Redynox Internship

TASK 2: INTRODUCTION TO WEB APPLICATION SECURITY

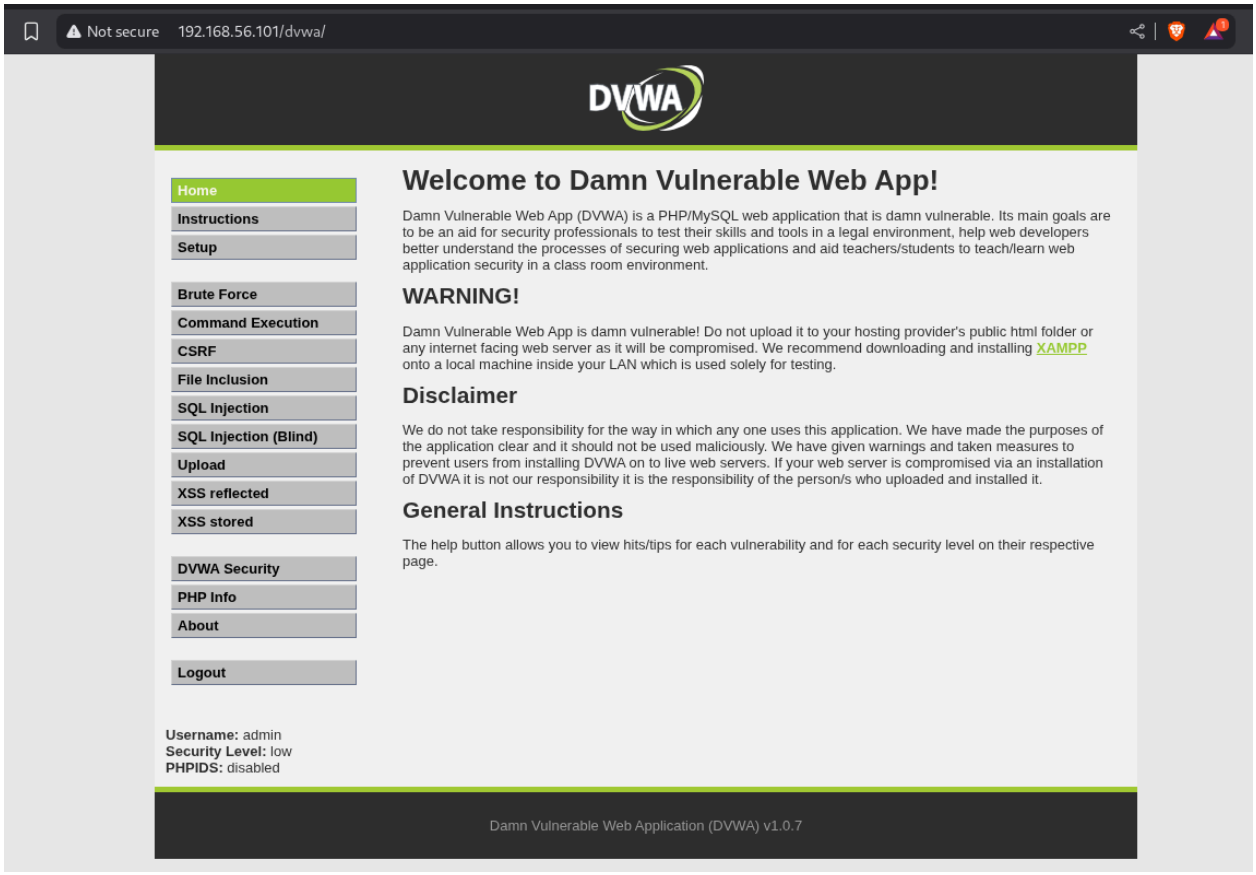
Objective: Learn about common web application vulnerabilities by analysing a simple web application. This task will help us understand how attackers can exploit weaknesses in web applications.

Tools used: OWASP ZAP, Metasploitable.

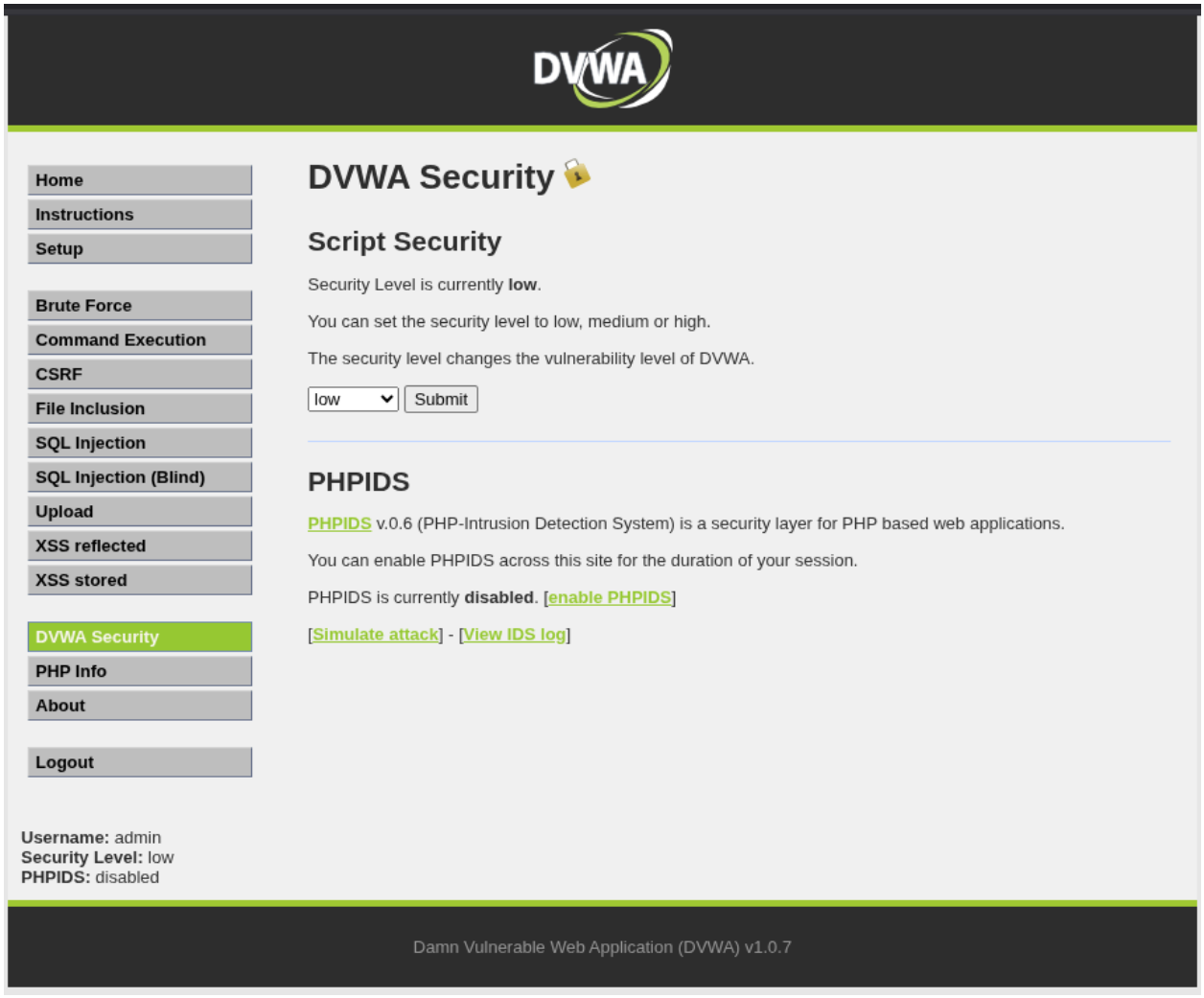
For this exercise, I have installed a vulnerable machine on my VirtualBox called "**Metasploitable 2**" which I think would be perfect for practice!

For my first exercise, I had launched a Metasploitable virtual machine and then checked what my IP was using the **ifconfig** command. After that I headed over to **DVWA** which stands for "**Damn Vulnerable Web App**". It is a PHP/MySQL web application that is susceptible. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a safe environment. We can easily access it through the following link "http://192.168.101/dvwa" (IP differs) which displays a login screen. It still has the default username and password of "**admin**" and "**password**" respectively. Once in, we can see the home page of DVWA as the following.

Redynox Internship



Since we are still starting out, we will get to learn about easy vulnerabilities and move to harder ones over time. You can do so by clicking on “**DVWA Security**” on the left hand side. Turning this setting on changes the vulnerability level of DVWA. It has a total of 3 settings, high, medium and low. As for our task, I’ll be setting it up to “**low**” for now and then click “**Submit**”.

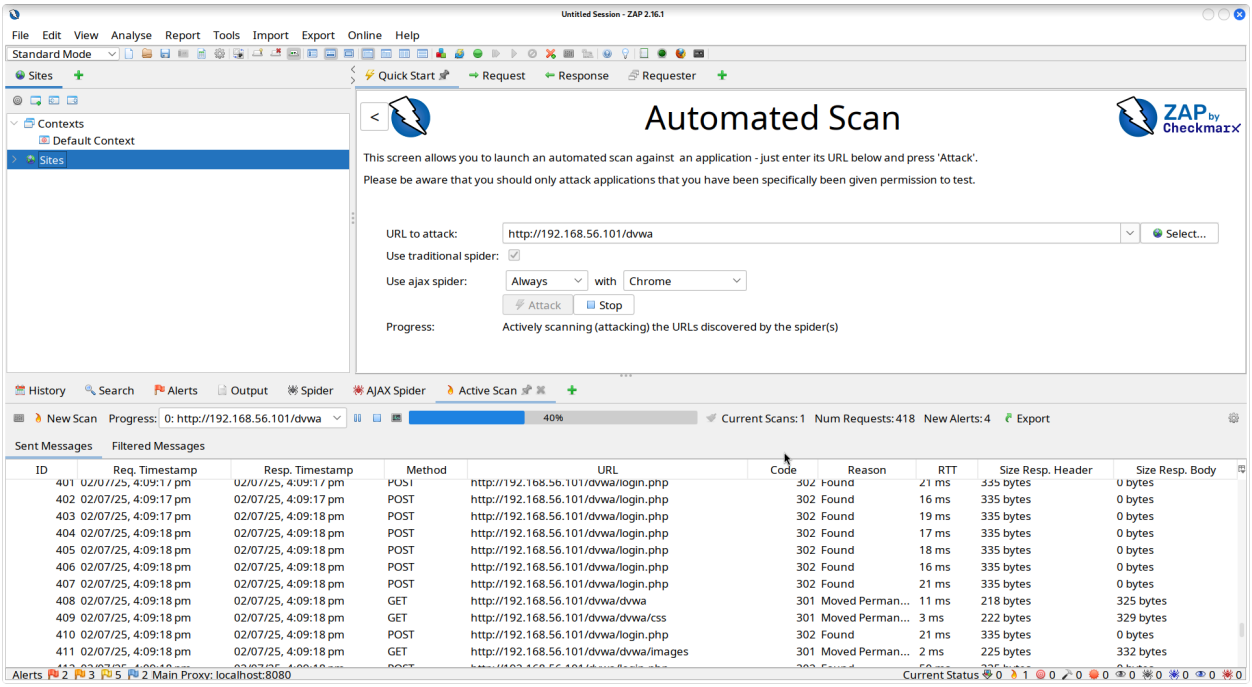


Once that's done, we are ready to perform some vulnerability analysis!

To perform a web scan, we will need a tool called "**OWASP ZAP".** You can download it from its official site "https://www.zaproxy.org/". On opening the app, we can see a URL input box and here we will enter in our DVWA website's link.

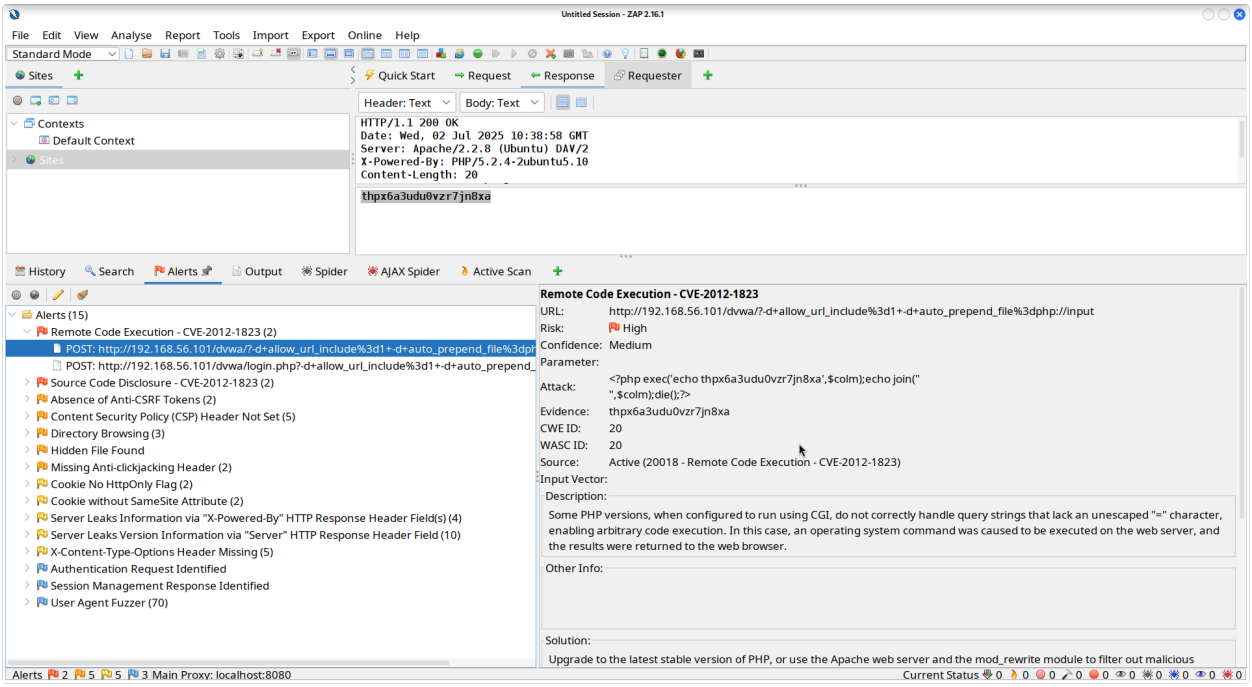
Then we click on "Attack" which starts the scan and let it run for a while since it will take a few minutes to scan the entire website for any vulnerabilities.

Redynox Internship



Once the scan is finished, on the bottom left hand side, we can see a tab for alerts which the app found when scanning. This tab displays a list of potential security vulnerabilities (alerts) discovered during a scan. It organizes alerts by risk level (High, Medium, Low) and provides details about each vulnerability, including the URL where it was found, the description of the vulnerability, and suggested solutions as well.

Redynox Internship

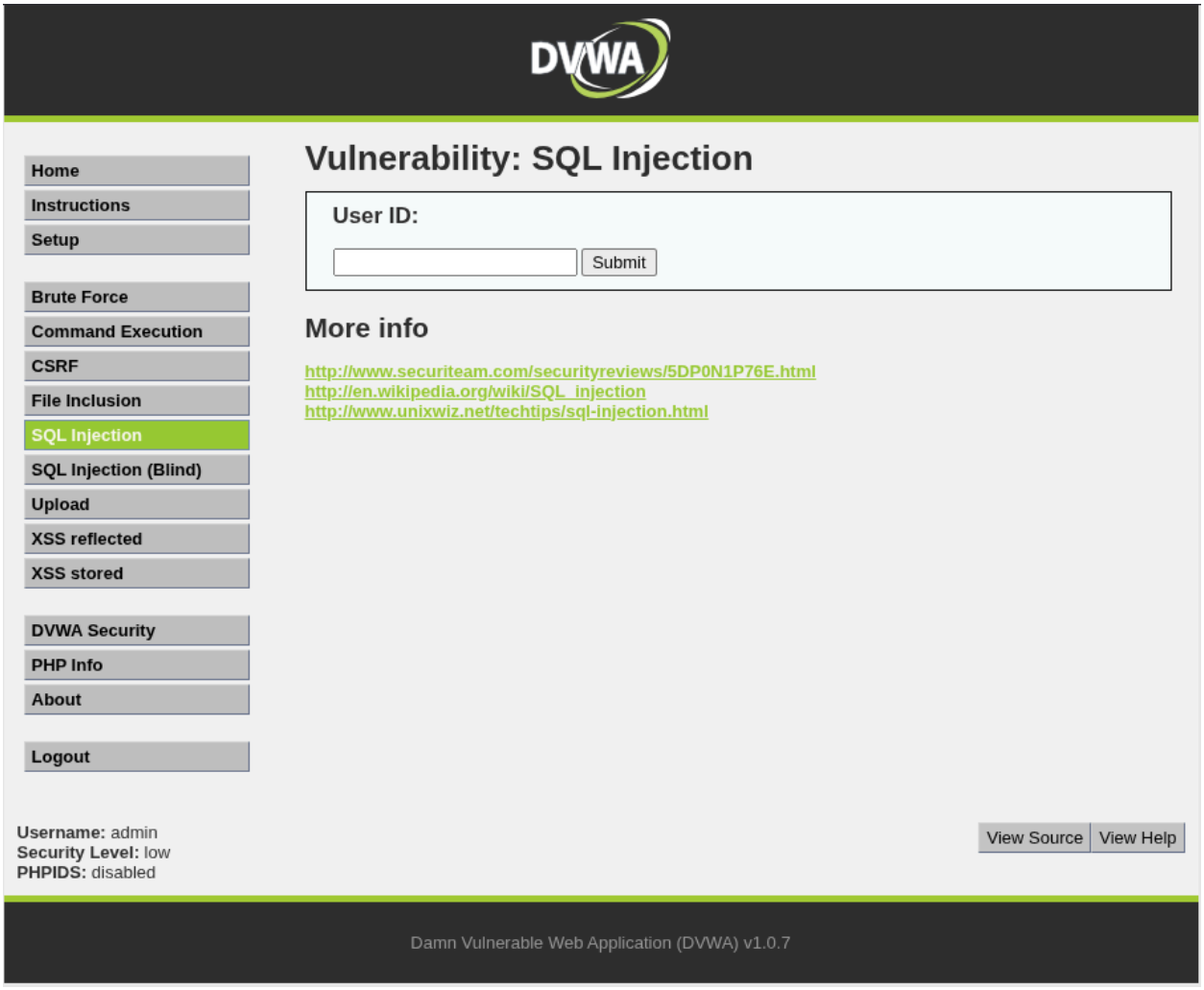


Now this was one way to scan for vulnerabilities which is more of an automated approach. Lets try to find some weaknesses in the website by manually performing some attacks on the website.

A. SQL INJECTION:

SQL injection is a code injection technique that attacks your database. It is one of the most common web hacking techniques and usually occurs when you provide an input area for the user to enter text like their username, password etc. So we will be exploiting this exact vulnerability. For this, on the DVWA website, we select the option for “SQL Injection” on the left hand side. After selecting it should display an input box on the page to enter our malicious code.

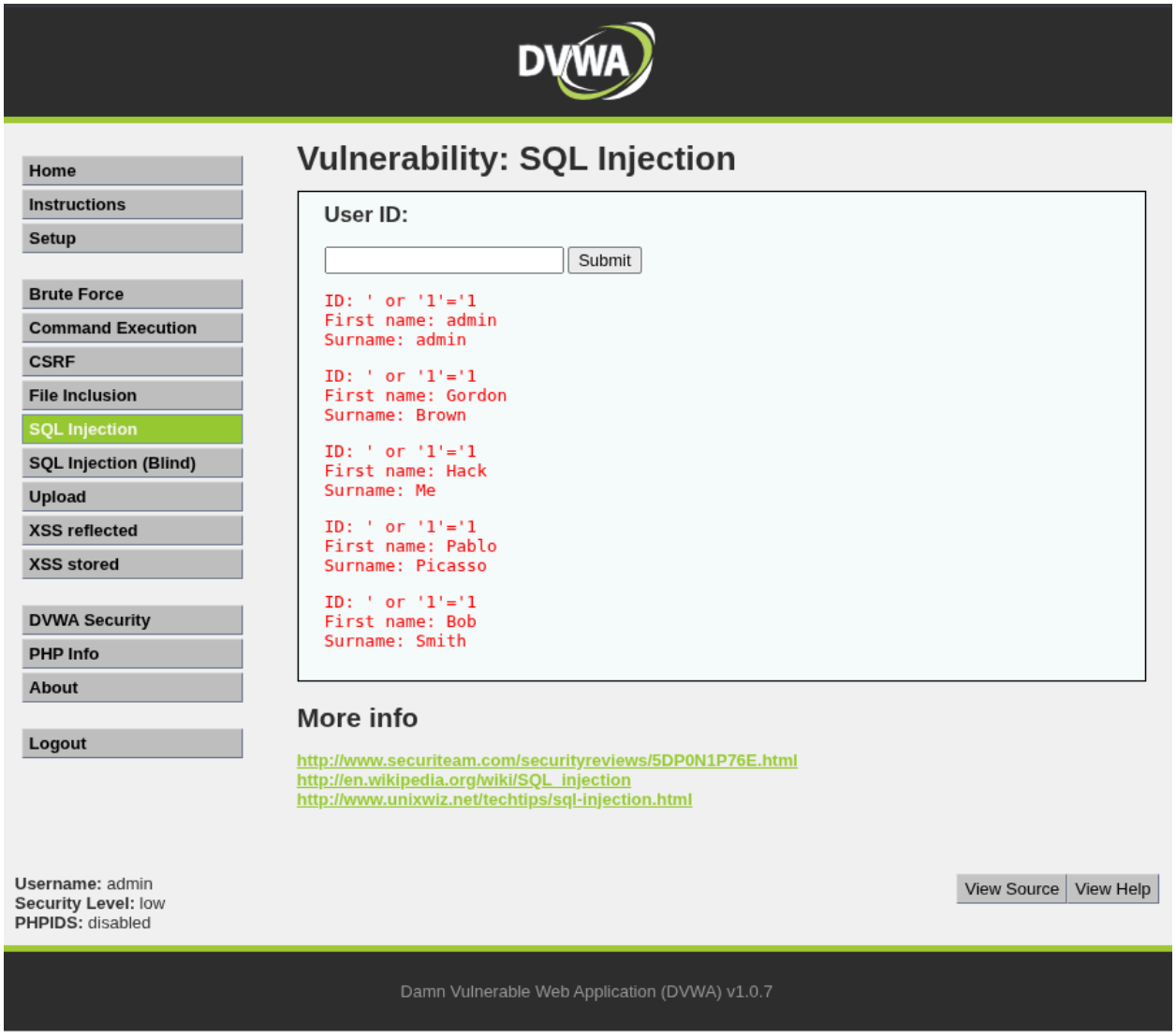
Redynox Internship



Here, in the input field, I entered a code "' or '1'='1" which when submitted, displayed the information regarding other user's first and last name. The code specifically is a malicious SQL query which always evaluates true i.e "1=1". Since '1'='1' is always true, the query ignores the real username/password check and returns the first row from the database.

This basically means that an attacker can log in without knowing any valid credentials of any user accounts and with an SQL injection attack be able to see confidential information of employees like company email addresses and their passwords listed right away for them.

The multiple records that were returned shows that our attempt for SQL injection was successful.



Ways to defend it:

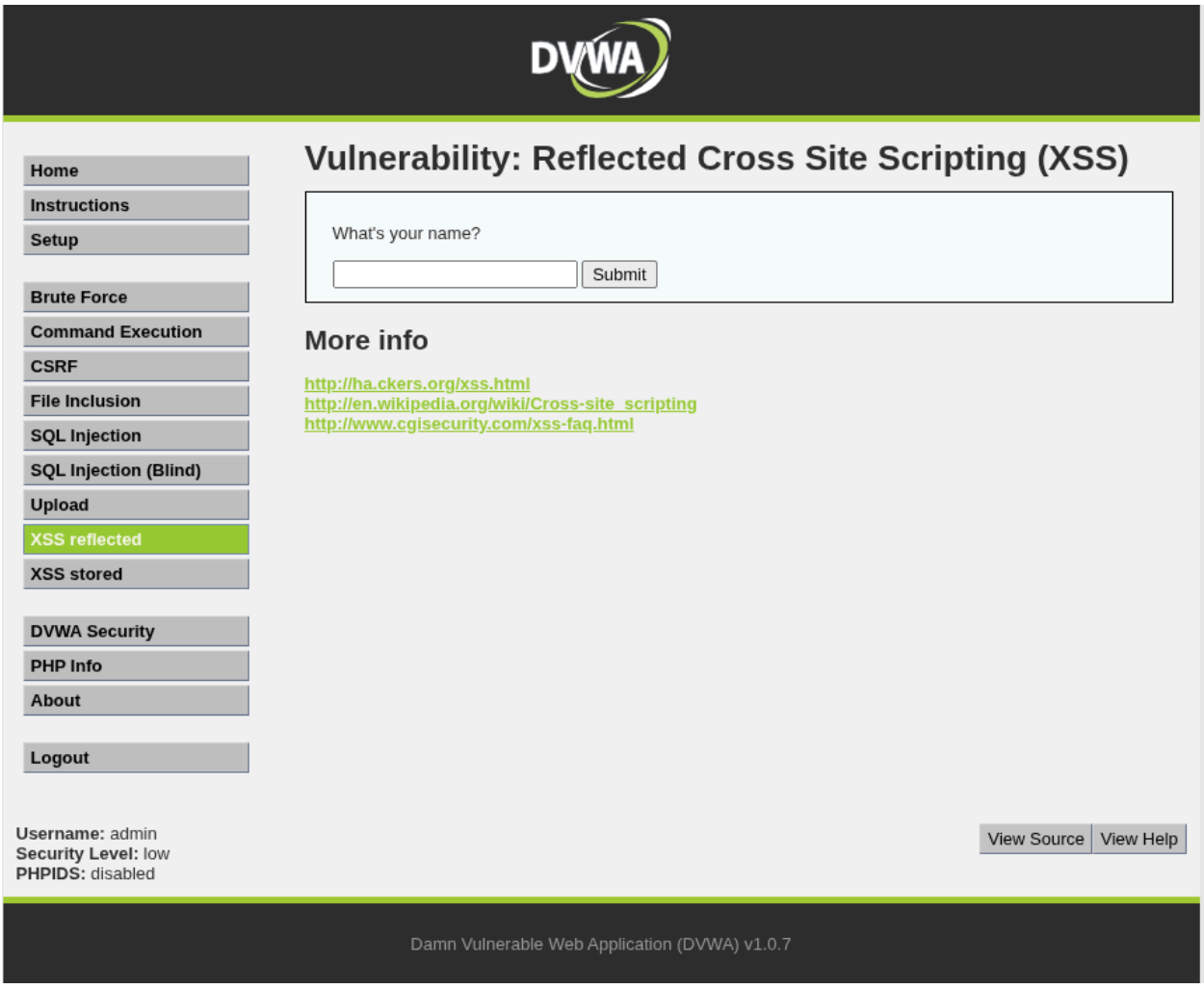
- Using prepared statements and parameterized queries.
- Having input validation setup so as to skip out any special characters attackers might input.
- Principle of least privilege.

Redynox Internship

B. CROSS-SITE SCRIPTING

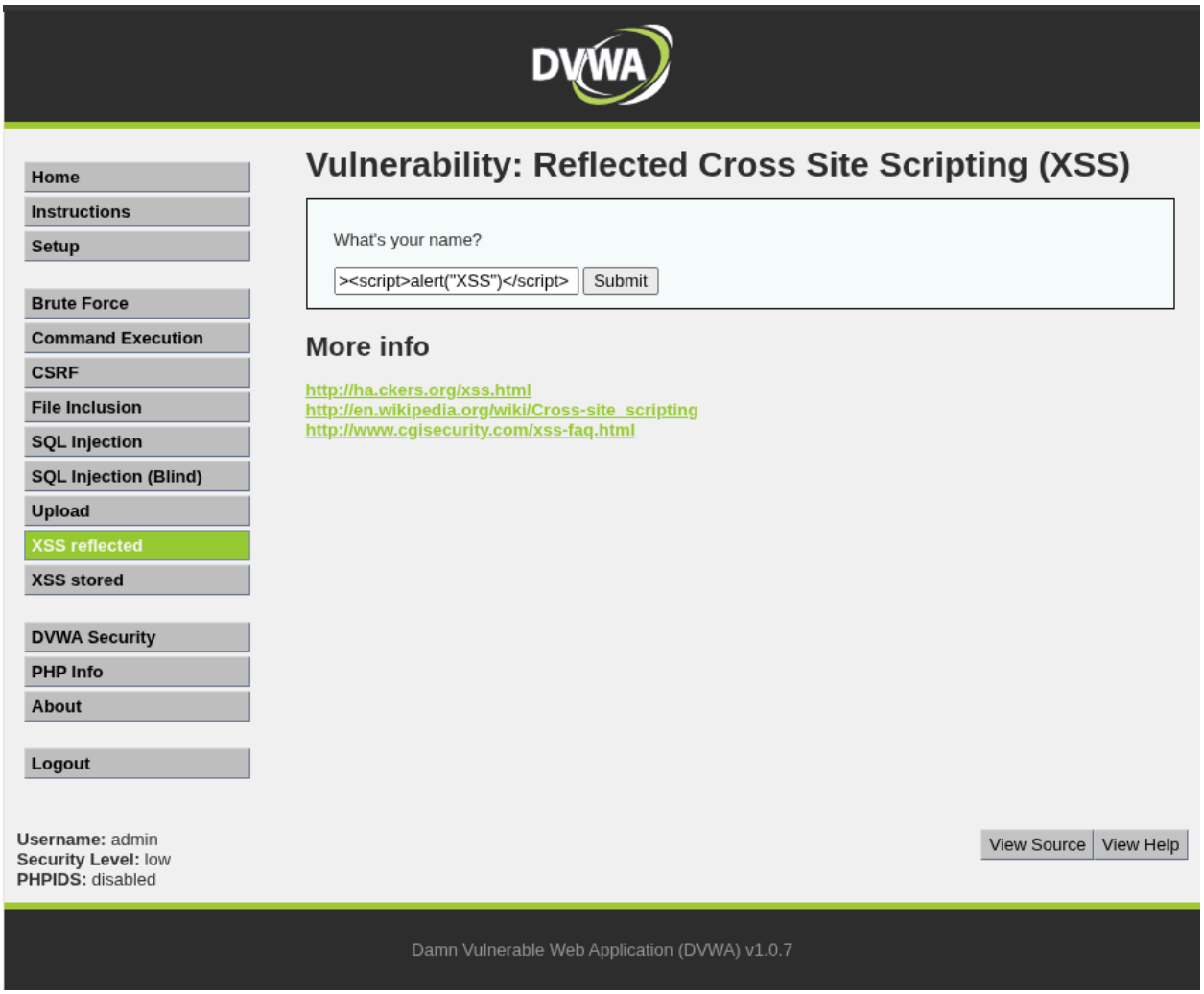
Cross site Scripting(XSS) is another common web vulnerability that attackers use to target a user's browser. It may seem similar to SQLi but they both target different parts of a web app and are used for different purposes. There's different types of XSS attacks but for this exercise, we will be performing an "**XSS- Reflected**" attack.

On the left sidebar, click on XSS(Reflected)



Redynox Internship

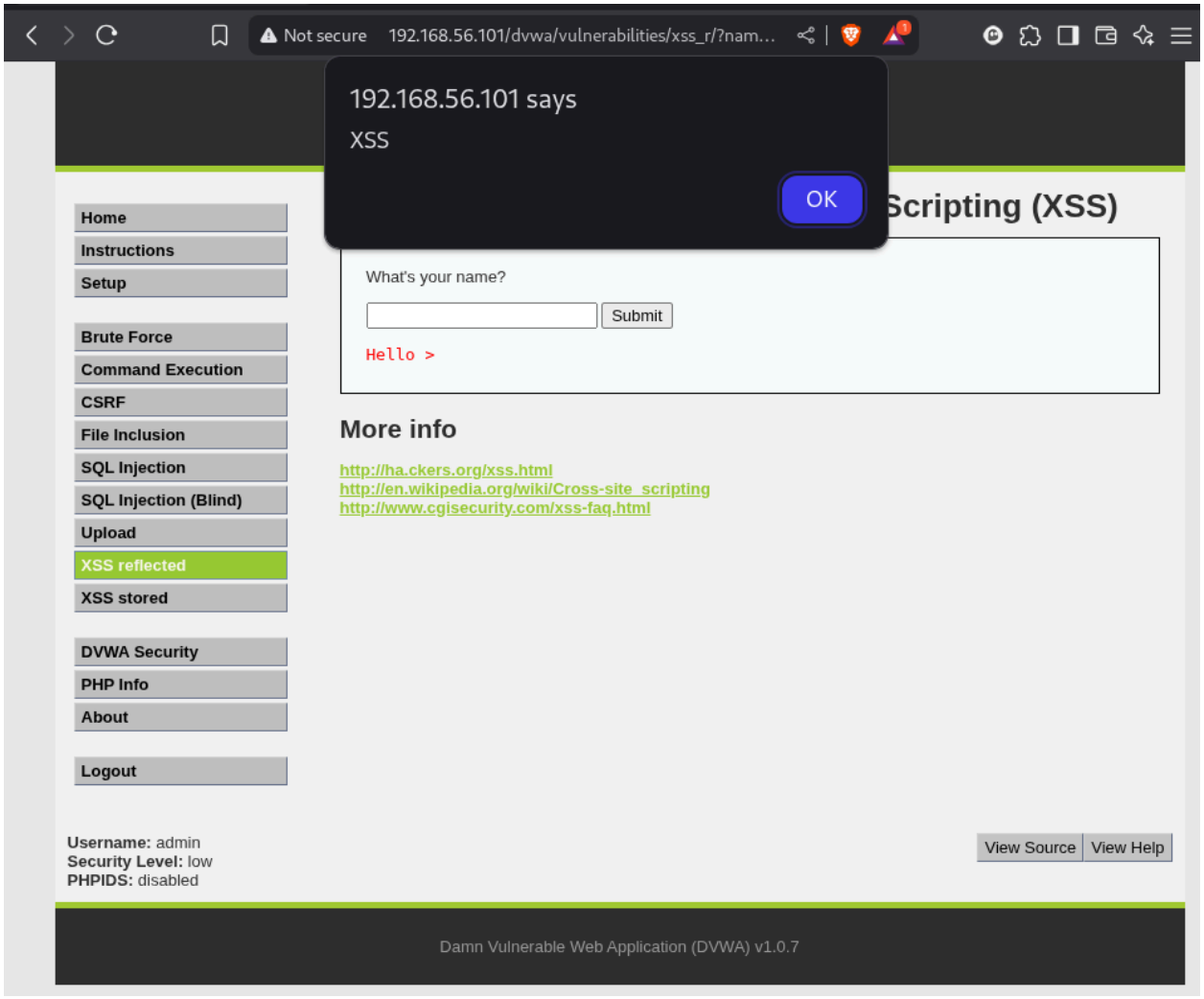
In the input field, I will be entering a javascript code which is not malicious but instead the website will take my prompt and it will act as a code that is inbuilt to the website. For example, I entered “><script>alert(“XSS”)</script>.



Once we click on submit, your browser will not display a popup with the word “XSS” in it.

This popup is proof that the website is vulnerable to cross site scripting and instead of our payload an attacker may be able to execute actual malicious code to get sensitive info like cookies, session tokens etc.

Redynox Internship



Ways to defend:

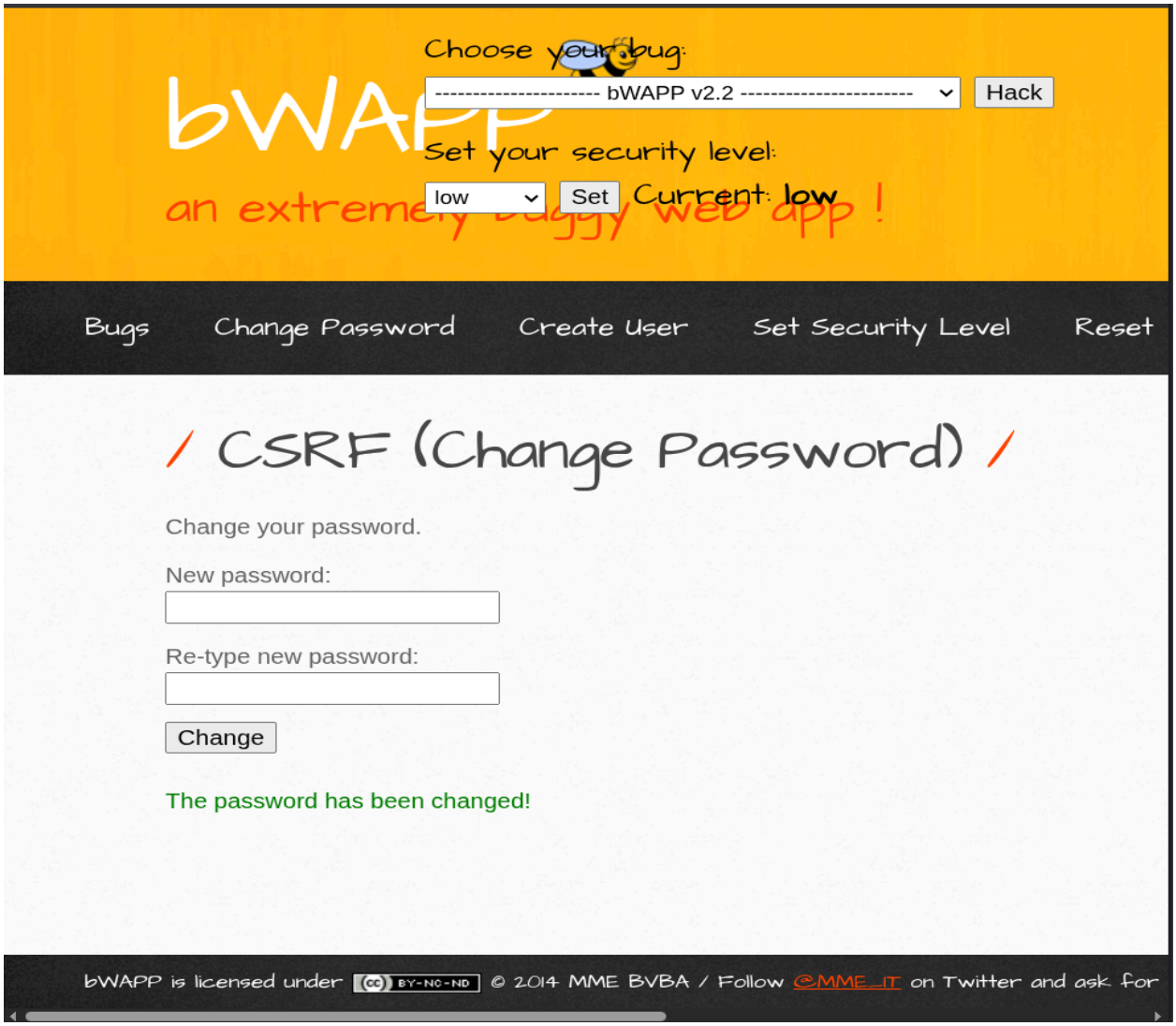
- Using Content Security Policy(CSP) (Defines what content is allowed to load on a webpage)
- Sanitizing/Validating input and output.
- Preventing Javascript from accessing session cookies by using HTTPOnly Cookies.

Redynox Internship

C. CSRF(Cross-Site Request Forgery)

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated/logged in. With a little help of social engineering (such as sending a link via email or chat), a threat actor may trick the users of a web application into executing actions of the actor's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth.

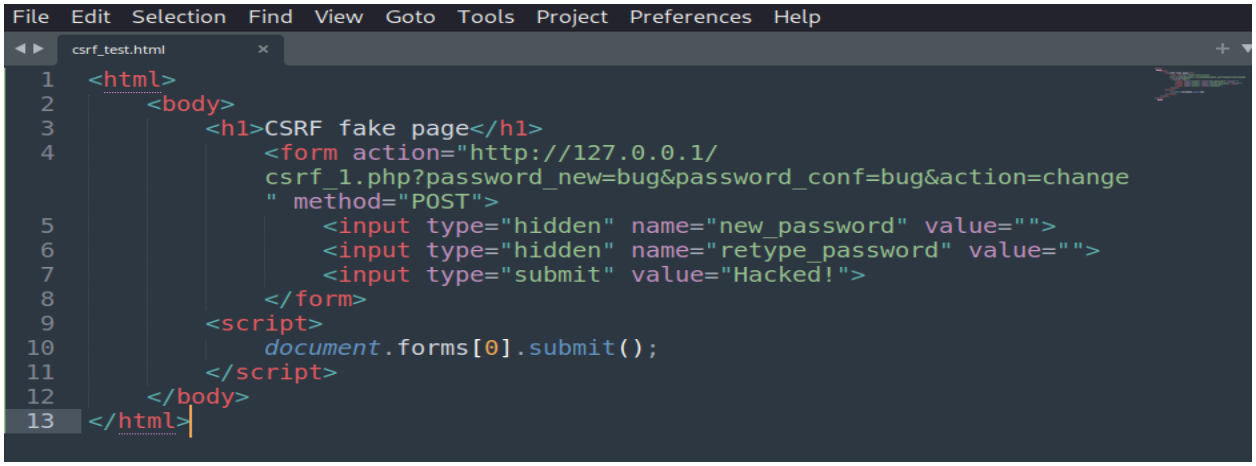
To test this out, we will be using bWAPP instead, which is also a buggy web app to practice and exploit vulnerabilities. I have installed it using **Docker** so it can run right on my localhost without having the need for the internet. The default username and password for the site are "**bee**" and "**bug**" respectively. After logging in, on the left hand drop down security menu choose the security level as **low** and from the drop-down bug menu select "**CSRF - Change password**".



After entering the new password and retyping it again to finally click on "**Change**" we can observe in the URL our changed password! This means that the application is letting you change your password via GET-requests. This flaw can easily be used by attackers to exploit your web app.

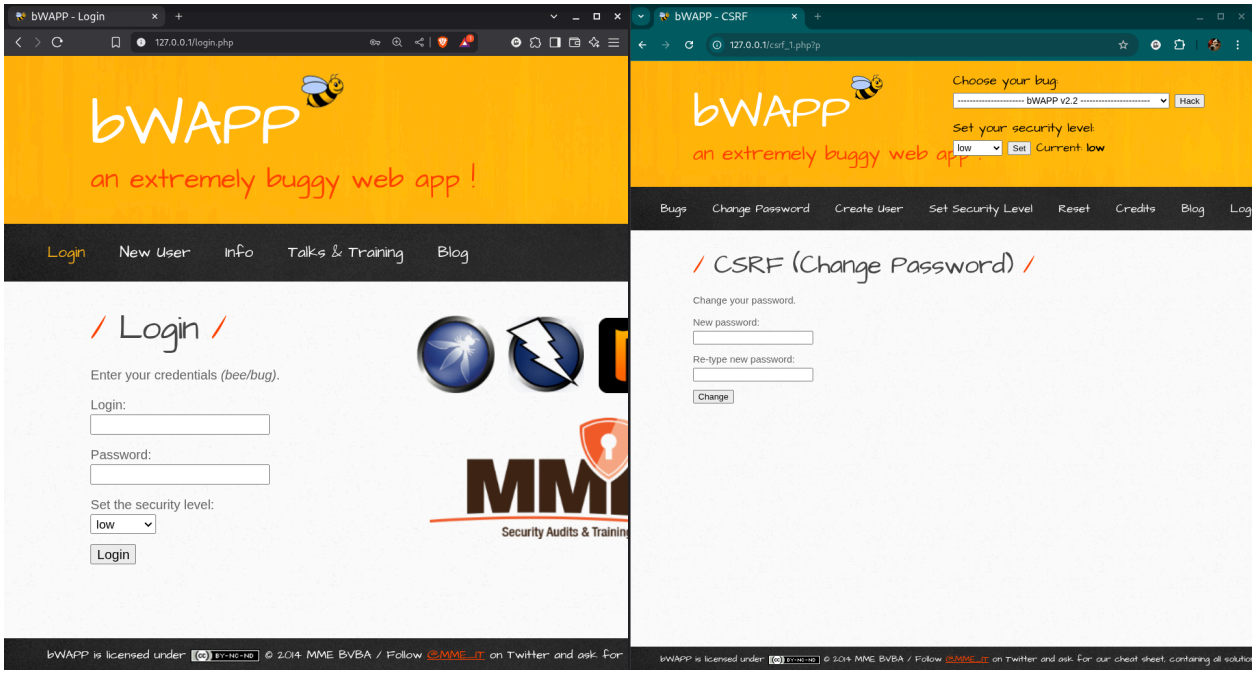
Redynox Internship

To test it out, I created a dummy html file which mimics the bWAPP web application to see if I am able to change passwords from this dummy file which in turn affects the original web app. Wrote a basic html code for some GUI.

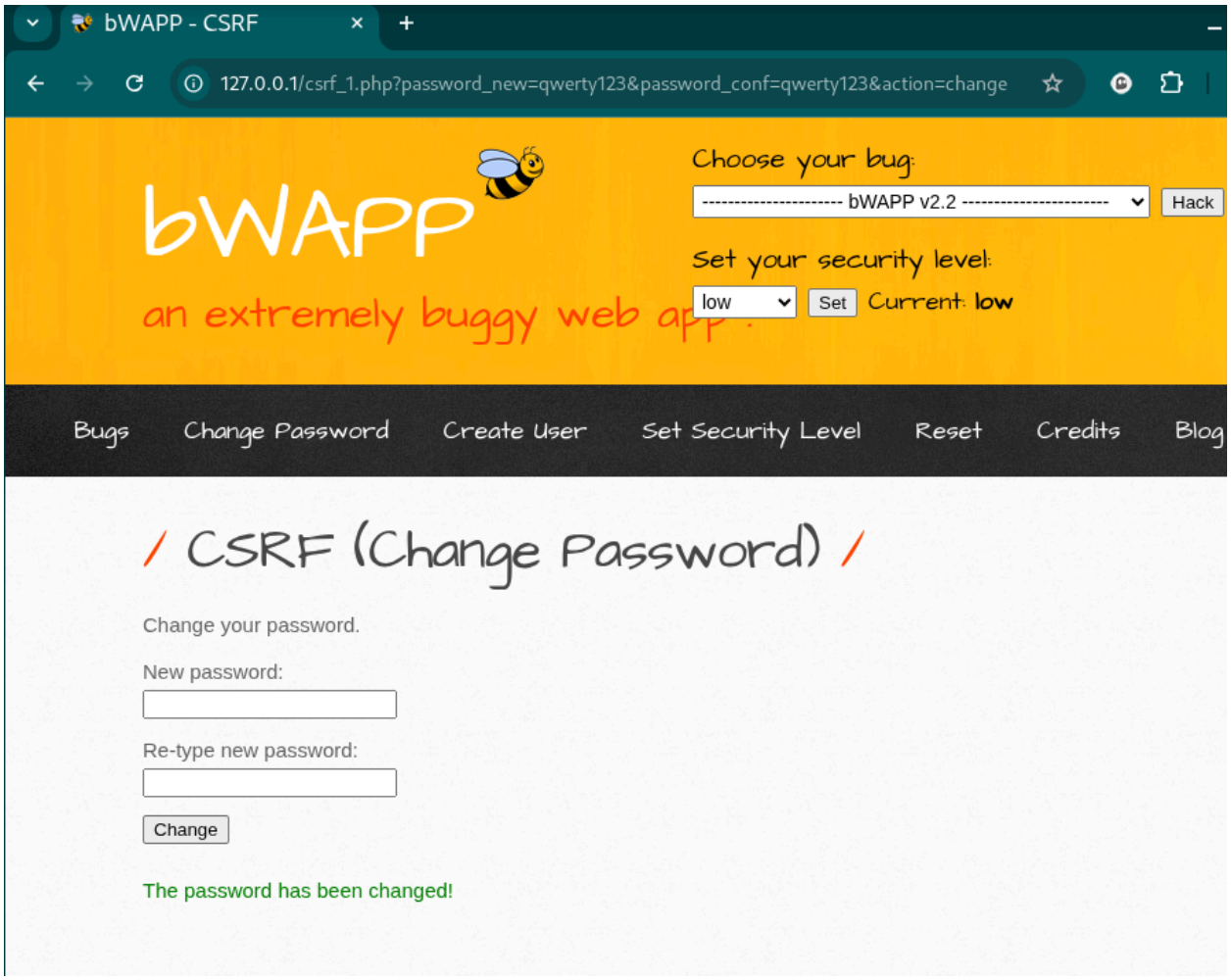


Redynox Internship

Here, I have provided a form action to Post the details provided in the form to the bWAPP app I am hosting on localhost(opened on Brave Browser). After opening this html file in a different browser(Chrome), it looks the same as the real one without the icons for other apps.



Now, we can try changing the password for the login on the Chrome browser where we have our local html opened and see if it actually affects the original web app.



Once clicking on **“Change”** we can confirm that the password was changed by the prompt **“The password has been changed”**, another way to confirm is by checking the URL, which we can confirm that the new password set is now “**qwerty123**”.

Lets try to login on the original web app which we have opened on our Brave Browser with the password “**qwerty123**” which we set earlier.

And voila! We are in! Hence using GET methods to change states is harmful. GET requests should never be allowed to perform actions, especially sensitive ones like password resets. Which is why OWASP also recommends that all state-changing actions must use POST methods and to include CSRF tokens to validate if the request is from a legit site.

Redynox Internship

Some popular frameworks that use standardized header names for CSRF protection:

- X-CSRF-Token - Ruby on Rails, Laravel, Django
- X-XSRF-Token - AngularJS
- CSRF-Token - Express.js (csurf middleware)
- X-CSRFToken - Django

Ways to Defend:

- Using CSRF tokens which confirms the legitimacy of a request.
- Origin Checks to detect illegitimate/suspicious requests
- Making use of CORS(cross-origin requests) controls to prevent untrusted site's access to sensitive information.