Web Application Vulnerability Assessment in a Controlled Lab Environment

Summary
This project evaluates common web application vulnerabilities in a controlled lab environment. The purpose is to understand how insecure design leads to security risks and how these vulnerabilities can be mitigated using secure development practices

Task 1

This task investigates reflected Cross-Site Scripting (XSS) vulnerabilities within a provided web application. The objective is to identify areas where user input is improperly handled and reflected back to the browser without adequate sanitization, allowing malicious scripts to execute. Such vulnerabilities pose a significant security risk, as they may lead to the exposure of session cookies, authentication tokens, and sensitive user information.

A reflected XSS attack is also known as Cross Site Scripting which is a vulnerability that is usually found in web applications. The reflected XSS injection attack is where an attacker would inject malicious code by executing the script which would give the attacker access to cookies, session tokens and other sensitive information that will be stored by the browser and is used with the website as it believes that the script is a reliable source.

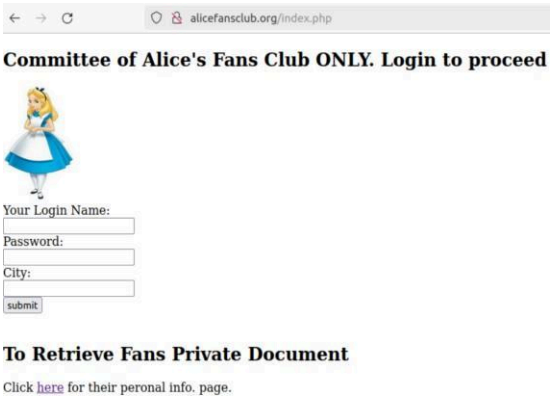


Figure 1 represents the home page of the website that was provided to us in the assignment specifications.

In figure 1, when we log in into the website that was provided to us there can be reflected XSS attacks that can occur. In this case, a user will be able to inject an attack into the fields that are provided in figure 1 which is the login name, password or the city page. This will be done by inserting the command of <script>alert (“XSS Exploit”) </script> which may allow us to insert malicious script where the attacker would be able to access sensitive information, session tokens and also users’ cookies.

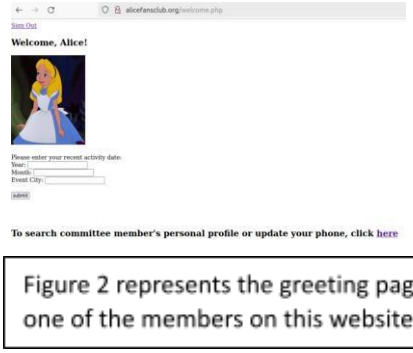


Figure 2 represents the greeting page of one of the members on this website.

In figure 2, we can see where a reflected XSS attack could probably take place which would be on the greeting page of Alice on the website. Here there are user input boxes that will authenticate the user or block those users who do not have access to the website. Therefore, we can assume that a reflected XSS injection can not take place and therefore a XSS vulnerability would not exist.

Figure 3 represents the page where Alice would be able to update her own phone number and access the information of all the other members.

In figure 3, we can see another place where a reflected XSS attack can exist where users can exist is where we can extract the user information of all users and also update the phone number. In this case, we would assume that a reflected XSS injection would take place.

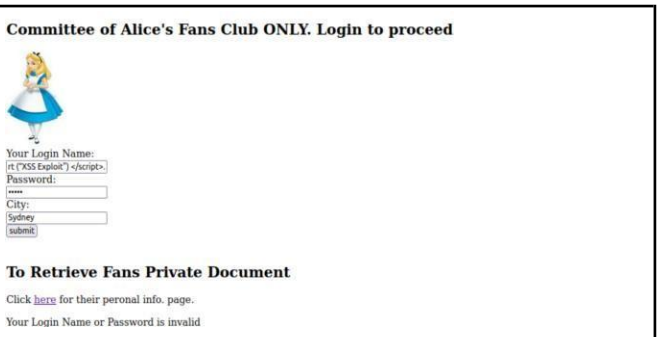
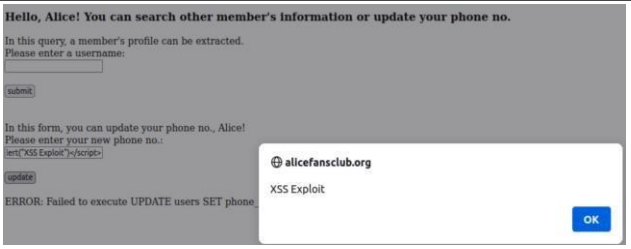
Below is a report which would demonstrate the reflected XSS attacks that have already been mentioned in the previous task of task 1a.

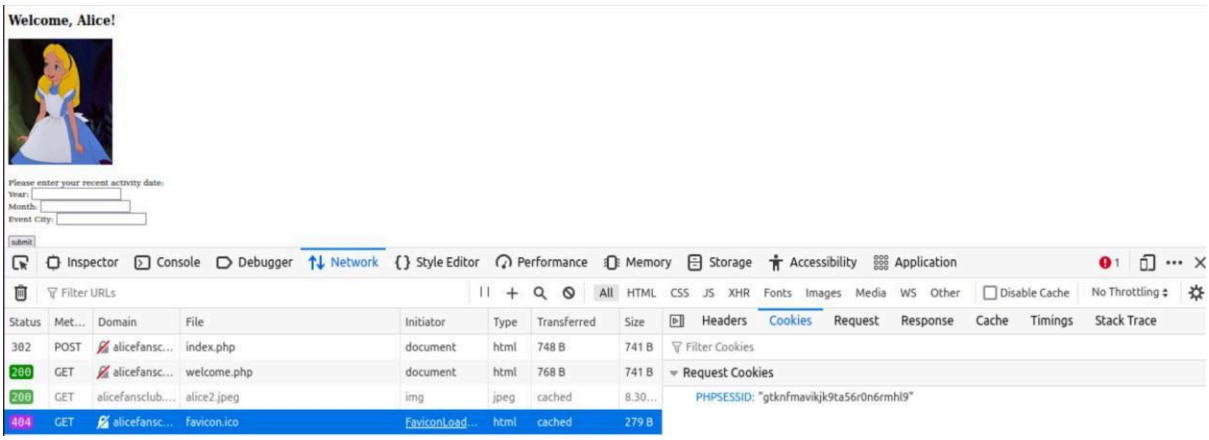
Figure 4 shows that the connection of the website is not secure

We can identify a vulnerability in Figure 4 when the web application is accessed through a browser, indicating that the connection is not secure.It can be deduced that the connection was not secure for that web page which will alert the users that the information that is sent and received from the page will be unprotected and therefore it will be prone to attacks where the data can be modified.

Below are the test results in a table that will allow us whether an XSS vulnerability exists or not in the different parts of the website that were given to us. In order to test whether it is a XSS vulnerability or not, we will add the command of <script>alert(“XSS Exploit”)</script>. This will run the alert function which will then open a new window with the provided message of XSS Exploit to show that a vulnerability exists.

Where does the vulnerability exist	Result (Whether it is or not a reflected XSS injection point.)	Interpretation

Home Page	No	 Looking at the image above, we can see that when a reflected XSS injection is executed there be no intercept that is made and therefore this is not a vulnerable injection point as it will only show the result of Your Login Name or Password is invalid instead of the pop up that there was a XSS exploit.
Greeting Page of Alice when she puts in her details on the home page.	Yes	 Looking at the image above, we can see that there will be a reflected XSS injection point in the welcome page of Alice. In this case, the user will receive a pop-up which would say “XSS Exploit” which is where the attacker will be able to locate the user’s cookies and the session tokens as well which might lead to them also accessing private information that they are not supposed to access.
Members updating their own phone number.	Yes	 Looking at the image above, we can see that there will be a reflected XSS injection point when the members are updating their own phone number. After the member has logged in or their login details have been found by the attacker, the attacker would then be able to use that information where they would be able to insert the malicious script which will allow the attacker to see information that they are not authorised to.



This image above shows the feasibility of receiving cookies at the attacker's application server.

To prevent reflected XSS vulnerabilities, the following security measures should be implemented:

- **Content Security Policy (CSP):** Restricts which scripts are allowed to execute on the page, reducing the impact of injected scripts.
- **Input Validation:** Ensures that all user input conforms to expected formats.
- **Output Encoding:** Encodes user input before rendering it in the browser to prevent script execution.
- **Secure Development Practices:** Avoid reflecting raw user input in responses whenever possible.

Task 2

This task examines an Insecure Direct Object Reference (IDOR) vulnerability within the web application. Although users are required to authenticate, the application fails to properly enforce authorization checks when accessing user-specific resources. As a result, an authenticated user (Grace) is able to access another user's (Camy's) private data by manipulating object identifiers in intercepted web requests.

Looking at the information below, we can see that Grace will be able to gain unauthorised access to Camy's personal private data.

Grace will be able to access the private information of Camy by exploiting an Insecure Direct Object Reference (IDOR) vulnerability. An IDOR vulnerability occurs when an application provides direct access to an object which will be based on the supplied input without enforcing any authorisation checks. This means that authenticated users can manipulate these inputs to access objects that they are not authorised to view or modify the message.

Below are steps that can be taken in order to exploit this vulnerability of IDOR that can be used by Grace to access the personal information of Camy.

The first step that needs to be done is that we will first have to log in as Grace on the website using the credentials that were given to us in the assignment specifications. This step will be completed using Firefox on the vm.

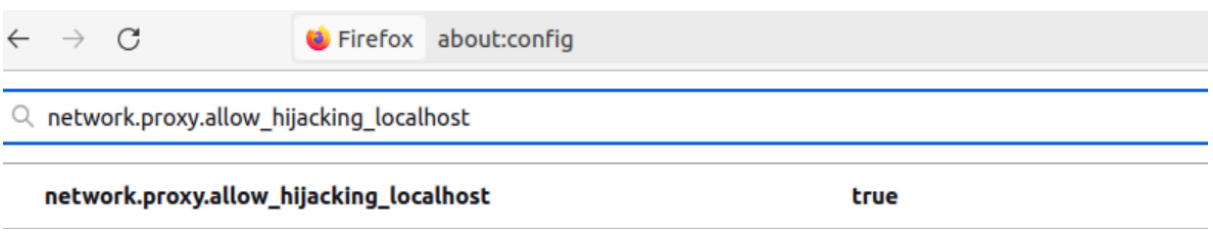


Figure 1

After we have logged in as Grace on the website that was given to us in the assignment specifications. The next process can be seen in figure 1, where by setting this to true it will allow Burp Suite to intercept and allow Grace to access the private information of Camy.

The next step that is done is that we will need to create a temporary project in Burp Suite and make sure that in the proxy tab that intercept is on. By doing so, the Burp Suite's proxy will be able to intercept HTTP and HTTPS requests between the browser and web servers.

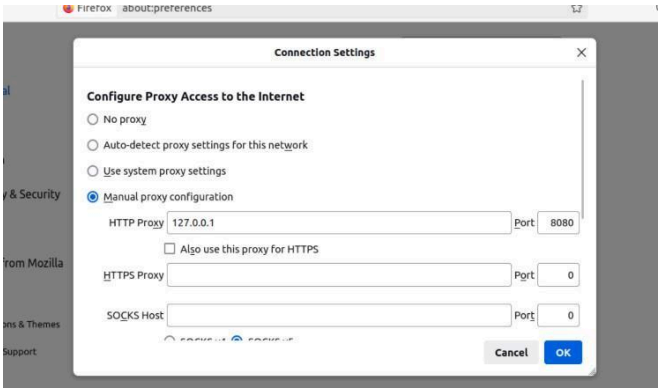


Figure 2

The next step can be seen in figure 2 where we can configure firefox by sending all the web traffic through Burp Suite's proxy that is running on 127.0.0.1 on the port of 8080. This is crucial for Burp Suite to modify and analyse the web traffic that happens on the firefox web browser.

After the configurations have been done to ensure that Burp Suite will intercept the message. When we are trying to view the document id 1 of Grace, figure 3 represents the intercepted message on Burp Suite in the Proxy intercept tab.

```
1  POST /csrf/personal_doc.php HTTP/1.1
2  Host: alicefansclub.org
3  User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:126.0)
   Gecko/20100101 Firefox/126.0
4  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,imag
   e/webp,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Content-Type: application/x-www-form-urlencoded
8  Content-Length: 310
9  Origin: http://alicefansclub.org
10 Connection: close
11 Referer: http://alicefansclub.org/csrf/personal_doc.php
12 Cookie: PHPSESSID=m0bskumlq4qtd8u8u587ibeerh
13 Upgrade-Insecure-Requests: 1
14 Priority: u=1
15
16 DocID=1&psID=m0bskumlq4qtd8u8u587ibeerh&uIDrx=3&sIDrx=
   020185c9cec3d7380939a2b2c2dfab188b5bd30ca5591b6a558e773e3bcabac042488
   da0671a8b44739124ee24a96b56733664826349c54a5de60512774ac74618b8025cf8
   a0600c2eda304f1702755fe4916ddf62659bcb47099e71c0095adb96fa6dda3d8d422
   31857851f5e706721512d76df0bcf5b0fcdfa307a948eaead
```

Figure 3

After Burp Suite has intercepted the message, we can modify the message directly by changing the uIDrx to 4 because of Camy's member id from 3.

Now that we have changed the member id to Camy, we can forward the request which will then allow Grace to access the private information of Camy even when she was trying to access her document id 1. The message that has been found can be seen in figure 4.

Contents of Document

Member Name: Camy, Private Document ID: 1:

I'm member Camy, This is my private document 1

I'm sure the following personal information is well protected from other group members by our Broken App.

Attackers have access to hundreds of millions of valid username and password combinations for credential stuffing, default administrative account lists, automated brute force, and dictionary attack tools. Session management attacks are well understood, particularly in relation to unexpired session tokens.

Figure 4

After Grace can access the private information of Camy when she is accessing her own document id 1, we can also do the same thing however when Grace is trying to access her document id 2. The steps that will be taken will be the same as what was used to intercept the document id 1. The message that has been found can be seen in figure 5.

Contents of Document

Member Name: Camy, Private Document ID: 2:

I'm member Camy, This is my private document 2

The prevalence of broken authentication is widespread due to the design and implementation of most identity and access controls. Session management is the bedrock of authentication and access controls, and is present in all stateful applications.
Attackers can detect broken authentication using manual means and exploit them using automated tools with password lists and dictionary attacks.

Figure 5

To prevent IDOR vulnerabilities, the following measures should be implemented:

- **Server-side Authorization Checks:** Ensure that every request verifies whether the authenticated user is authorized to access the requested object.
- **Indirect Object References:** Use randomized or session-based identifiers instead of exposing internal object IDs.
- **Access Control Enforcement:** Apply role-based or ownership-based access controls.
- **Security Testing:** Regularly test for authorization flaws during development.

Task 3A

Task 3A examines an SQL injection vulnerability in the authentication process, where unsanitized user input allows unauthorized access to database information.

3A.(i)

The SQL statement that is used in this web application is using select in order to authenticate the users.

In order to craft a malicious input for the username input box which will list all the information about the users. This process can be seen in figure 1 where we will use this command in order to retrieve information such as the title of each member, the salary, phone number and also the username that they use to log in.

The command that will be used in order to see all the information on all the users can be seen below

%’ and 1=0 union select null, null, concat(Title, 0x0a, Salary, 0x0a, Phone_No, 0x0a, username)from users #

Hello, Alice! You can search other member's information or update your phone no.

In this query, a member's profile can be extracted.
Please enter a username:

submit

```
ID: %' and 1=0 union select null,null, concat(Title,0x0a,Salary,0x0a,Phone_No,0x0a,username) from users #
Title:
Salary:
Phone No.: Supervisor
1000000
123456
alice

ID: %' and 1=0 union select null,null, concat(Title,0x0a,Salary,0x0a,Phone_No,0x0a,username) from users #
Title:
Salary:
Phone No.: Clerk
2000

agnes
```

Figure 1

3A.(ii)

In order to see all the tables in the database we can use the SQL command that can be seen below.

%’ and 1=0 union select null, null, table_name from information_schema_tables #

Hello, Alice! You can search other member's information or update your phone no.

In this query, a member's profile can be extracted.
Please enter a username:

submit

```
ID: %' and 1=0 union select null,null,  table_name from information_schema.tables #
Title:
Salary:
Phone No.: ALL_PLUGINS

ID: %' and 1=0 union select null,null,  table_name from information_schema.tables #
Title:
Salary:
Phone No.: APPLICABLE_ROLES

ID: %' and 1=0 union select null,null,  table_name from information_schema.tables #
Title:
Salary:
Phone No.: CHARACTER_SETS

ID: %' and 1=0 union select null,null,  table_name from information_schema.tables #
Title:
Salary:
Phone No.: CHECK_CONSTRAINTS

ID: %' and 1=0 union select null,null,  table_name from information_schema.tables #
Title:
Salary:
Phone No.: COLLATIONS

ID: %' and 1=0 union select null,null,  table_name from information_schema.tables #
Title:
Salary:
Phone No.: COLLATION_CHARACTER_SET_APPLICABILITY
```

Figure 2

This process can be seen in figure 2 where we will put the SQL command on the web application in order to view all the tables that are in the database. Please Note- that these are just some of the fields of the table that are in the database.

After we are able to find all the tables in the database in order to find all the information of all the users, we can now find the list of tables which will be inside the database which can be done by injecting a SQL command into the username input box. This process can be seen in figure 3.

The command that will be used in the web application can be seen below which would allow us to see the name of the database.

%’ or 0=0 union select null, null, database() #

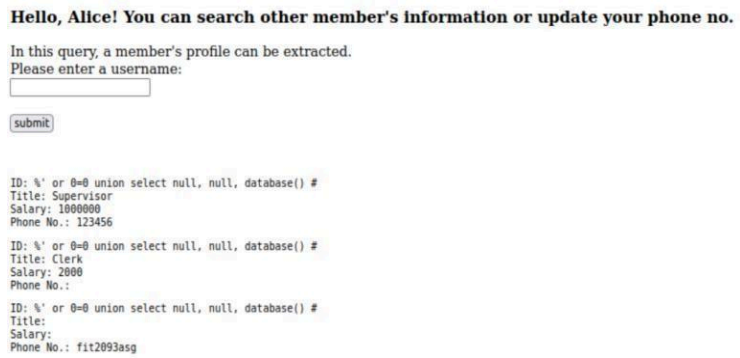


Figure 3

Looking at the information that has been retrieved from the SQL command that has been used, we can see that the name of the database would fit2093asg.

Below is the command that can be used in order to extract the personal information of the members such as the Title, Salary, Phone number. This process can be seen in figure 4 where this table will contain this information under the table “users.”

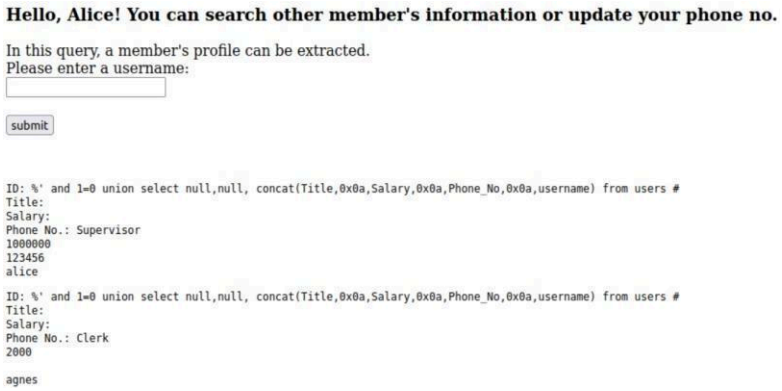


Figure 4

Task 3B

Task 3B examines an SQL injection vulnerability in the user profile update functionality, where improper input validation allows unauthorized modification of database records.

3B.(i)

The SQL statement that is used in this web application is using the update function in order for the members to update their phone numbers.

In order to exploit an SQL injection vulnerability in the phone update textbox, we can use the command which can be seen in below

1234’, salary=5000 WHERE username=’Alice’;#

In this case, we can see that we will be updating the salary of Alice to 5000 instead of her phone number in the phone update textbox.

3B.(ii)

We can craft a malicious input using the example of 1234', salary=5000 WHERE username='Alice';# which will attempt to update the salary of Alice to 5000

Figure 1

Figure 1 shows a screenshot of the member profile search page before any changes have been applied in the phone update.

After we have done this we will be able to add the malicious input of 1234', salary=5000 WHERE username='Alice';# into the figure 1 phone number update textbox.

After this step has been done, we can see that the result of adding the malicious input into the update phone number textbox can be seen in Figure 2 which states that the "Records are updated successfully."

Figure 2

Looking at the results that have been gathered in figure 2, It gives the response of "Records are updating successfully" after injecting the SQL code of 1234', salary=5000 WHERE username='Alice';#. This is seen as even though we are updating Alice's salary to 5000 within the update phone number textbox, it still says that the records have been updated successfully. This might have occurred due to there being a flaw or incomplete validation checks that might have occurred in the system which is failing to detect that we are updating the wrong input in the wrong textbox.

3B.(iii)

In order to resolve the issue of the system saying that the "Records are updated successfully" even though the user is attempting to update the salary field in the phone number update textbox, we could implement a validation check. This check would first validate if the user is updating the correct field that they are supposed to update in that textbox. By doing so, this will ensure that the user will not be able to update fields when we are trying to update another field.

What I Learned and Gained

Through this project, I gained hands-on experience identifying and exploiting real-world web application vulnerabilities, including XSS, IDOR, and SQL injection. I learned how weak authorization controls, unchecked input, and improper validation can expose sensitive data and allow unauthorized manipulation of application logic.

By using tools such as Burp Suite, Firefox, and a virtual machine environment, I developed practical skills in intercepting, modifying, and analyzing web requests. This project reinforced the importance of secure development practices such as input validation, output encoding, access control enforcement, and the use of Content Security Policy (CSP).

These skills are directly applicable in real-world security testing and highlight how careful application design can prevent critical security vulnerabilities.