Shopping List

CS3SP Coursework Assignment

Mehran Raja & Hani Hussain

220199384 & 220183497

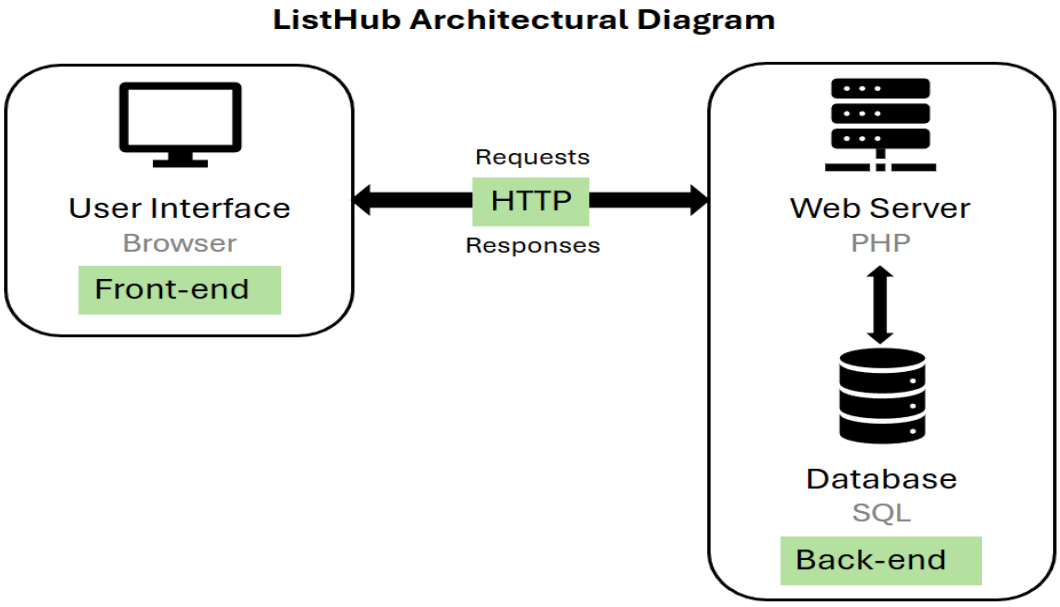
220199384@aston.ac.uk &
220183497@aston.ac.uk

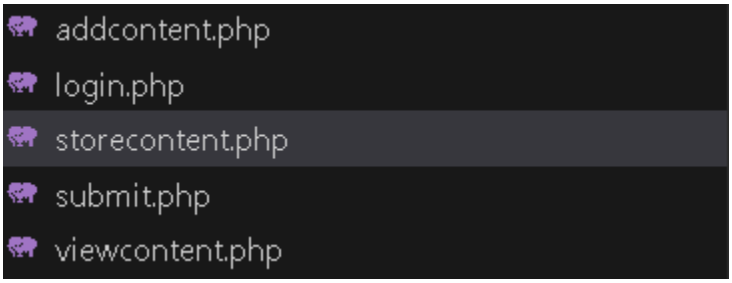
Both members contributed equally to both the report and the website.

AI was used for research only.

Introduction

ListHub is a web app with the purpose of allowing a user to sign in and create a personal shopping list that will be tied to their account. This report will show all mitigations made to do so. A functional version that has all functionality will be created, then a secure version of the functional version, showing what was changed. The functional requirements are for a user to be able to submit items into a list via a form, they will then be stored to their account using SQL. They will be able to navigate and view items, whilst also having a session attached to their user. I have attached a diagram below of the Architecture, giving a rough idea of how the website operates and is layered. The user interacts with a bowser to request from the PHP endpoints, then PHP scripts interact with the SQL database, responses are then returned as HTML. In my mitigation I will be going over SQL injections, XSS, IDOR and CSRF.



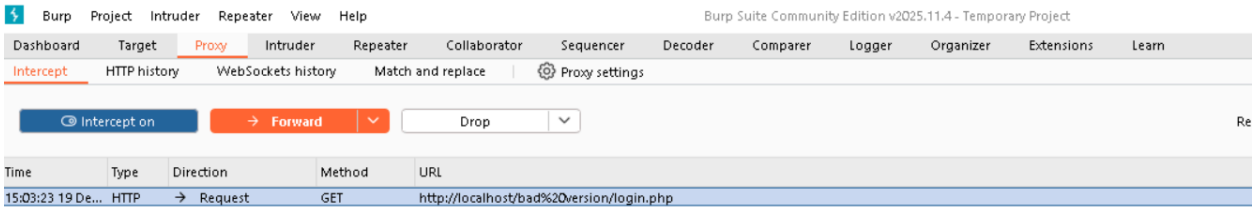


Baseline Implementations – Application 1

The first application I made was for the purpose of the functional aspects of the web application, with no security. This version was intentionally made insecure to show how vulnerabilities can occur in basic websites. The functional requirements would be; Users are able to login with an identifier, users can add items to a shopping list, users can view this list and the list is saved per user using SQL.

For the first application the user would input using a HTML form, then this is sent to the PHP web server, PHP then sends it to the database, which

are then returned to the users as HTML. This is with no security, and it is vulnerable to common attacks as shown below with the raw parameters.



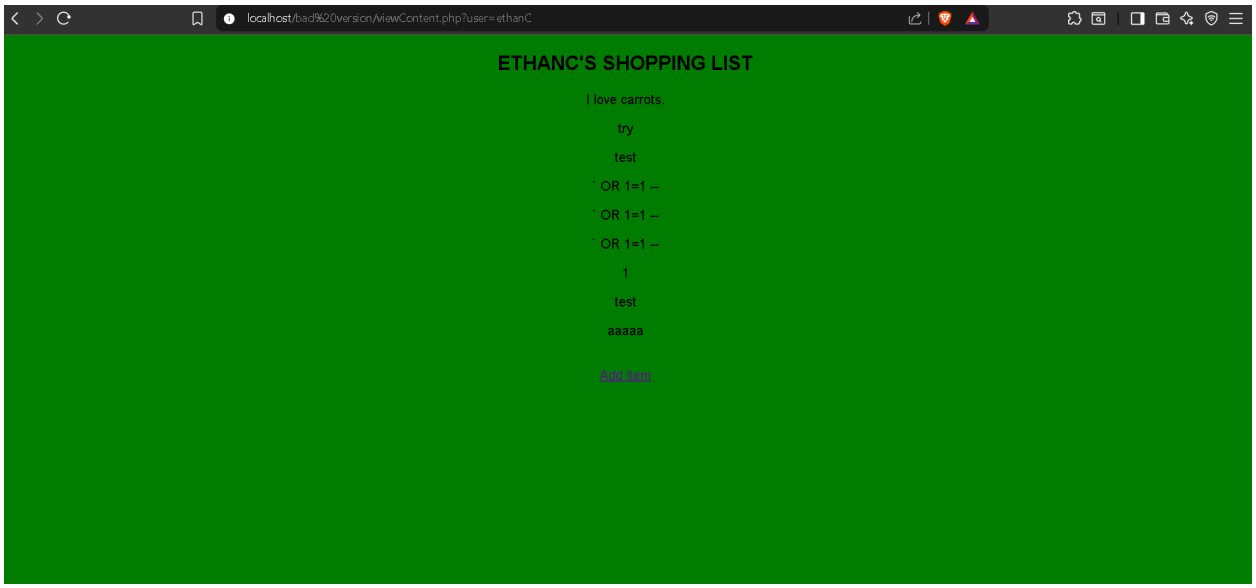
Vulnerability Analysis

In the basic application there were weak authentication checks. For example pages were able to be accessed without even checking the users login state and users were identified based on the URL rather than the session data.

```
$user = $_SESSION['username'];
```

This code would allow for attackers to view and edit other's data, meaning

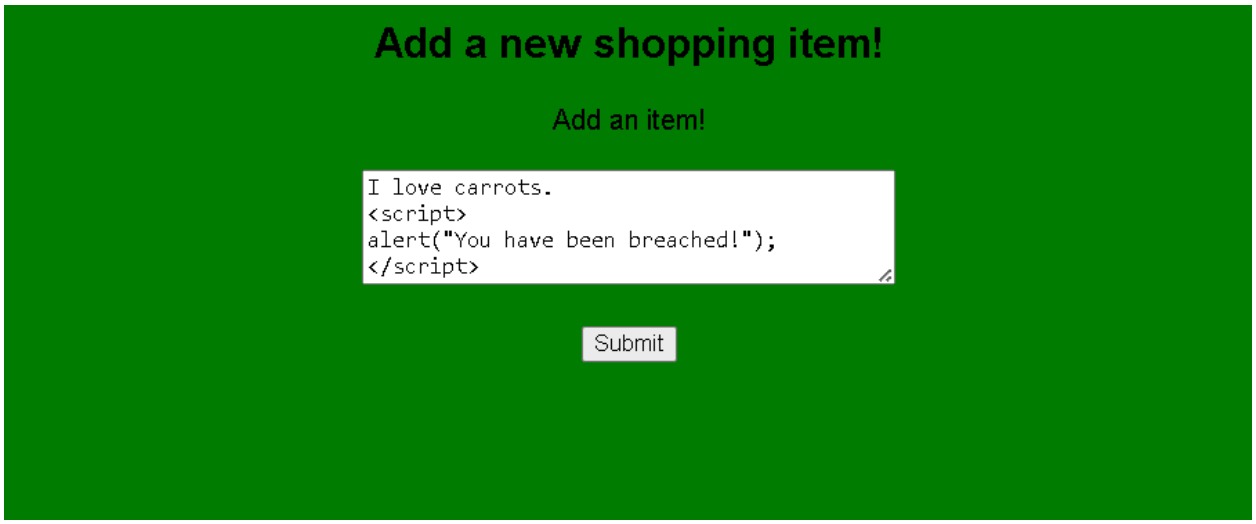
this would be classed as an Insecure Direct Object Reference (IDOR) vulnerability. Shown visible in the screenshot below you can see at the top it says “viewContent.php?user=ethanC” and that was enough to allow the attacker to view someone else’s data. This means the attackers can steal data and edit data as they please, they could then go on to sell this data too.

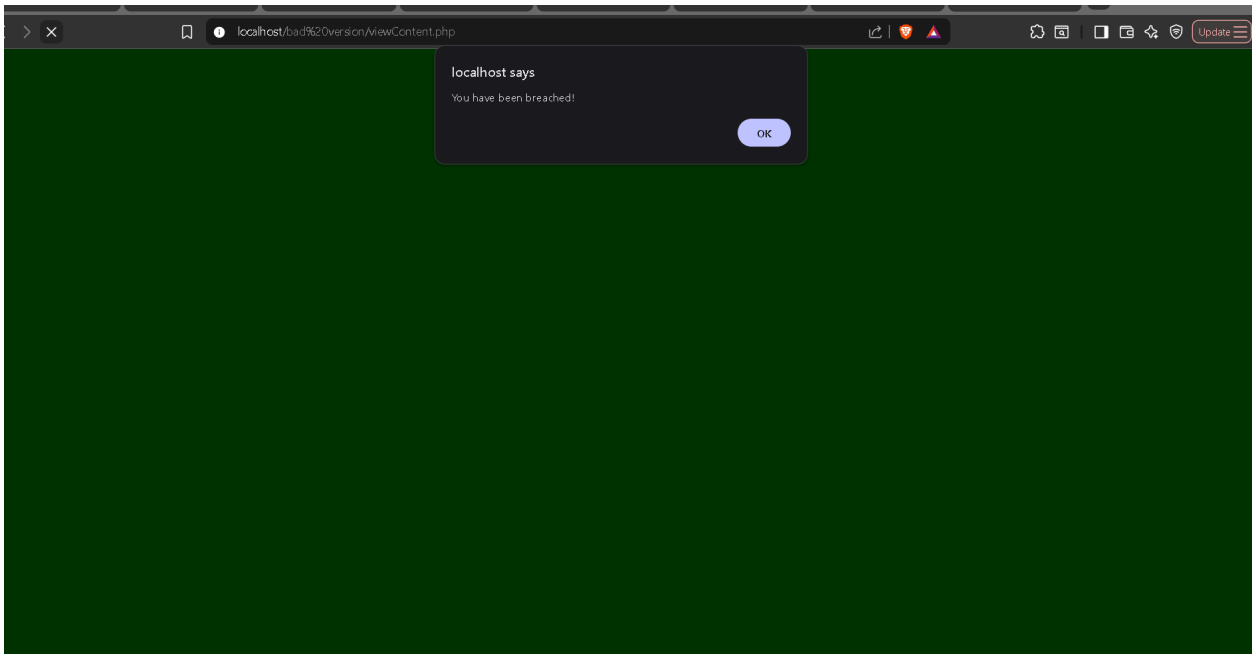


There was a clear lack of input validation in the first application. This was because the user’s inputs were directly passed through the SQL queries and were then reflected into the HTML responses with no sanitisation. Which looked like the below in the code.

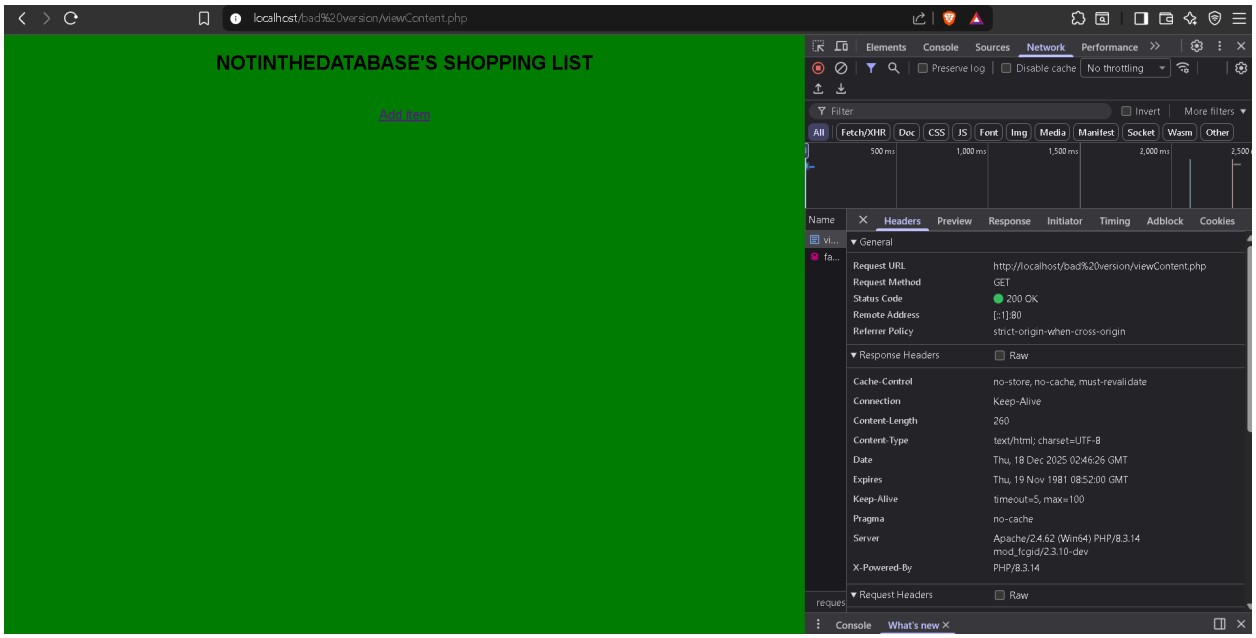
```
$sql = "INSERT INTO user_items (user, items) VALUES ('$user', '$thought')";
```

This let attackers inject SQL in the code, resulting to SQL Injection (SQLi) vulnerabilities. Leading to the following attacks, which can edit the HTML of the website.

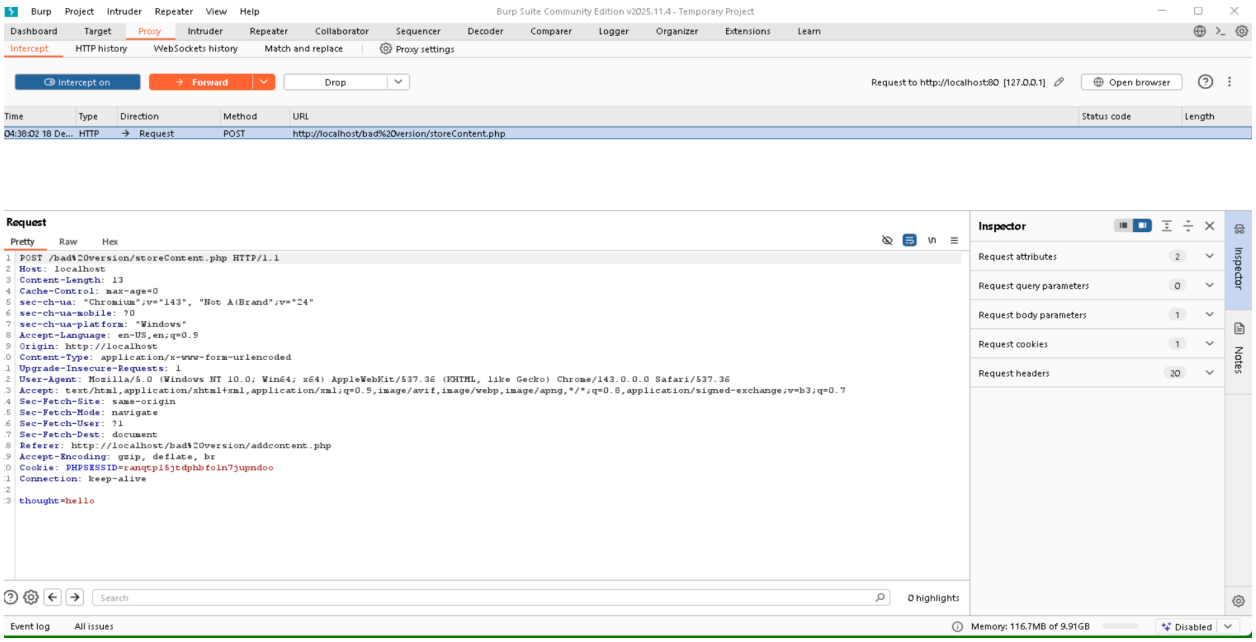


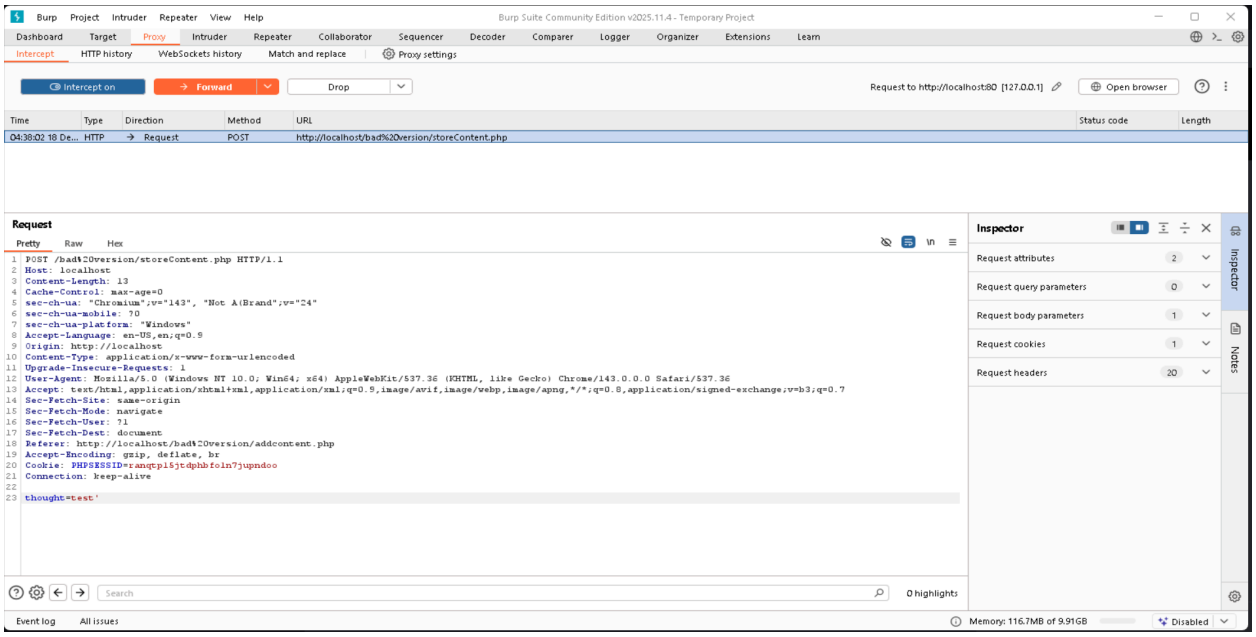


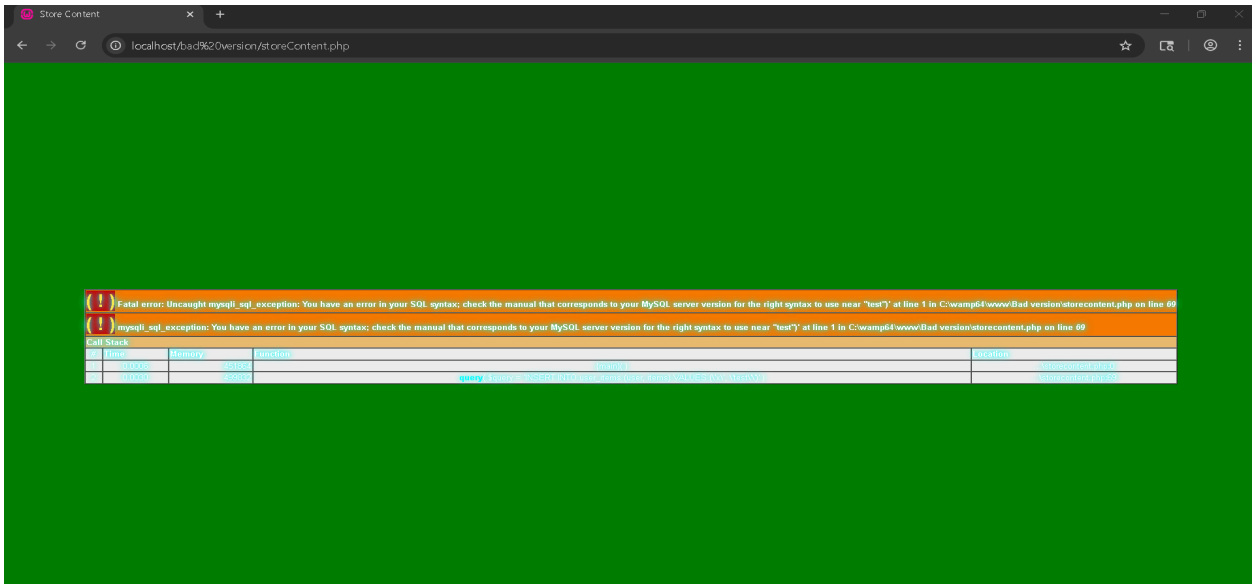
There were no security headers or any request protections. The insecure version did not have HTTP headers related to security, CSRF protection or session enforcements. This meant that requests could be forged and browser mitigations were not enabled. This is shown in the screenshot below. This means that requests can be forged by users.



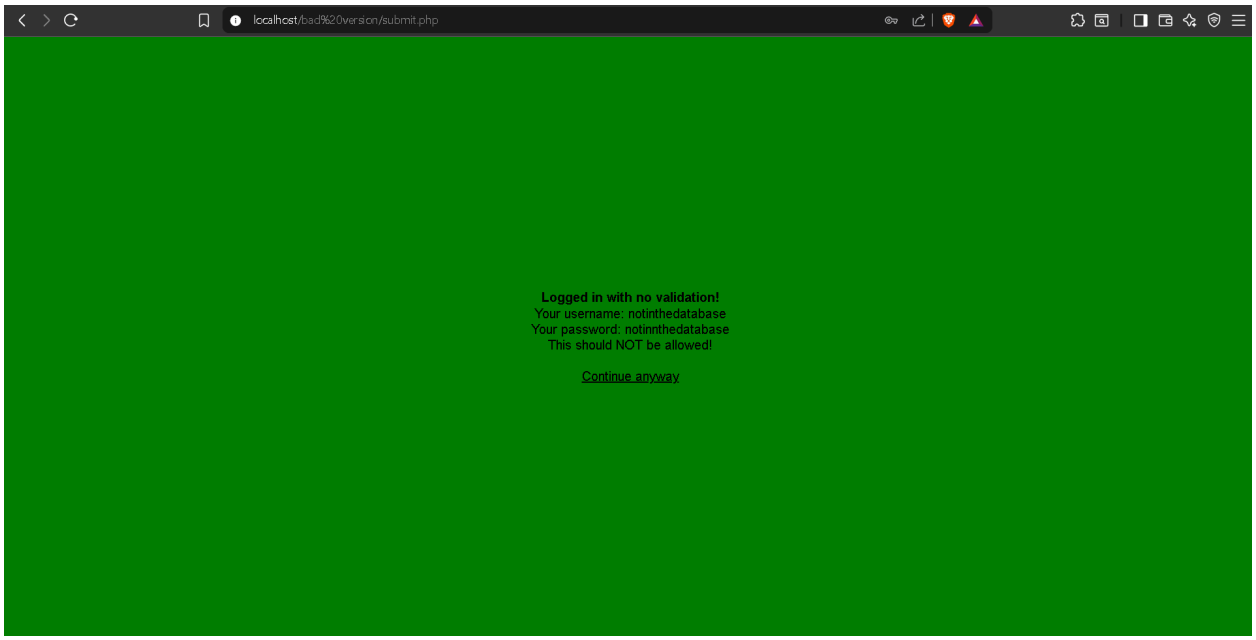
The below screenshots show how that because there is no CSRF protection, the request can be edited by Burp and then results into an SQL error. This means that users can just change their data and it would be able to cause an SQL error or cause for malicious data to be inserted. This can result in an SQL attack.


Burp Project Intruder Repeater View Help
Burp Suite Community Edition v2025.11.4 - Temporary Project
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on
Forward
Drop
Request to http://localhost:80 [127.0.0.1]
Open browser
Time Type Direction Method URL Status code Length
04:38:02 18 De... HTTP Request POST http://localhost/bad%20version/storeContent.php
Request
Pretty Raw Hex
1 POST /bad%20version/storeContent.php HTTP/1.1
2 Host: localhost
3 Content-Length: 13
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="143", "Not A(Brand";v="24"
6 sec-ch-ua-mobile: ?0
7 sec-ch-ua-platform: "Windows"
8 Accept-Language: en-US,en;q=0.9
9 Origin: http://localhost
10 Content-Type: application/x-www-form-urlencoded
11 Upgrade-Insecure-Requests: 1
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36
13 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer: http://localhost/bad%20version/addcontent.php
19 Accept-Encoding: gzip, deflate, br
20 Cookie: PHPSESSID=ranqtpi5jtdphbfoln7jupndoo
21 Connection: keep-alive
22
23 thought=hello
Search
0 highlights
Inspector
Request attributes 2
Request query parameters 0
Request body parameters 1
Request cookies 1
Request headers 20
Inspector
Notes
Event log All issues
Memory: 116.7MB of 9.91GB
Disabled


Burp Project Intruder Repeater View Help
Burp Suite Community Edition v2025.11.4 - Temporary Project
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on
Forward
Drop
Request to http://localhost:80 [127.0.0.1]
Open browser
Time Type Direction Method URL Status code Length
04:38:02 18 De... HTTP Request POST http://localhost/bad%20version/storeContent.php
Request
Pretty Raw Hex
1 POST /bad%20version/storeContent.php HTTP/1.1
2 Host: localhost
3 Content-Length: 13
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="143", "Not A(Brand";v="24"
6 sec-ch-ua-mobile: ?0
7 sec-ch-ua-platform: "Windows"
8 Accept-Language: en-US,en;q=0.9
9 Origin: http://localhost
10 Content-Type: application/x-www-form-urlencoded
11 Upgrade-Insecure-Requests: 1
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36
13 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer: http://localhost/bad%20version/addcontent.php
19 Accept-Encoding: gzip, deflate, br
20 Cookie: PHPSESSID=ranqtpi5jtdphbfoln7jupndoo
21 Connection: keep-alive
22
23 thought=test'
Search
0 highlights
Inspector
Request attributes 2
Request query parameters 0
Request body parameters 1
Request cookies 1
Request headers 20
Inspector
Notes
Event log All issues
Memory: 116.7MB of 9.91GB
Disabled



The below shows that when logging in there is no proper authentication method and it would allow any account, even when they are not in the SQL database, show in the screenshots below, it then allows for the user to continue to the other parts of the application with no proper authentication. This means that any person can log in and then commit attacks with no proper way to do anything about them because they are not in the database.



Baseline Implementations Mitigation – Application 2

For the second application, the main objective was to build upon the previous website but remove all the “bad” parts of the previous applications, mitigating the vulnerabilities. In the example below the $user variable is controlled directly by the users unput, therefore allowing them to directly input SQL, resulting in malicious behaviour being possible.

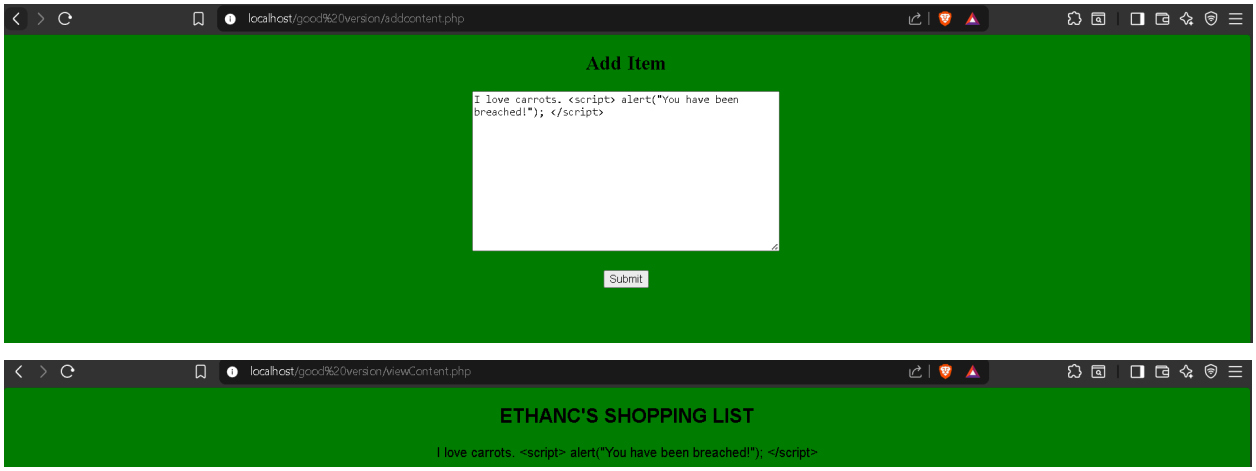
One of the first ways I mitigated was to mitigate SQL Injections and XSS. To do so database queries were rewritten using prepared statements with queries that have parameters. For XSS encoding was applied to content that user’s controlled and a CSP header was introduced.

```
header("Content-Security-Policy: default-src 'self'; style-src 'self' 'unsafe-inline'");
```

```
echo "<p>" . htmlspecialchars((string)$row['items'], ENT_QUOTES, 'UTF-8') . "</p>";
```

```php
$stmt = $conn->prepare("INSERT INTO user_items (user, items) VALUES (?, ?)");
$stmt->bind_param("ss", $user, $thought);
$stmt->execute();
```

Having prepared statements means the user input is stricter and data cannot change the SQL database. In order to test if it worked, I tried a SQL injection as shown below, and it did not work anymore.



The outcome was that the attack did not work and the database and the HTML was not affected by the attack and only the users data is visible. This proves that the mitigation was successful. Additionally the below security implementations were also used.

IDOR was the next to be mitigated, to do so user identity was session exclusive and not from the link as shown below, so now when they login to the incorrect section, it is not possible. This keeps the other users information safe and only accessible to them.

```php
if (isset($_GET['user']) && $_GET['user'] !== $_SESSION['username']) {
    http_response_code(403);
    die("Unauthorized");
}
```

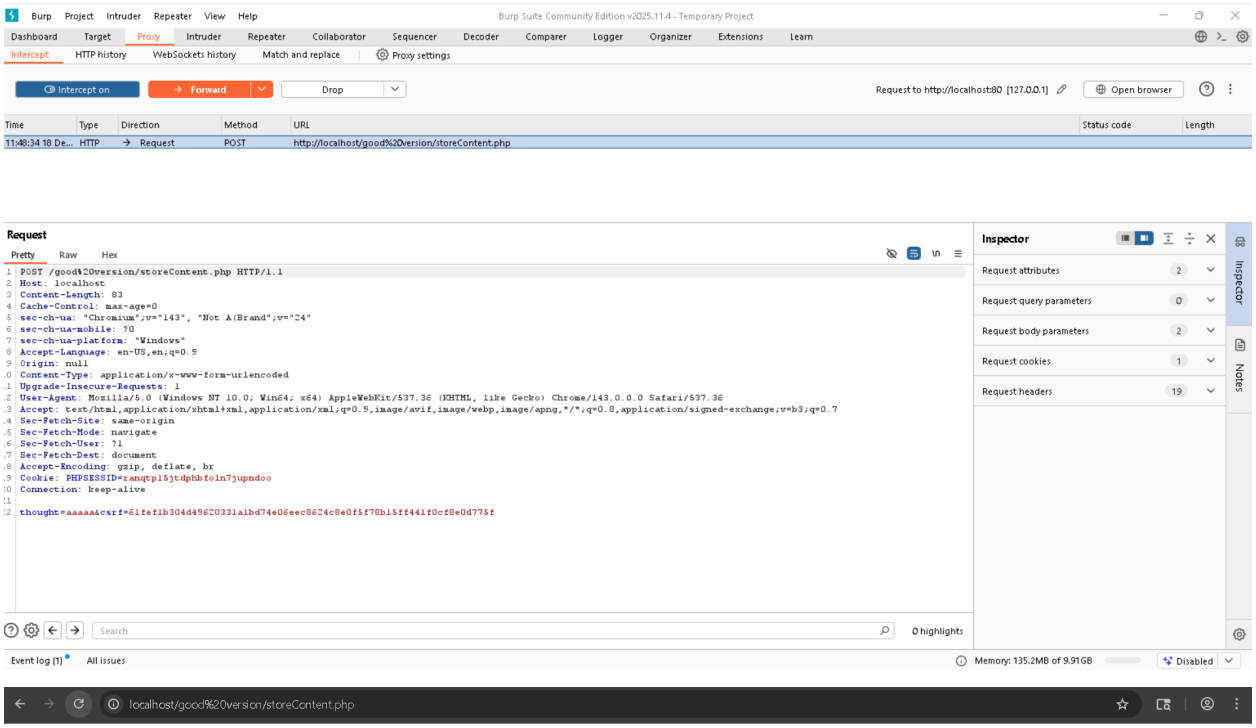
CSRF mitigation was also done. In order to do so a token is generated when a session is created as shown below.

```php
if (!isset($_SESSION['csrf'])) {
    $_SESSION['csrf'] = bin2hex(random_bytes(32));
}
```

The token is then validated when a form is submitted.

```php
if (
    !isset($_POST['csrf']) ||
    !isset($_SESSION['csrf']) ||
    !hash_equals($_SESSION['csrf'], $_POST['csrf'])
) {
    die("CSRF validation failed");
}
```

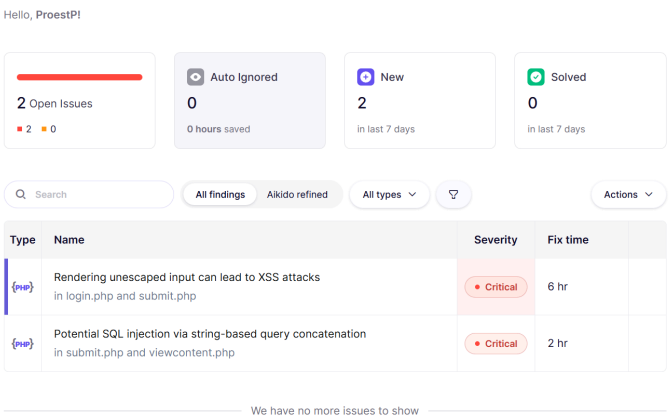
In order to test this, I used Burp Suite and edited the message. Resulting to a “CSRF validation failed” message. This means users cannot forge requests anymore and only legitimate sessions can be submitted.

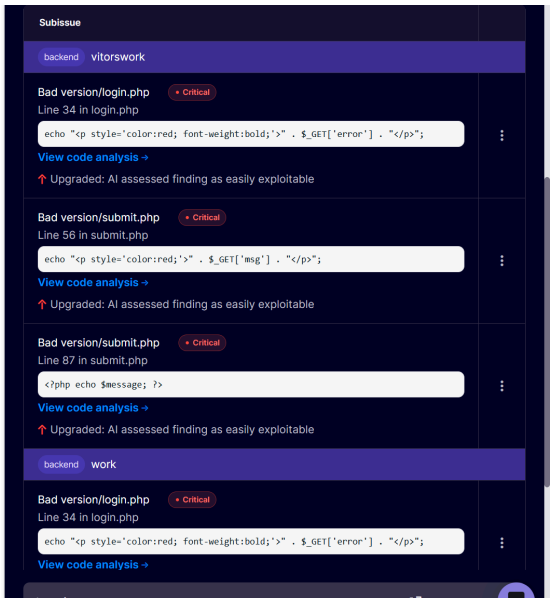


CSRF validation failed

SAST Scan

Using a SAST scan I scanned my bad version and my good version to find the differences between the two. I used aikido security for this SAST scan.





The SAST tool had several security errors labelled as “Critical” showing that the application is vulnerable to XSS attacks and SQL injections. It shows some mitigations that were made in the good version such as the PHP echo. This scan of both my good and bad versions resulted in there being 0 errors for my good version and 2 critical ones for my bad version. I would therefore conclude that my mitigations were effective. I would also believe SAST is effective as it was able to pick up on insecurities in minutes which may take a developer hours to do and would be increasingly more effective as a project expands. However SAST can’t detect every possible error, such as runtime or configuration issues, also showing that it cannot be effective everywhere and some manual checking is required.

Reflection

The applications were made using a security focused methodology that was inspired by Secure Software Development Lifecycle (SSDLC) principles that I had learned. At first, I focused on the functionality and once the functionality was done I used security tools and manual testing to find the vulnerabilites in the bad version. This approach allowed for me to be precise and clear. It also allowed for me to compare the good and bad version to see if I made an effect.

Security requirements were then introduced in the section application, which included session authentication, access control, input validation, HTTP headers and CSRF protection. Common vulnerabilities such as SQL Injections, XSS, IDOR and CSRF were made intentionally then they were mitigated, showing the understanding of both the attacking and the defensive side.

Tools such as Burp Suite and SAST were used along with manual inspection for a more in-depth analysis. Which also shows the importance of both automation and the human side of testing.

Overall this methodology reflections on real world practices, securing based on a more continuous basis rather than a one time effort through maintenance.

References

https://app.aikido.dev/ - used to SAST scan

Burp Suite – Used for vulnerabilities and testing