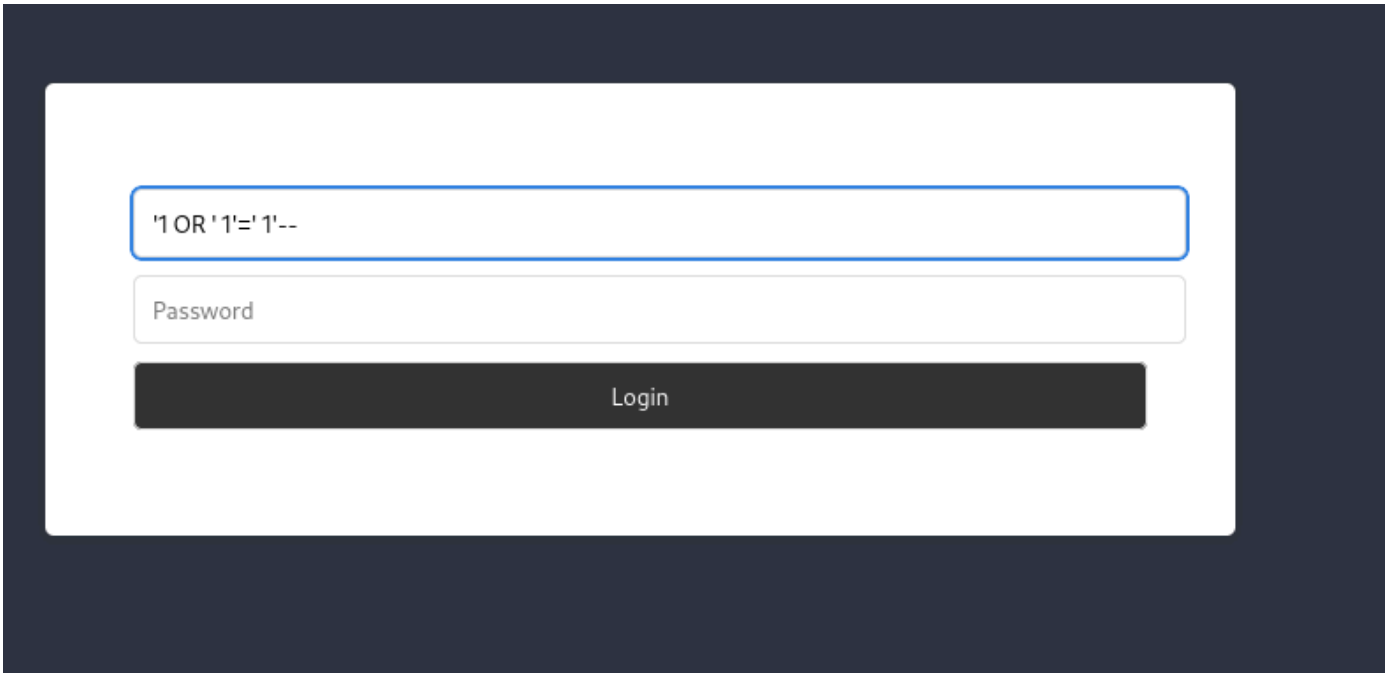
Dopo aver individuato una potenziale vulnerabilità nella pagina /oldsite/login.php, abbiamo tentato di eseguire un attacco di SQL Injection. L'obiettivo era verificare se il sistema fosse vulnerabile, sfruttando un payload standard:

### Comando Utilizzato

```
' OR '1' = '1' --
```

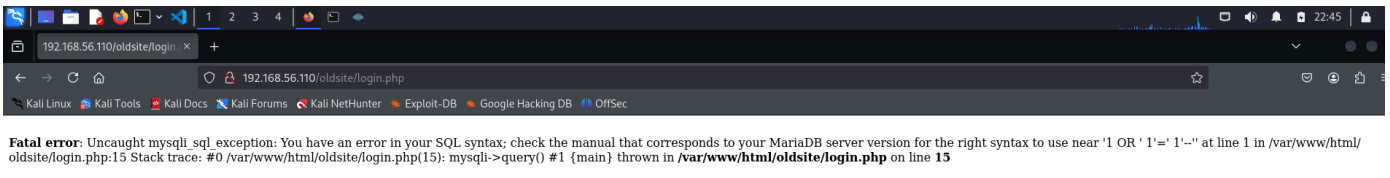
### Descrizione

Il payload SQL è una tecnica di bypass che forza una condizione sempre vera nella query sottostante. La sintassi ' OR '1' = '1' -- sfrutta l'operatore logico OR per trasformare la condizione della query in un'espressione sempre vera (1 = 1). Il doppio trattino (--) indica l'inizio di un commento in SQL, ignorando tutto ciò che segue.




### Risultato

In questo caso, il server ha restituito un Fatal Error, indicando che il database non è stato in grado di elaborare la query manipolata. Questo comportamento suggerisce che il sistema è effettivamente vulnerabile a SQL Injection, poiché la query viene direttamente elaborata dal backend senza alcuna sanificazione adeguata.



### Implicazioni

Il risultato conferma la presenza di una vulnerabilità sfruttabile, aprendo la strada a ulteriori test per estrarre informazioni dal database o ottenere accesso non autorizzato.

Dopo aver identificato che la pagina /oldsite/login.php potrebbe essere vulnerabile a SQL Injection, abbiamo effettuato ulteriori test manuali per comprendere meglio la struttura della query SQL sottostante.

### Primo Tentativo: ORDER BY 3 --

Abbiamo inizialmente utilizzato il payload:

```
' ORDER BY 3 --
```

### Descrizione

Questo payload verifica se esistono almeno tre colonne nella query SQL del backend. L'istruzione ORDER BY 3 tenta di ordinare i risultati della query in base alla terza colonna.

- ': Inizia la manipolazione della query.
- ORDER BY 3: Tenta di ordinare i risultati in base alla terza colonna.
- -- Commenta il resto della query originale per evitare conflitti.

### Risultato

Il server ha restituito un errore fatale.

Questo indica che la query SQL non è riuscita a trovare una terza colonna, suggerendo che la tabella interrogata ha meno di tre colonne.

### Secondo Tentativo: ORDER BY 2 --

Per confermare il numero effettivo di colonne, abbiamo successivamente utilizzato il payload:

```
' ORDER BY 2 --
```

### Descrizione

Questa volta, abbiamo verificato se la query SQL del backend contiene almeno due colonne. L'istruzione funziona nello stesso modo, ma con il numero di colonne ridotto.

### Risultato

Il server ha restituito il seguente messaggio di errore:

```
'ORDER BY 2- is unknown or password
```

Questo errore implica che il backend accetta ORDER BY 2, ma il resto della query non è stato interpretato correttamente a causa del nostro payload.

### Conclusione

Dai risultati, possiamo dedurre che:

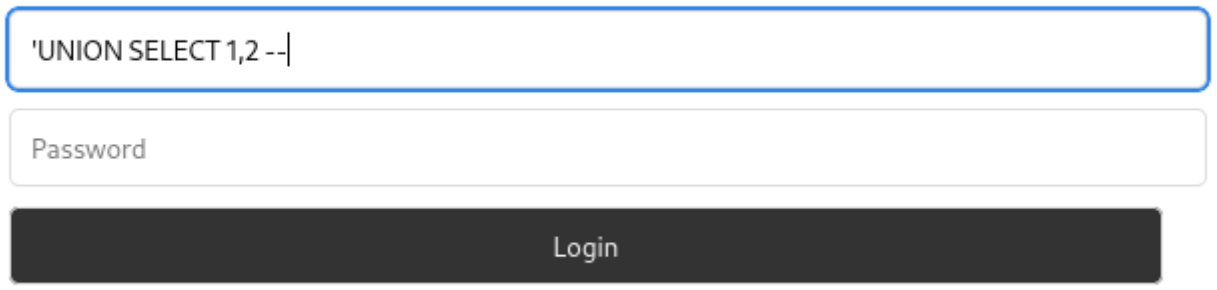
La query SQL è vulnerabile a manipolazioni tramite SQL Injection.
La tabella interrogata dal backend ha esattamente due colonne, poiché ORDER BY 3 restituisce un errore fatale, mentre ORDER BY 2 è accettato. Questo sarà utile per costruire ulteriori payload, come quelli per l'enumerazione dei dati.

Abbiamo proseguito i tentativi di SQL Injection sul sito alla pagina /oldsite/login.php utilizzando il payload:

```
' UNION SELECT 1,2 --
```

### Descrizione del Comando

Questo payload sfrutta l'operatore UNION per combinare il risultato della query originale con quello di una query appositamente costruita. In questo caso, stiamo tentando di selezionare due valori arbitrari (1 e 2) per determinare la vulnerabilità della query e verificare se il numero di colonne della tabella corrisponde a due.




### Risultato

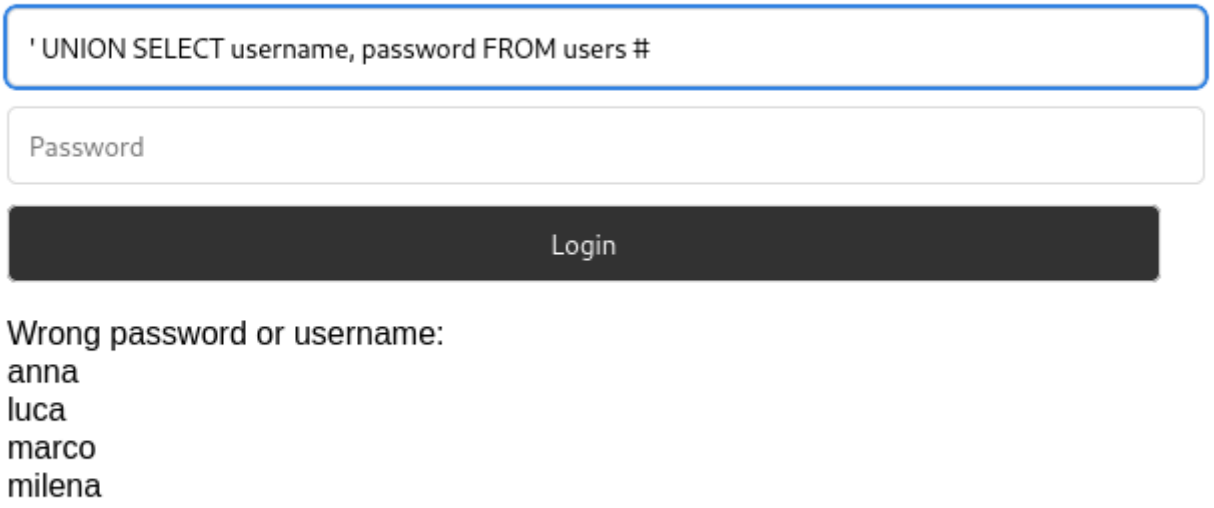
L'applicazione ha restituito il messaggio: "wrong password or username: 1". Questo risultato indica che il payload è stato accettato dalla query SQL e che il numero di colonne nella tabella corrisponde al numero di colonne specificate nel nostro UNION SELECT.


**Fatal error**: Uncaught mysqli_sql_exception: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '' at line 1 in /var/www/html/oldsite/login.php:15
Stack trace: #0 /var/www/html/oldsite/login.php(15): mysqli->query() #1 {main} thrown in **/var/www/html/oldsite/login.php** on line **15**

Abbiamo effettutato un nuovo tentativo di SQLi con la query `' UNION SELECT username, password FROM users #`

## Descrizione del Comando

Questo payload sfrutta l'operatore UNION per combinare il risultato della query originale con quello di una query costruita ad hoc. In questo caso, si tenta di estrarre i valori delle colonne username e password dalla tabella users, presupponendo che i nomi delle colonne siano corretti e corrispondano alla struttura del database. L'uso di # come commento elimina il resto della query originale, evitando errori di sintassi.



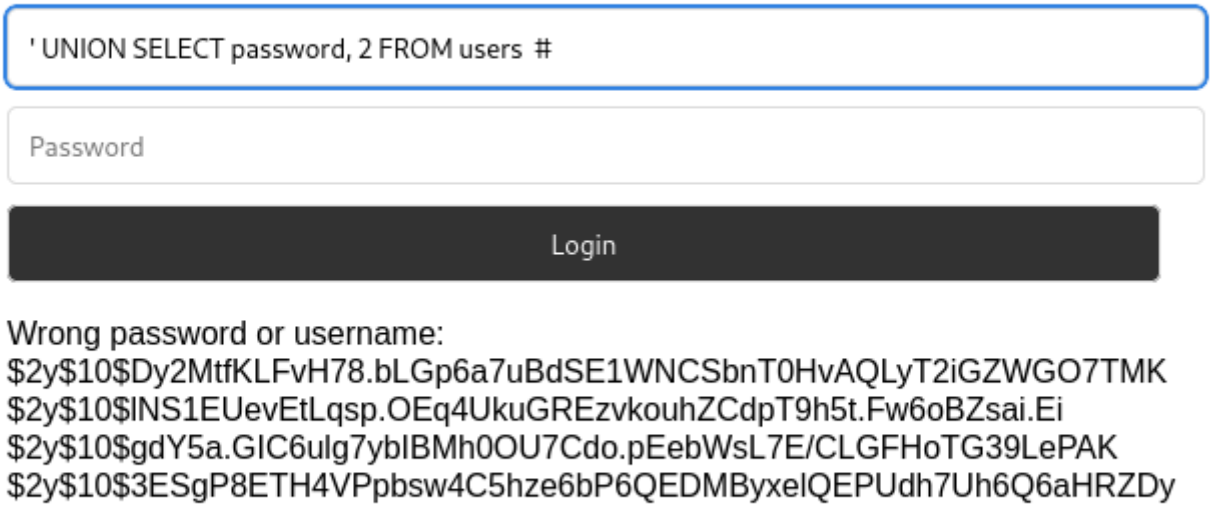

## Risultati

La query che abbiamo digitato ci ha restituito i valori della colonna username della tabella users.
I dati ottenuti sono:

- anna
- luca
- marco
- milena

Abbiamo effettuato un altro tentativo per le password con la query:

```
' UNION SELECT password, 2 FROM users #
```

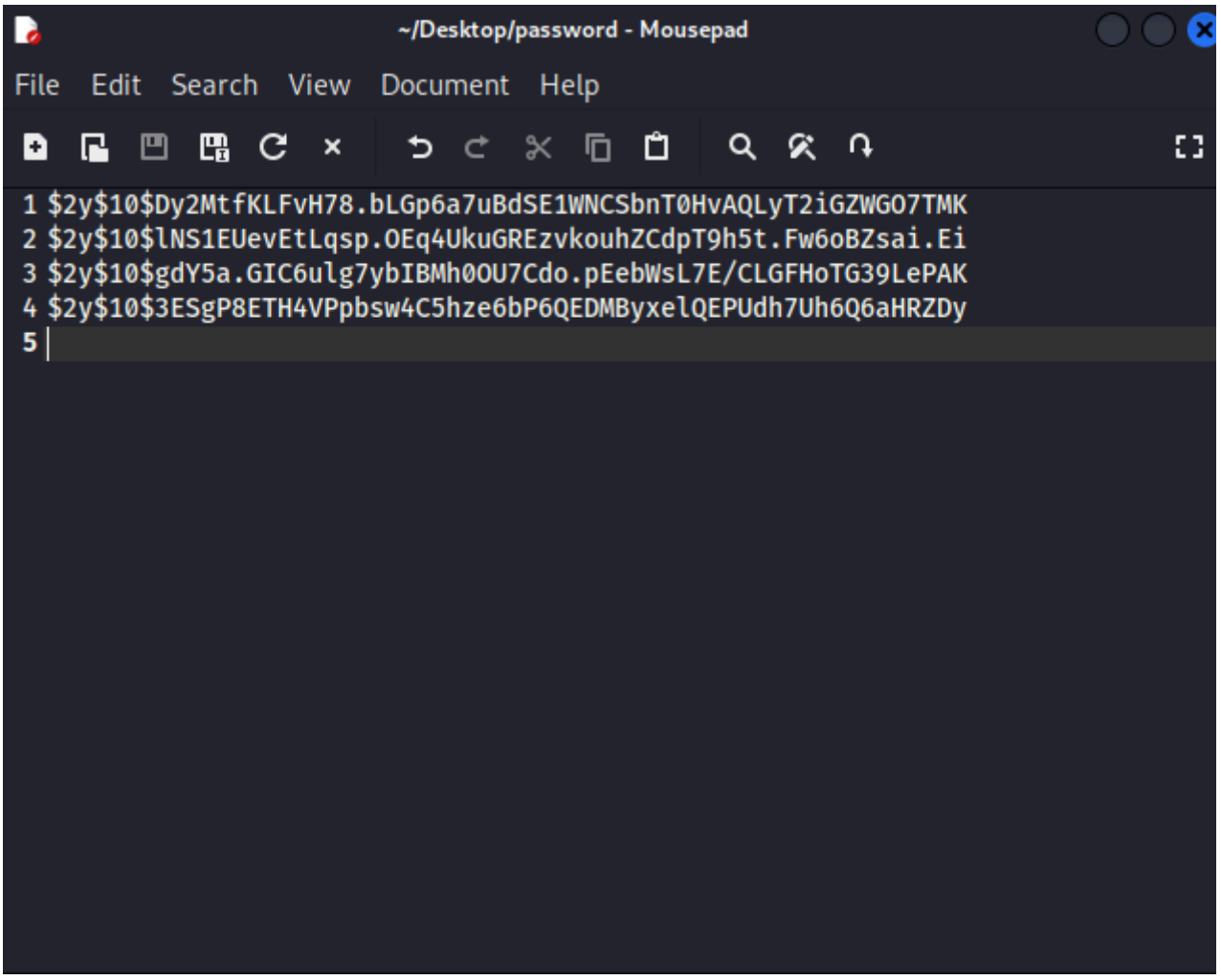



### Descrizione del Comando:

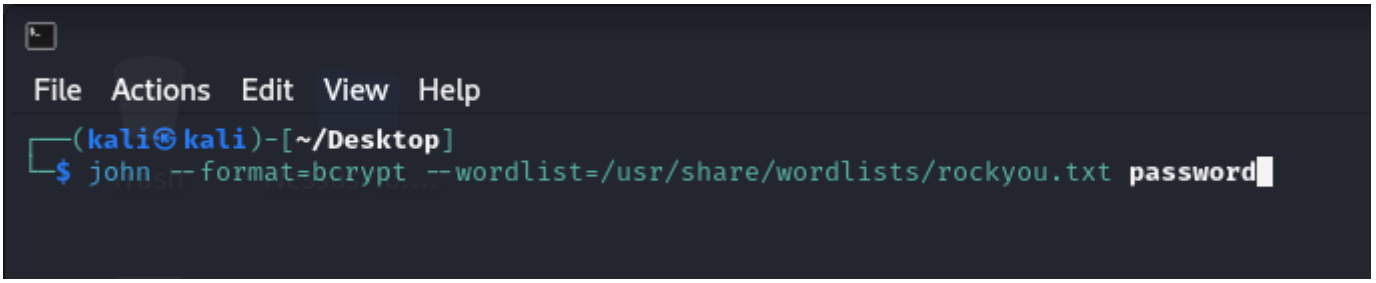
Anche qui il payload sfrutta l'operatore UNION per combinare i risultati della query originale con una query che seleziona la colonna password dalla tabella users, affiancandola a un valore arbitrario (in questo caso il numero 2). Il # commenta il resto della query.

### Analisi del risultato:

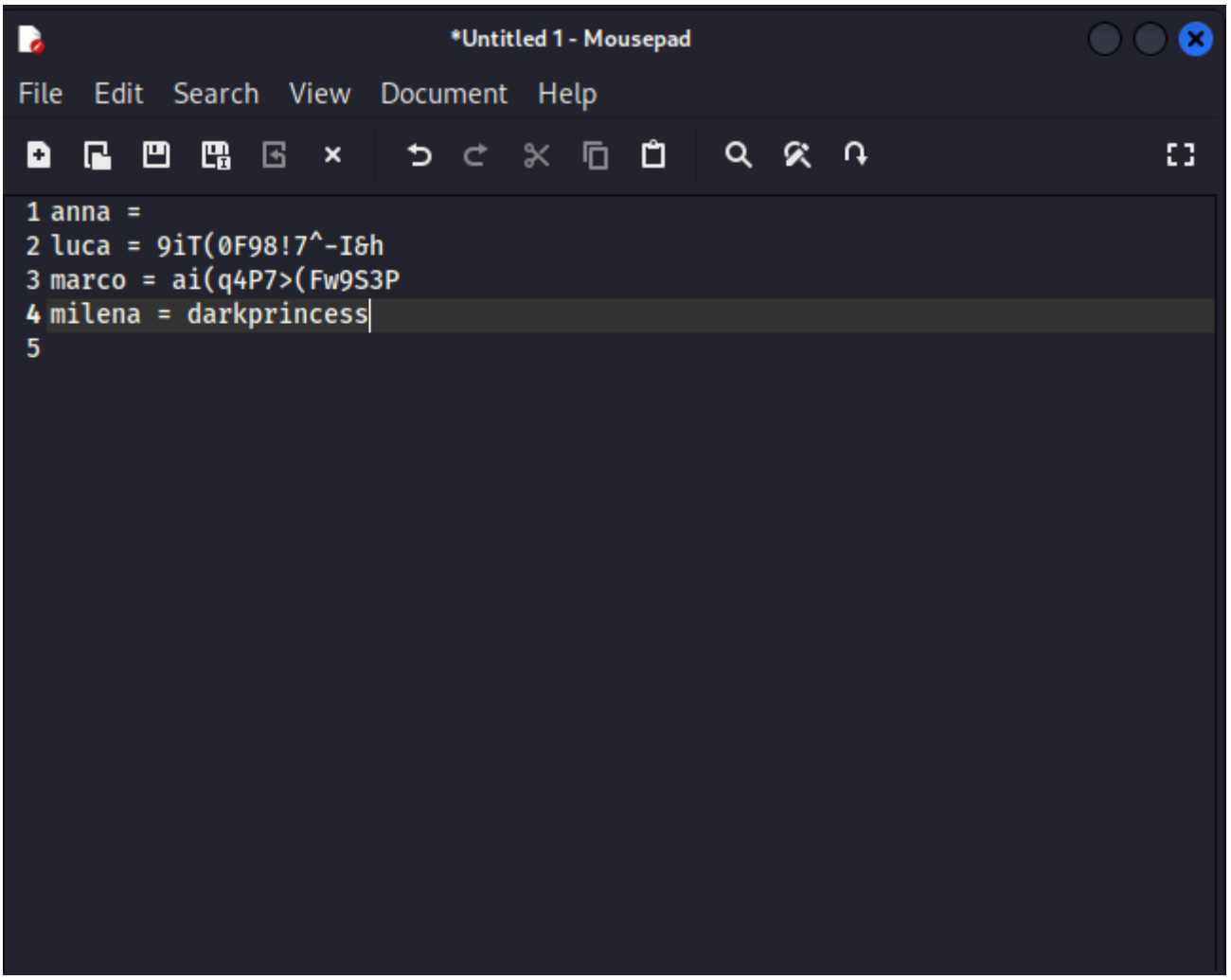
L'applicazione ha restituito i valori della colonna password, che sono verosimilmente hash delle password salvate nel database. I dati estratti includono hash in formato bcrypt



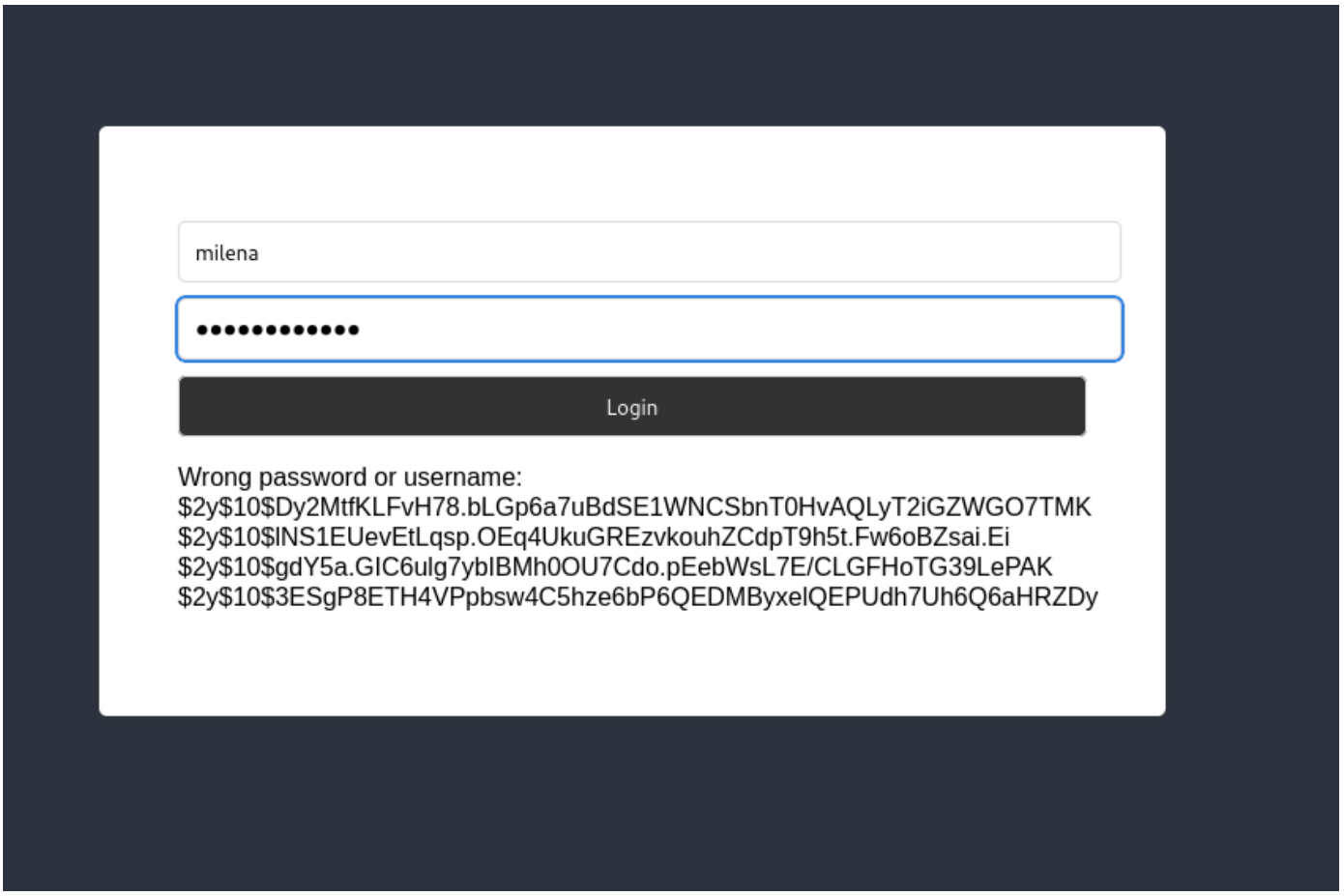

Dopodichè abbiamo provato ad encryptare gli hash trovati usando John The Ripper




Dopo aver atteso diverse ore abbiamo ottenuto gli hash encryptati!

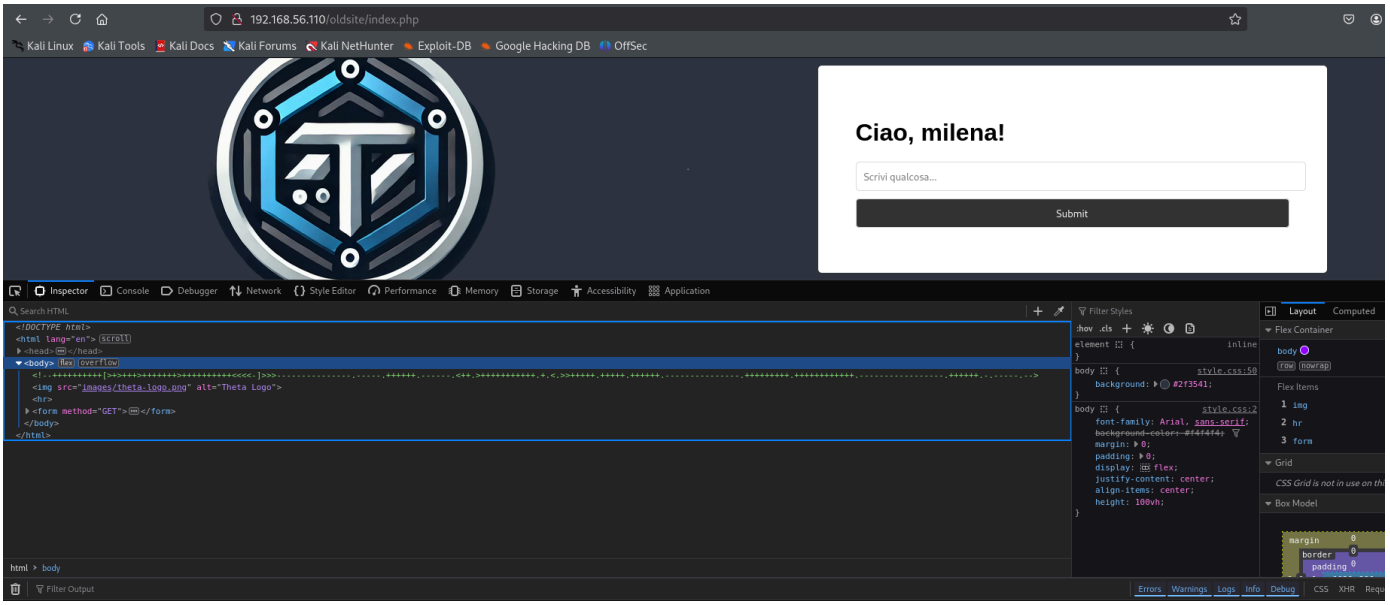



Dato che le password non erano molte le abbiamo testate tutte per ogni utente!
Abbiamo tentato il login sul sito con le credenziali di milena.



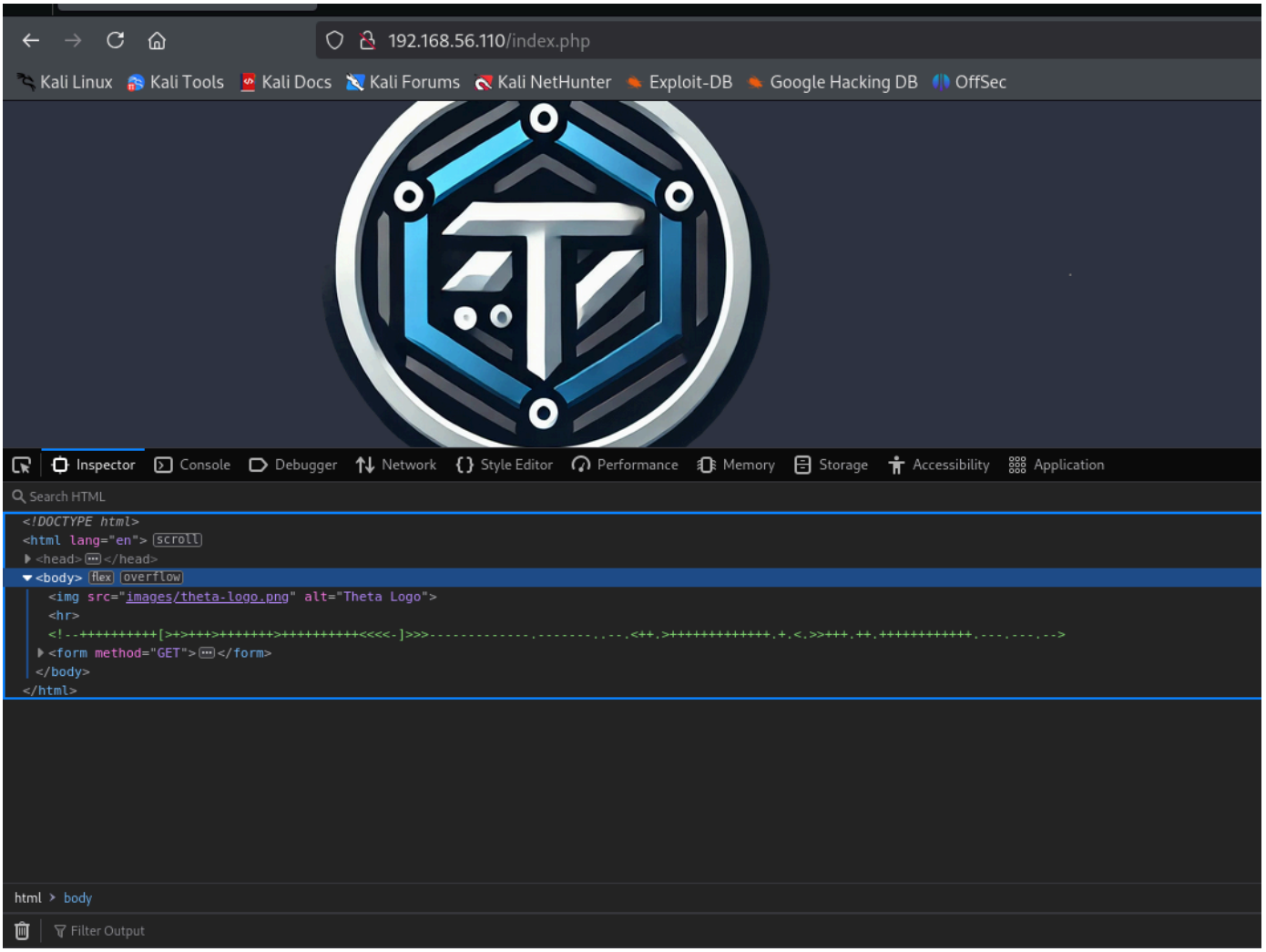

> Il form è rimasto sporco dalla query di prima.

Nel codice sorgente della pagina, abbiamo individuato un commento contenente un frammento di codice scritto nel linguaggio Brainfuck:

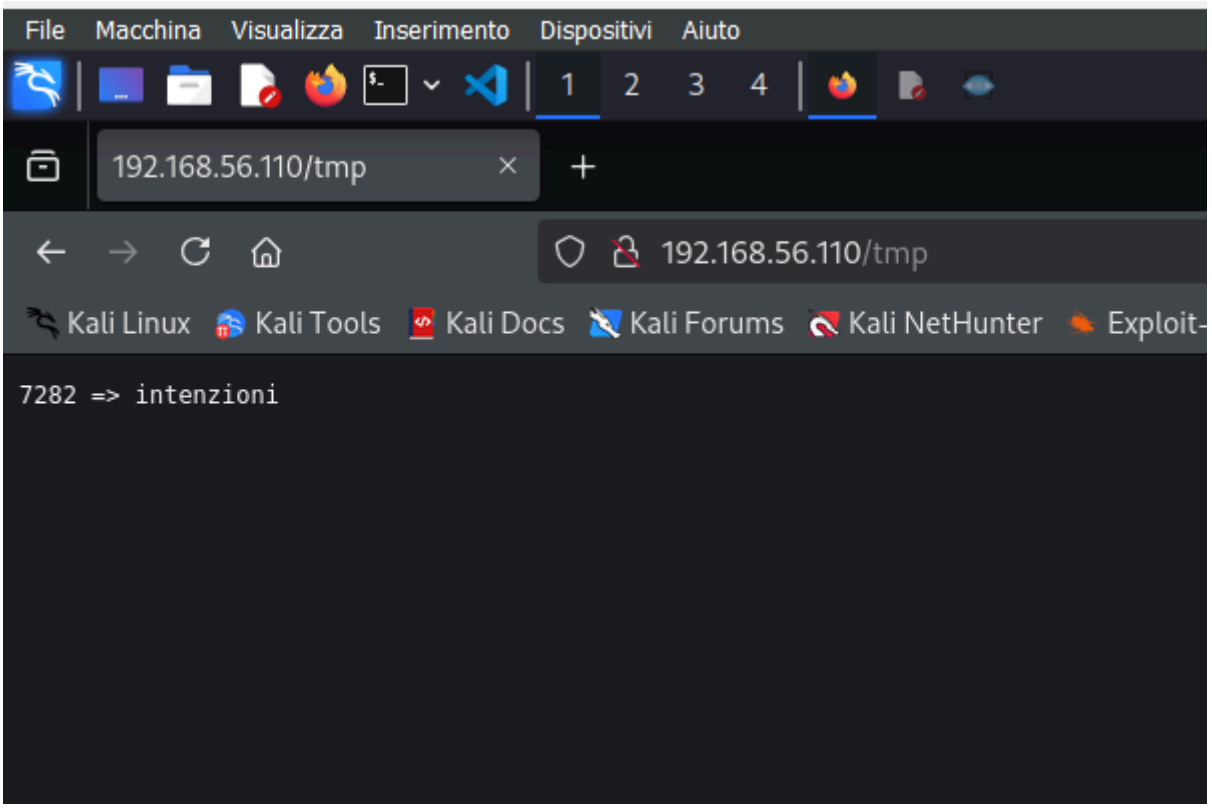



Per decodificare il messaggio, abbiamo utilizzato un decodificatore online, come decode.fr, che consente di interpretare e tradurre Brainfuck in testo leggibile.

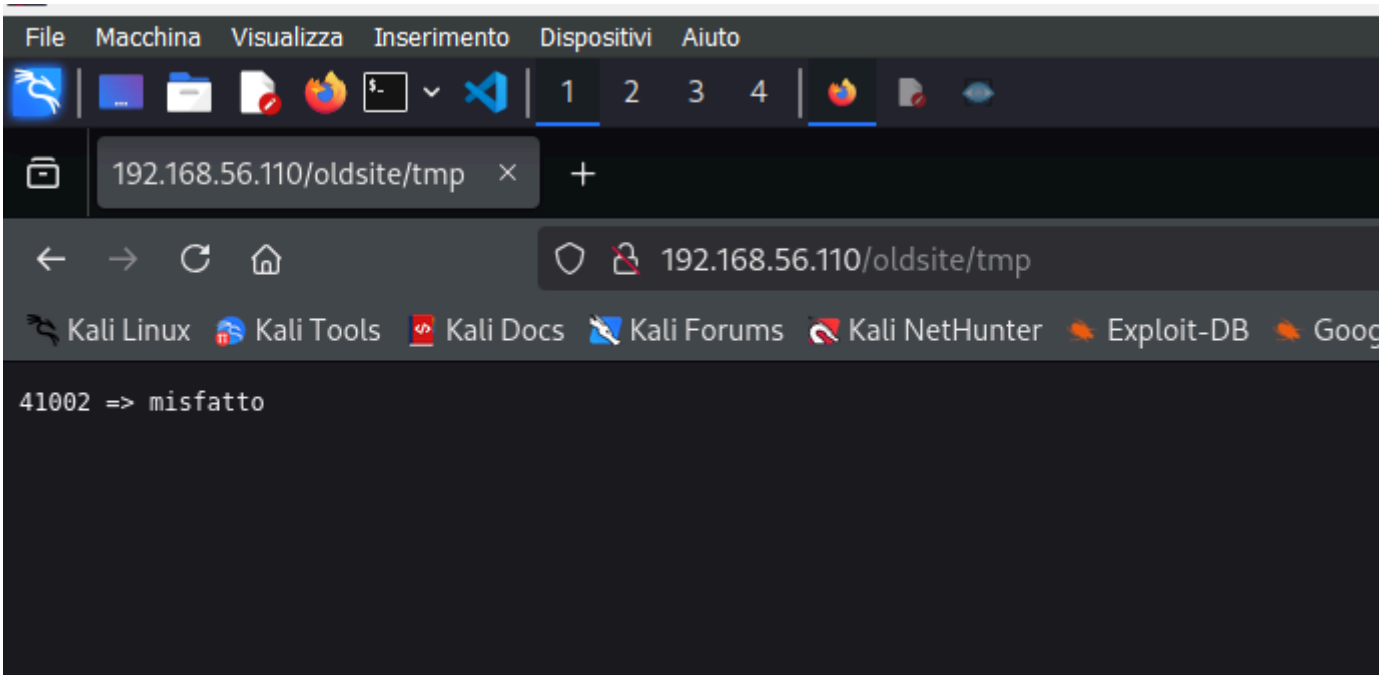
Abbiamo fatto gli stessi passaggi anche per la versione non /oldsite del sito.



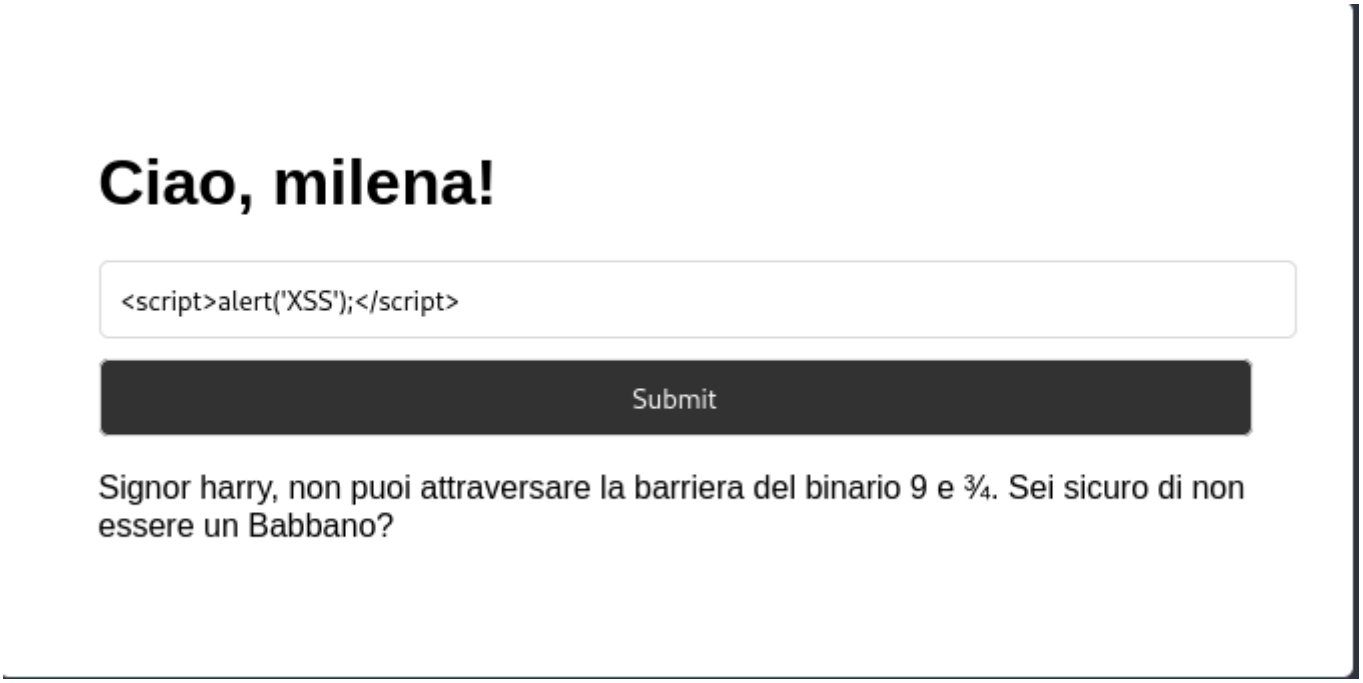

Dopo abbiamo controllato le altre pagine controllate con gobuster, la prima che abbiamo esplorato è /tmp



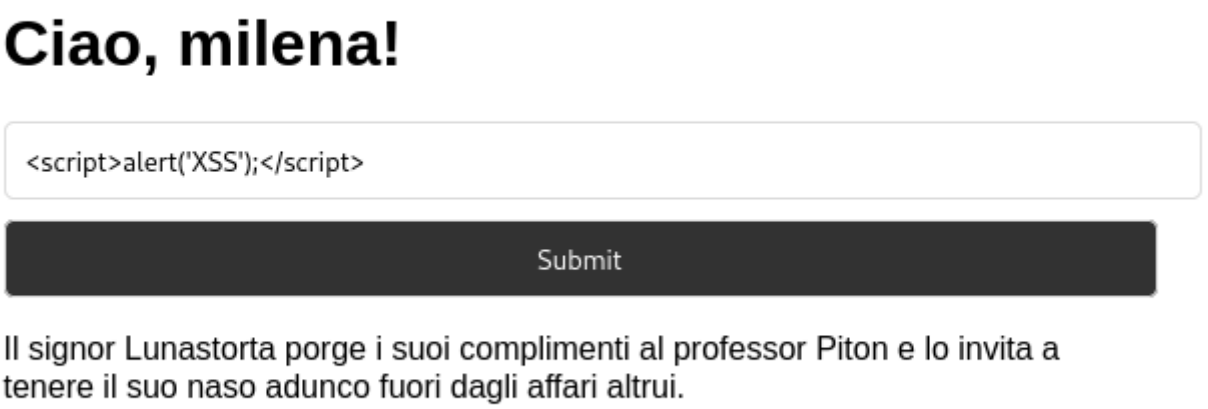
e a seguire /oldsite/tmp.



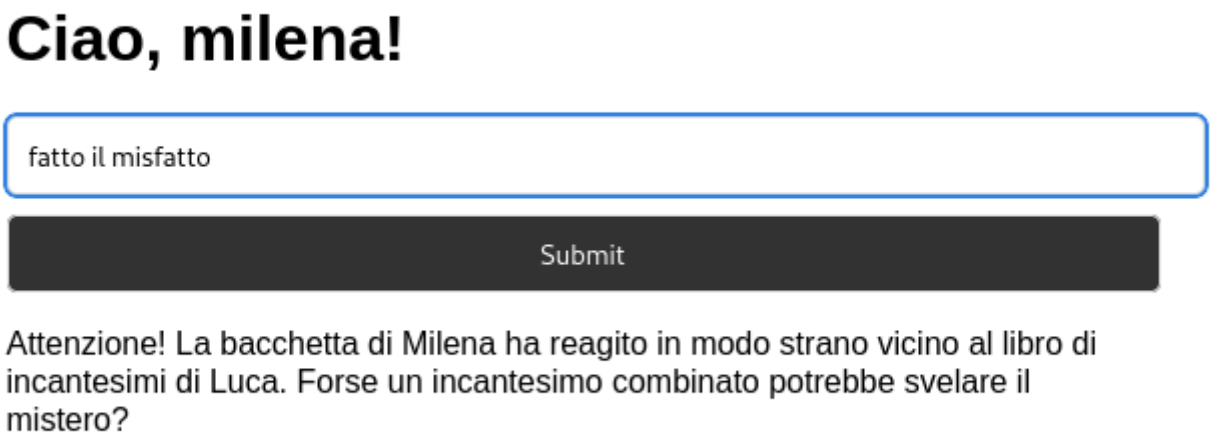

Ragionando sul fatto che la SQLi ha funzionato abbiamo anche tentato una XSS su entrambi i siti. questa query: `<script>alert('XSS');</script>`



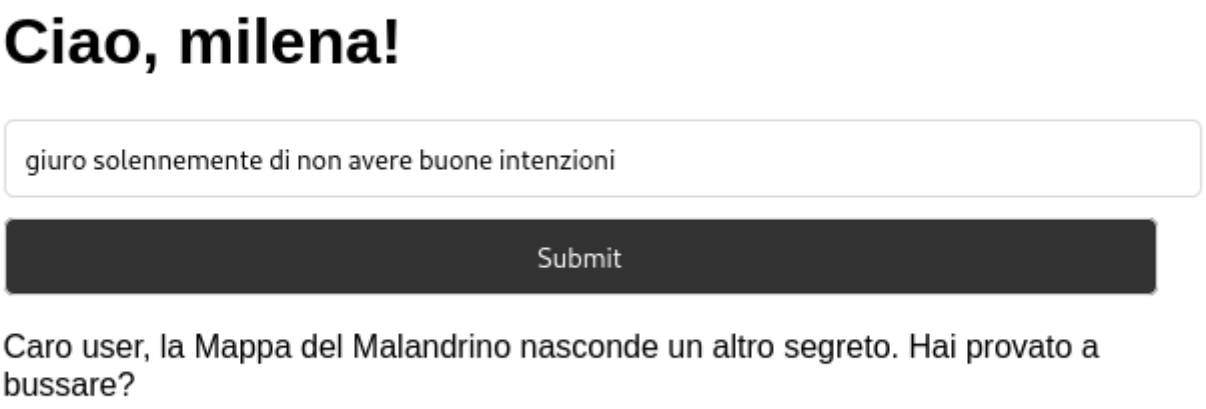

> Quello qui sotto è /oldsite:




dal risultato del /oldsite notiamo 2 personaggi della saga di Harry Potter, uno dei quali appartenente al gruppo chiamato "Malandrini" , abbiamo infatti tentato le formule usate nel film all'interno del FORM le uniche 2 combinazioni tra formula e sito che ci hanno dato un riscontro sono:



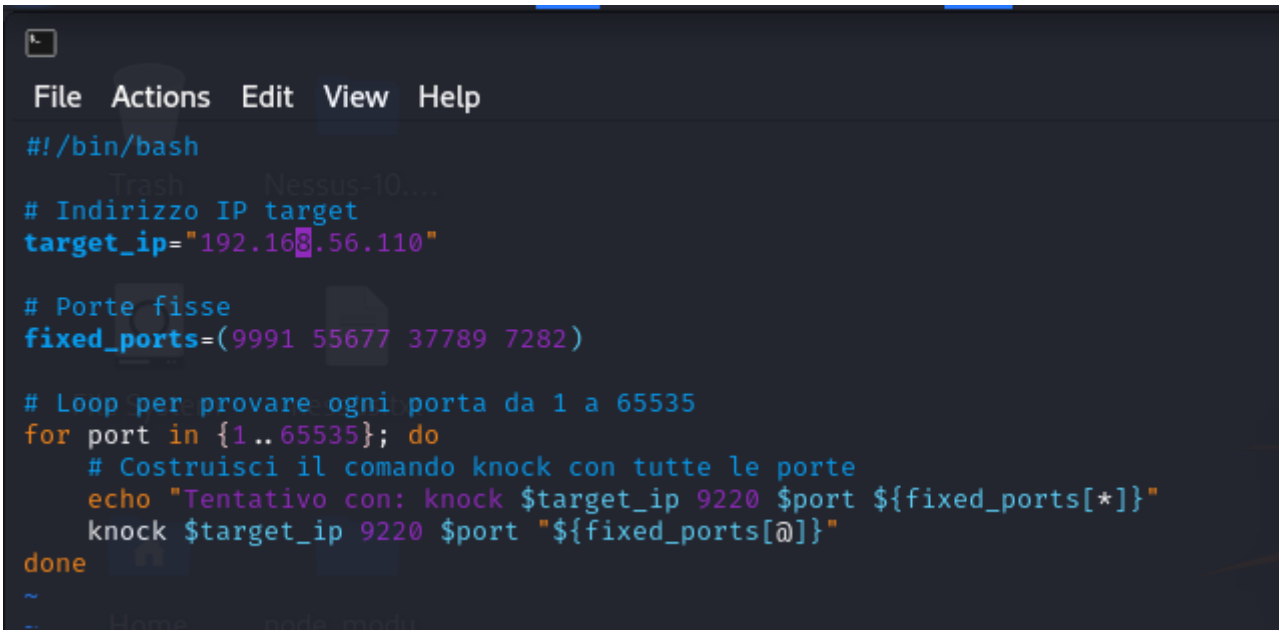

> Lo screen sotto è del sito nuovo!!



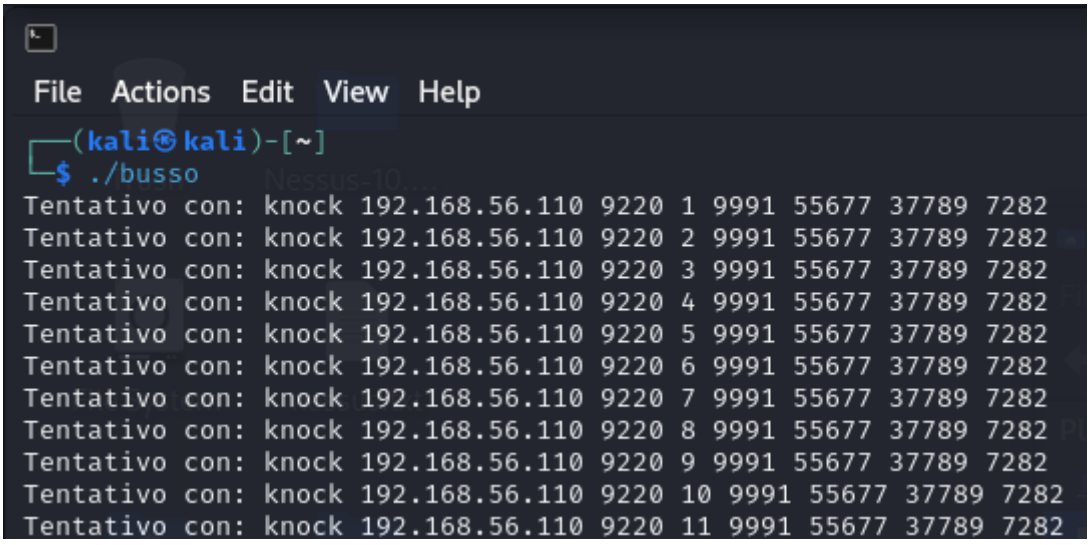

Alla domanda del sito nuovo immaginiamo di dover usare il port knocking, le porte che abbiamo pensato fossero giuste da utilizzare per il port knocking sono i derivati delle codifiche dei codici brainfuck.

- 9220 => giuro
- 9991=> di
- 55677 => non avere
- 37789 => buone
- 7282 => intenzioni
- 65511 => fatto
- 12000 => il
- 41002 => misfatto

Le ultime 3 porte fanno parte della pagina /oldsite e quindi visto che l'idea del port knocking è arrivata dalla nuova versione del sito decidiamo di bussare con quelle porte, tuttavia ci mancava una porta per formare la frase completa, per ovviare al problema del tempo in quanto non riuscivamo a trovarla abbiamo utilizzato uno script che escludeva le ultime 3 porte e che tentava il knocking sulle porte della prima frase che già avevamo, provando tutte le porte come porta incognita.

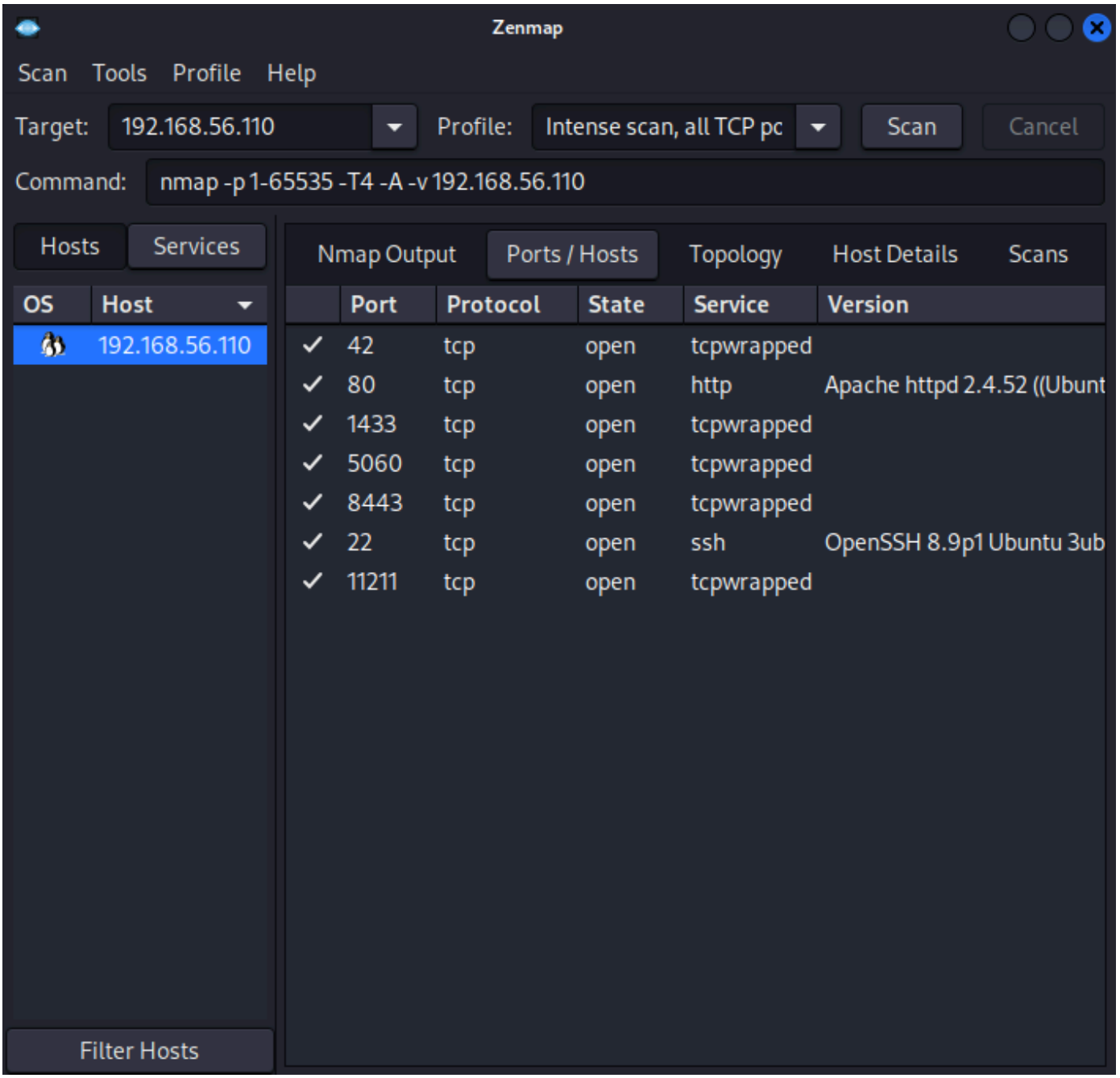



Ed ecco i tentativi di knocking.

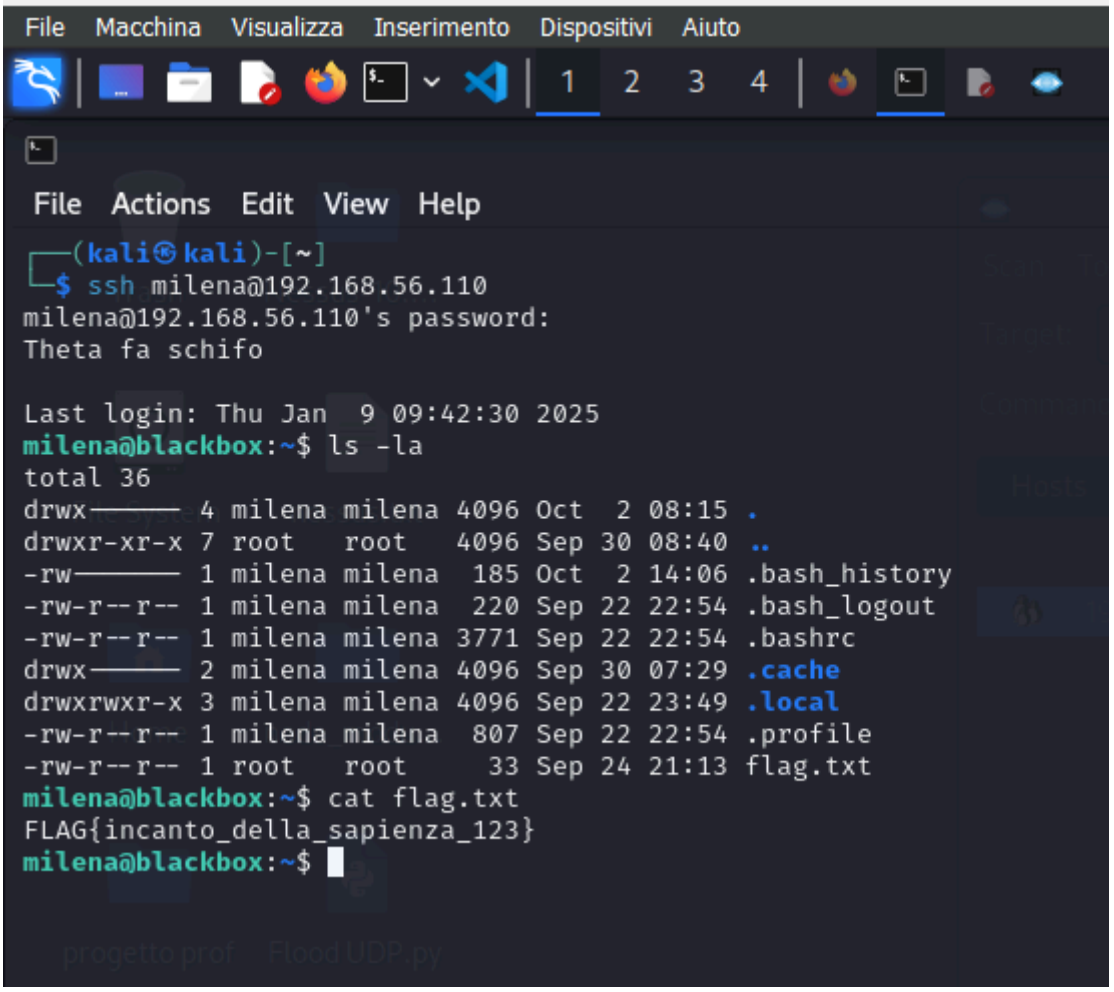



### Risultato

Dal knocking effetuando una nuova scansione abbiamo visto che la porta 22 si è improvvisamente aperta.




A questo punto avendo l'ssh aperta abbiamo tentato un login con tutti gli utenti e le rispettive password ottenute con l'SQLi e encryptate con John.




Dopo esserci loggati con milena navigando nelle sue directory abbiamo trovato la prima FLAG.

Dopodichè abbiamo navigato un po dentro la macchina e ci siamo imbattuti nella cartella shared che conteneva un'altro file interessante con altre 2 password che avevamo encryptato già con John.

```
milena@blackbox:~$ ls -la
total 36
drwx------ 4 milena milena 4096 Oct  2 08:15 .
drwxr-xr-x 7 root   root   4096 Sep 30 08:40 ..
-rw------- 1 milena milena  185 Oct  2 14:06 .bash_history
-rw-r--r-- 1 milena milena  220 Sep 22 22:54 .bash_logout
-rw-r--r-- 1 milena milena 3771 Sep 22 22:54 .bashrc
drwx------ 2 milena milena 4096 Sep 30 07:29 .cache
drwxrwxr-x 3 milena milena 4096 Sep 22 23:49 .local
-rw-r--r-- 1 milena milena  807 Sep 22 22:54 .profile
-rw-r--r-- 1 root   root     33 Sep 24 21:13 flag.txt
milena@blackbox:~$ cat flag.txt
FLAG{incanto_della_sapienza_123}
milena@blackbox:~$ cd ..
milena@blackbox:/home$ ls
anna  luca  marco  milena  shared
milena@blackbox:/home$ cd shared
milena@blackbox:/home/shared$ ls
milena@blackbox:/home/shared$ ls -la
total 12
drwxrwx--- 2 anna   shared 4096 Oct  2 15:21 .
drwxr-xr-x 7 root   root   4096 Sep 30 08:40 ..
-rw-rw-r-- 1 milena shared   45 Oct  2 15:21 .myLovePotion.swp
milena@blackbox:/home/shared$ cat .myLovePotion.swp
ai(q4P7>(Fw9S3P
9iT(0F98!7^-I&h
darkprincess
milena@blackbox:/home/shared$
```

Subito dopo abbiamo effettuato il login con Luca sempre nel servizio SSH e abbiamo trovato un'altra FLAG.

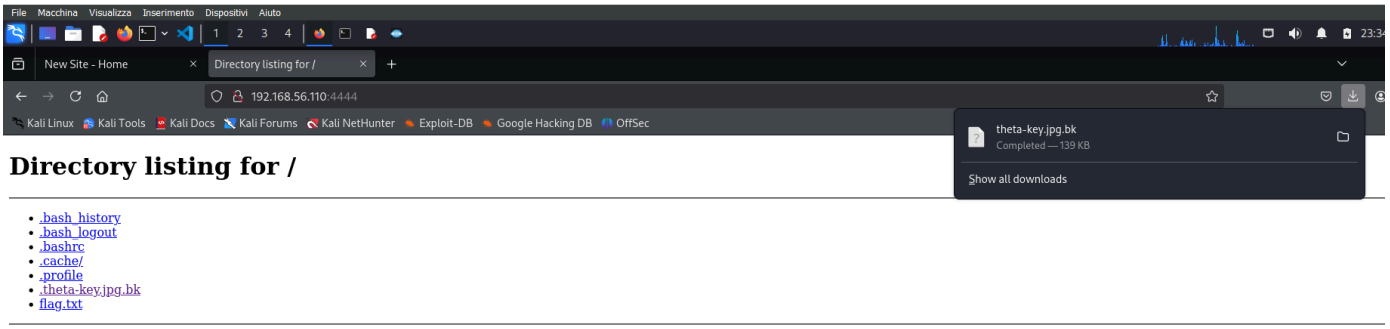
```
File  Macchina  Visualizza  Inserimento  Dispositivi  Aiuto
1  2  3  4
File  Actions  Edit  View  Help
┌──(kali㉿kali)-[~]
└─$ ssh luca@192.168.56.110
luca@192.168.56.110's password:
Theta fa schifo

Last login: Thu Jan  9 16:14:48 2025 from 192.168.56.107
luca@blackbox:~$ ls
flag.txt
luca@blackbox:~$ cat flag.txt
FLAG{cuore_di_leone_456}
luca@blackbox:~$
```

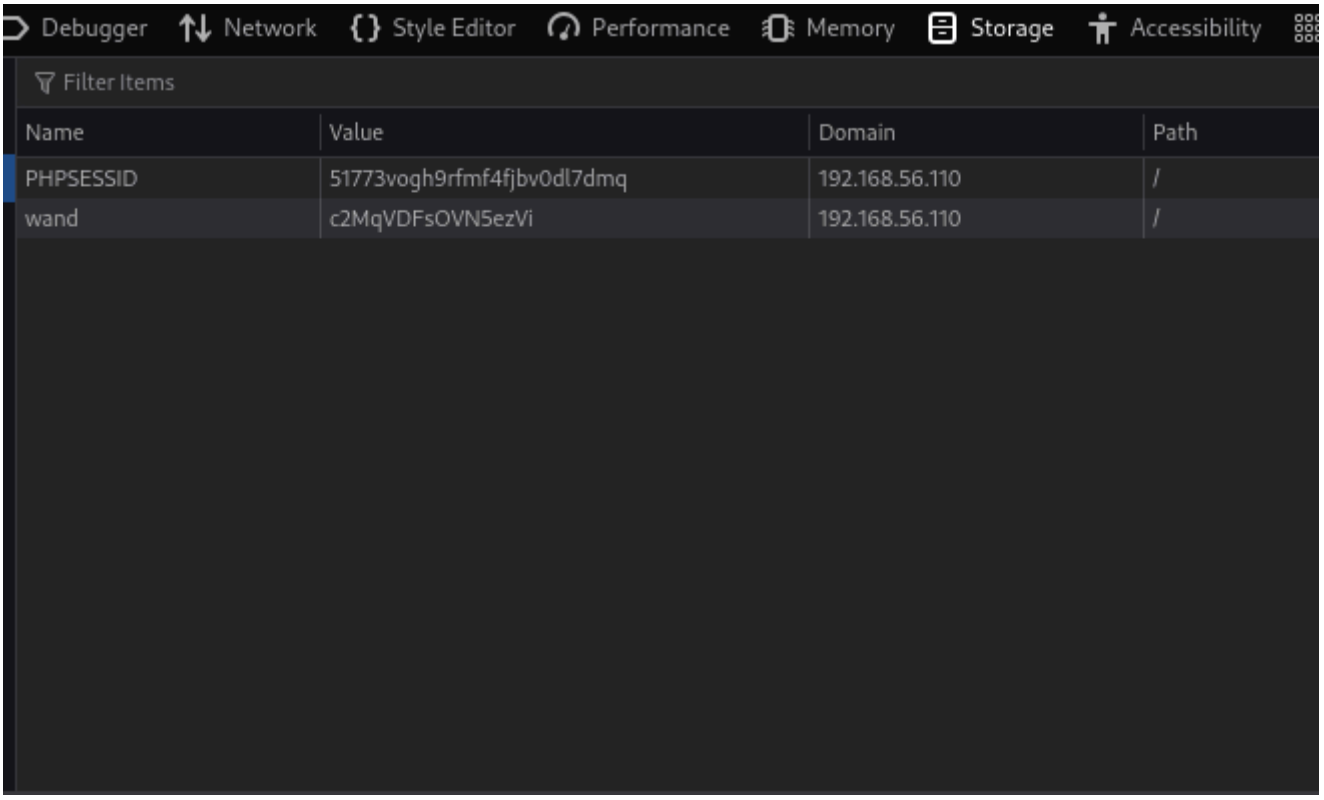
Continuando a navigare con l'account di luca abbiamo trovato un file nascosto che si è rivelato essere il backup di un'immagine.
A questo punto abbiamo avviato un server python per poter scaricare l'immagine in questione.

```
luca@blackbox:~$ ls -la
total 172
drwx------ 3 luca luca   4096 Jan  9 15:38 .
drwxr-xr-x 7 root root   4096 Sep 30 08:40 ..
-rw------- 1 luca luca     63 Jan  9 16:19 .bash_history
-rw-r--r-- 1 luca luca    220 Sep 22 22:56 .bash_logout
-rw-r--r-- 1 luca luca   3771 Sep 22 22:56 .bashrc
drwx------ 2 luca luca   4096 Jan  9 15:37 .cache
-rw-r--r-- 1 luca luca    807 Sep 22 22:56 .profile
-rw-r--r-- 1 luca luca 142396 Oct  2 15:16 .theta-key.jpg.bk
-rw-r--r-- 1 root root     25 Sep 24 21:14 flag.txt
luca@blackbox:~$ python3 -m http.server 4444
Serving HTTP on 0.0.0.0 port 4444 (http://0.0.0.0:4444/) ...
```

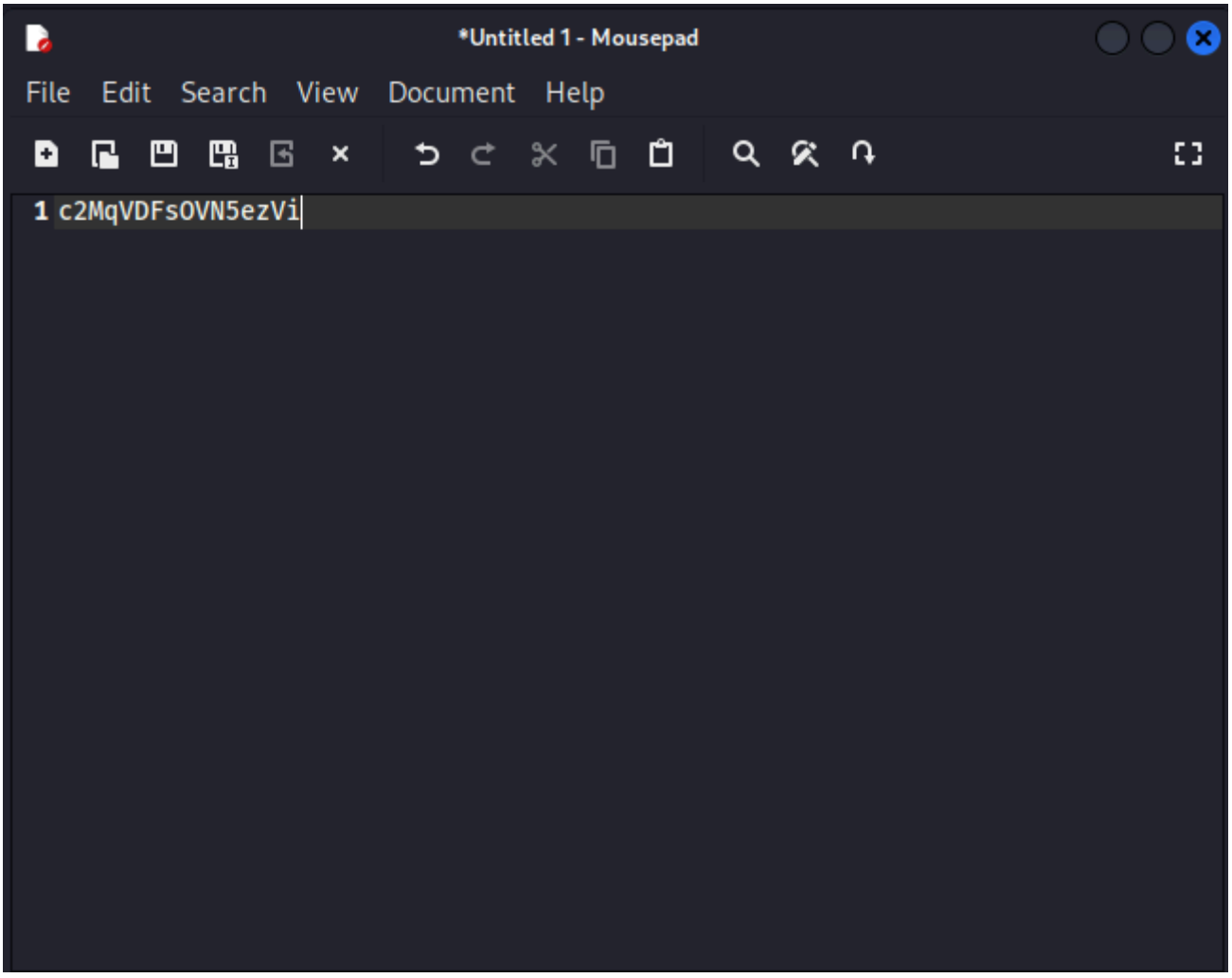
tramite il browser da kali abbiamo inserito il nostro ip e la porta in ascolto per scaricare il file.



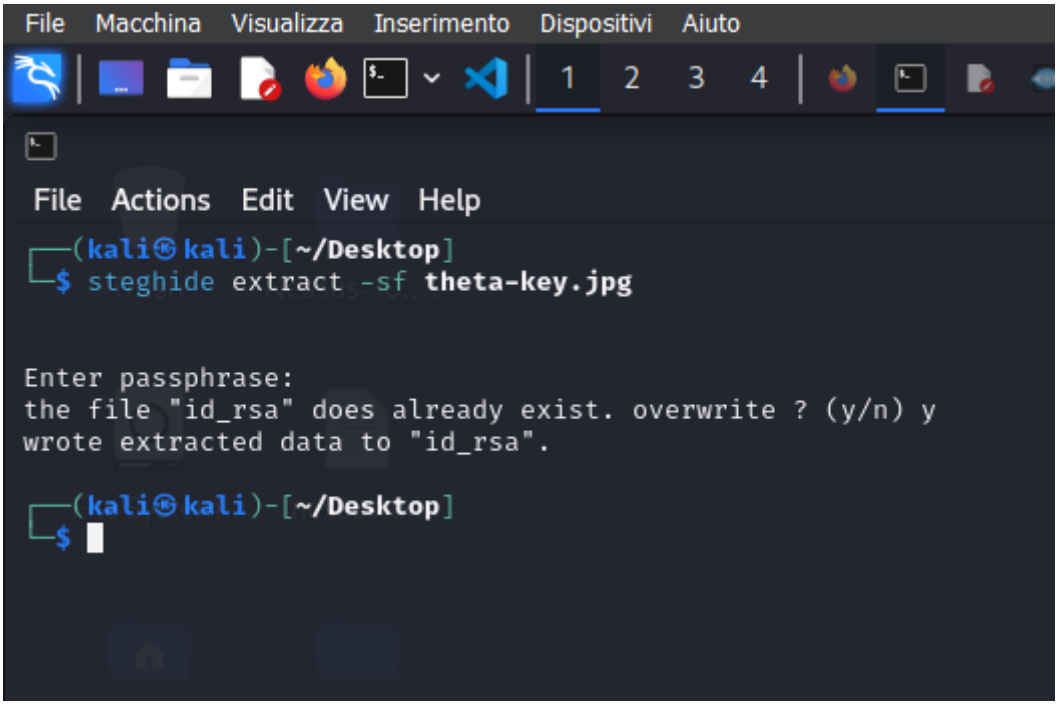

Avuta l'immagine abbiamo tentato l'estrazione con steghide.
Al primo tentativo ci è stata richiesta una passfrase.
Controllando il sito alla ricerca della passfrase ci siamo imbattuti nella sezione storage del dev tool del browser.



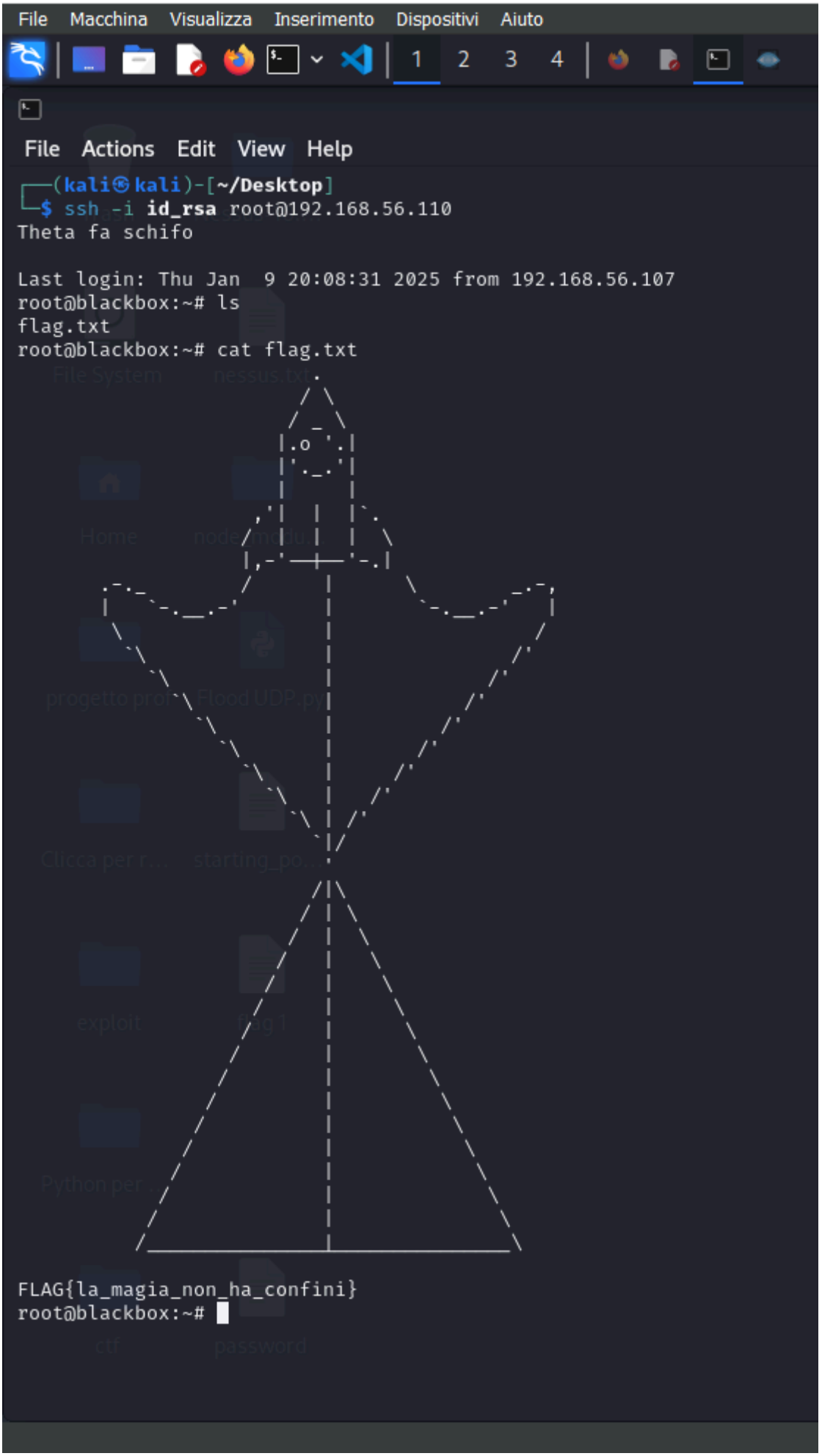
Tra i cookie quello chiamato "wand" ha catturato la nostra attenzione in quanto stranamente a tema con altre informazioni ricavate in precedenza abbiamo deciso quindi di salvarlo in un file di testo.



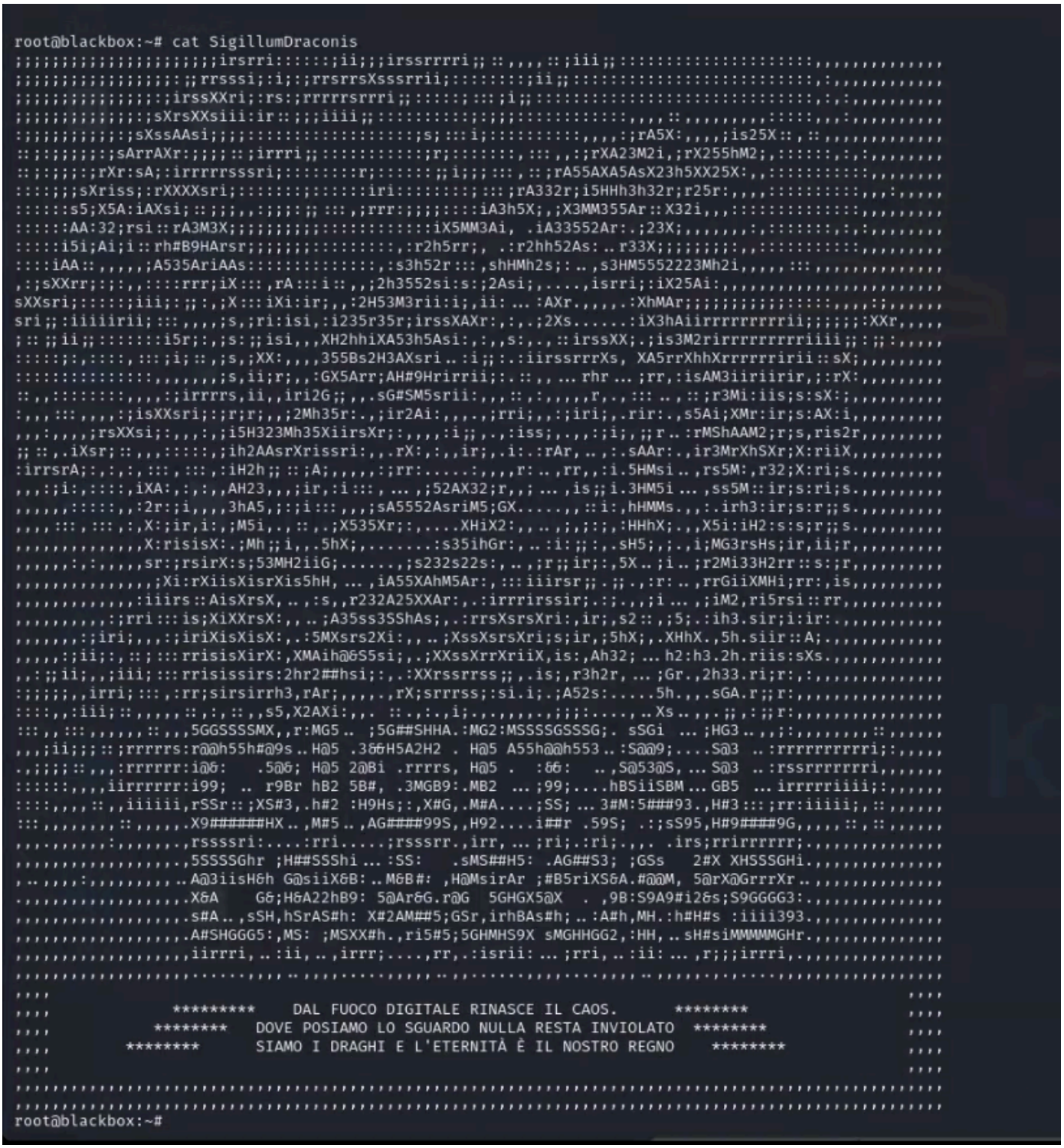

A questo punto abbiamo di nuovo tentato l'estrazione con steghide utilizzando questo come passfrase.



Dal risultato dell'estrazione abbiamo ottenuto una chiave id_rsa.
Visto che dalle investigazioni OSINT fatte prima dell'operazione sulla macchina gli unici 2 utenti ad essere collegati al misfatto erano luca e milena abbiamo deciso di tentare il login SSH con questa chiave con l'utente root.
Il login ha avuto successo controllando la macchina abbiamo trovato anche l'ultima FLAG.




Una volta raccolta l'ultima FLAG abbiamo deciso di lasciare la nostra firma!


root@blackbox:~# cat SigillumDraconis
DAL FUOCO DIGITALE RINASCE IL CAOS.
DOVE POSIAMO LO SGUARDO NULLA RESTA INVIOLATO
SIAMO I DRAGHI E L'ETERNITÀ È IL NOSTRO REGNO
root@blackbox:~#