

## CMPE 279 – Assignment 3

### DVWA – Damn Vulnerable Web App

#### Team Members:

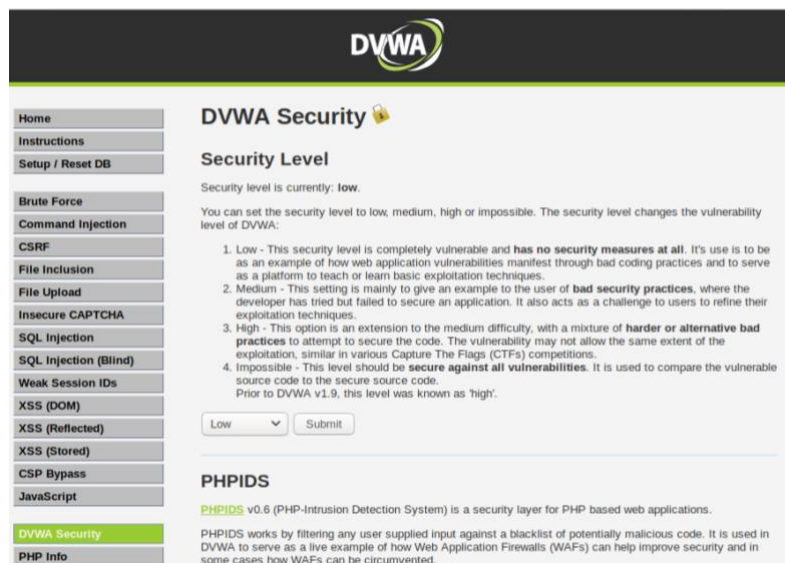
Sindhana Krishnamurthy

Nikitha Kallepalli

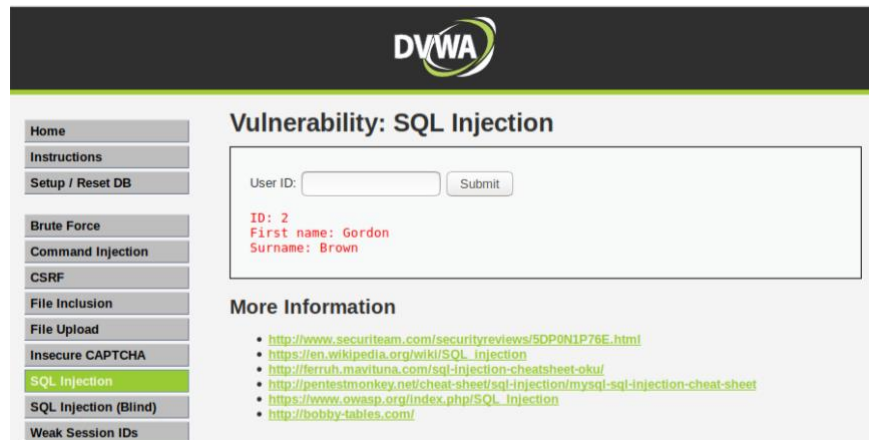
- Describe the SQLi attack you used, how did you cause the user table to be dumped?

**What was the input string you used?**

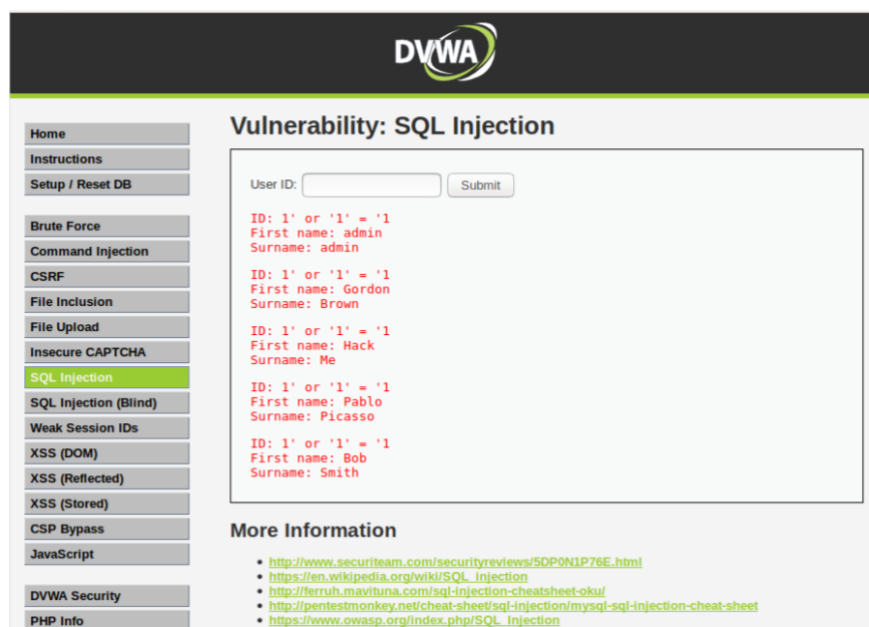
**Ans:** Once the DVWA is set up, choose DVWA security tab from the dashboard, set it “Low” and click “Submit” button next to it. Now the security level of the DVWA has been set to low.



Next, choose the SQL Injection tab to perform the SQL injection attack. We now notice that the application has a User ID field and when we input any id, for example, 2, it shows the details of the user such as the ID, First name and Surnames shown in the screenshot below.



SQL injection attacks work by modifying the SQL query and thereby attacks sensitive data. By looking at the source code using the “View Source” option we can understand that the SQL query is “SELECT first\_name, last\_name FROM users where user\_id = ‘\$id’;”. There are no security measures provided for the \$id input. So, when we execute a query like “SELECT first\_name, last\_name FROM users where user\_id = 1 or ‘1’ = ‘1 by giving an **input string** like **=1’ or ‘1’=’1** , the OR condition is always true and this will cause the user table to be dumped on the screen as shown below.

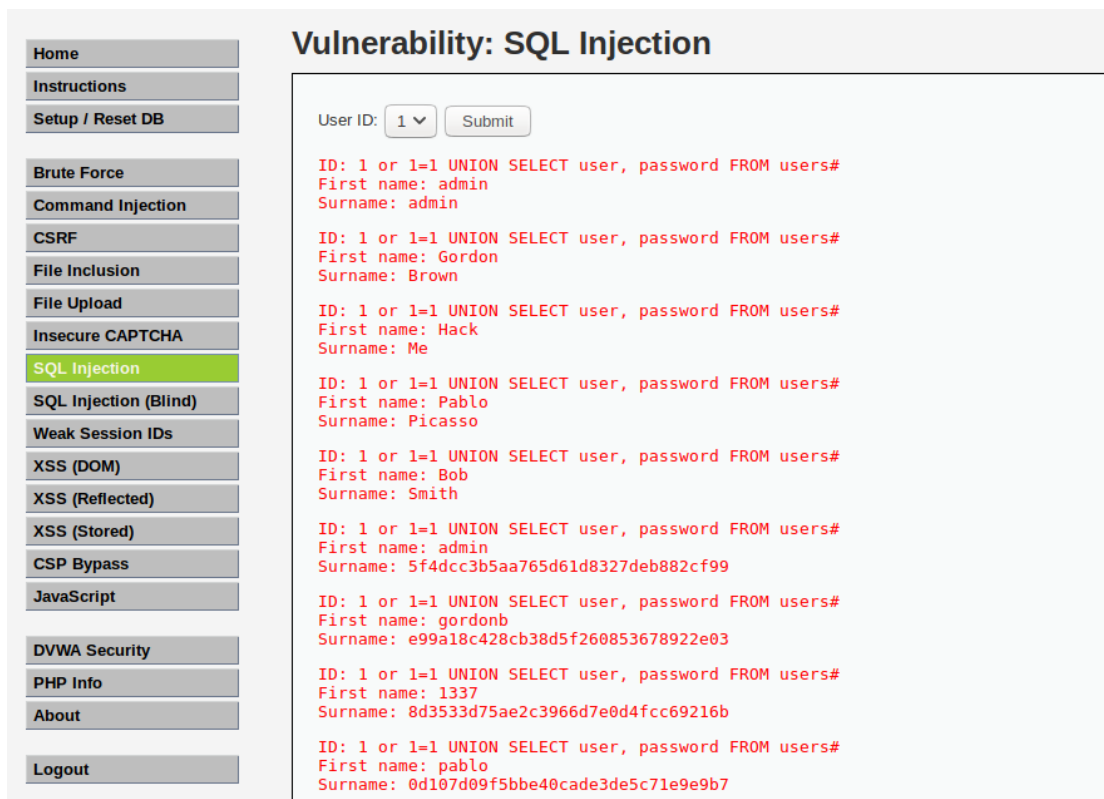


- If you switch the security level in DVWA to “Medium”, does the SQLi attack still work?

**Ans:** When we switch the security level in DVWA to “Medium” the User ID field changes to a drop-down menu. Hence, we cannot use the same SQLi attack that we used in the “Low” setting.

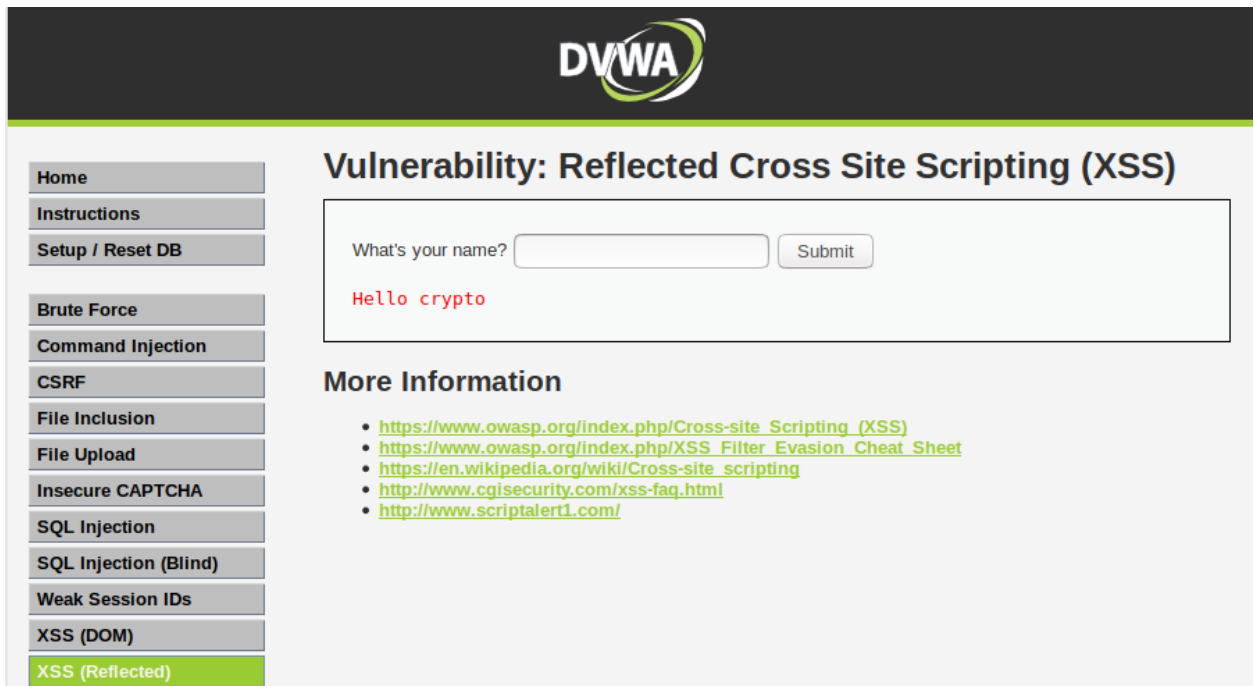


We can still perform an SQLi attack by modifying the value field of the drop-down menu by using the “Inspect Elements” option on the web page. We can directly add the query inside the html tag as `<option value = '1 or 1=1 UNION SELECT user, password FROM users#> 1` `</option>`. Now when we select the option “1” we get the details of the user table dumped as shown in the picture below.

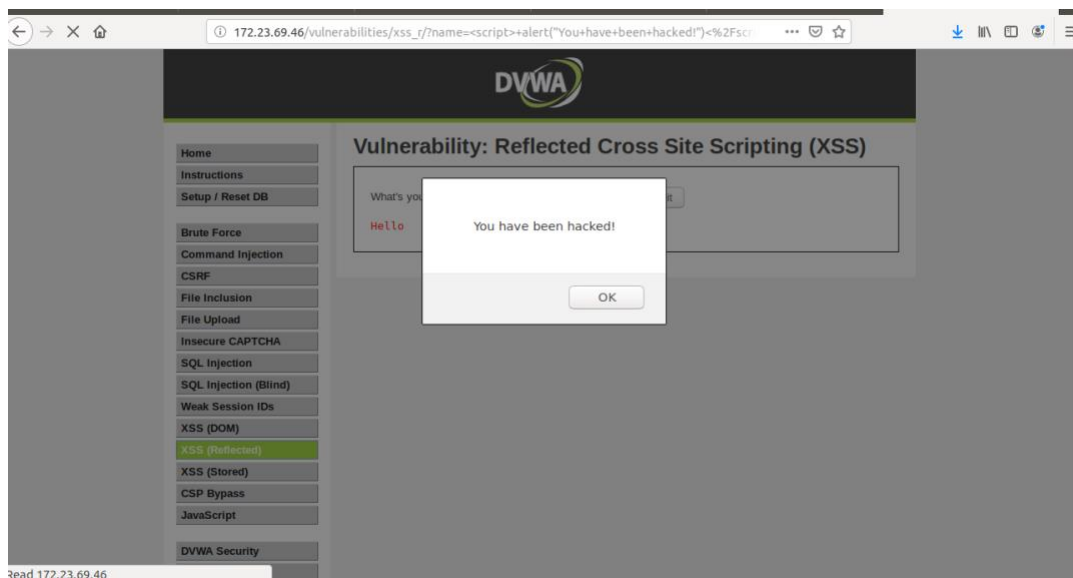


- Describe the reflected XSS attack you used, how did it work?

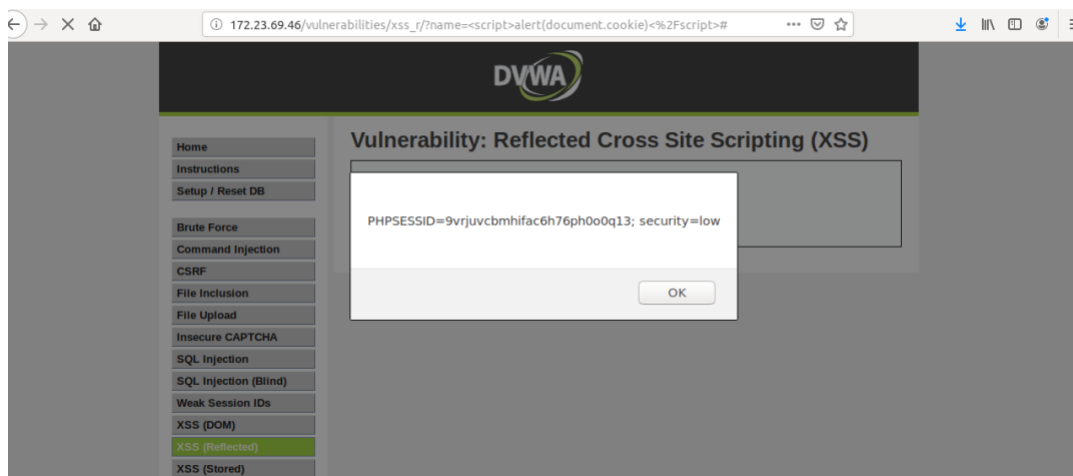
**Ans:** Reflected XSS attack occurs when malicious script is reflected off of a web application to the victim's browser. First, we keep the DVWA security as "Low" and then choose the "XSS(Reflected)" tab. Here, when we enter the name as "Crypto" and press "Submit" we get the following results.



Upon inspection, we notice that the name "crypto" gets reflected in the page source, thereby making it vulnerable to reflected XSS as there are no security measures for the HTML tags. We can now exploit this vulnerability by entering an **input string** like `<script>alert("You have been hacked!")</script>` in the text input area which is then reflected in the page as a pop up which is shown in the below picture. This proves that the code is vulnerable to reflected XSS.




We can also use an input string like `<script> alert(document.cookie)</script>` to obtain the session cookies as shown below.



- If you switch the security level in DVWA to “Medium”, does the XSS attack still work?

**Ans:** When the security level in DVWA was switched to “Medium” the exact input string did not work as the application source code checked for the occurrence of the “<script>” tag and the request was ignored given specific matches. Therefore, the <script> tag was removed, and the rest was taken as input as shown in the below figure.



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)

XSS (Stored)

CSP Bypass

JavaScript

## Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Hello alert("You have been hacked!")

### More Information

- [https://www.owasp.org/index.php/Cross-site\\_Scripting\\_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))
- [https://www.owasp.org/index.php/XSS\\_Filter\\_Evasion\\_Cheat\\_Sheet](https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet)
- [https://en.wikipedia.org/wiki/Cross-site\\_scripting](https://en.wikipedia.org/wiki/Cross-site_scripting)
- <http://www.cgisecurity.com/xss-faq.html>
- <http://www.scriptalert1.com/>

But, when we modify the `<script>` tag to be in uppercase or embed it with another `<script>` tag, the security fails as JavaScript is not case sensitive and there are no checks for the ending tag(`</script>`). This is shown in the below figures for the respective inputs.

**Input:** `<sc<script>ript> alert("You have been hacked!")</script>`

172.23.69.46/vulnerabilities/xss\_r?name=<sc<script>ript>alert("You+have+been+hacked!")</script>

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)

XSS (Stored)

CSP Bypass

JavaScript

## Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Hello

You have been hacked!

OK

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

