Lab: Remediation and Testing

Jaiden Shoel, June 2th, 2025

Introduction and Materials

For this lab, we shifted from breaking into the HusKey Password Manager to hardening it, where we essentially patched the exact vulnerabilities that my peers/classmates found earlier in the week.

We focused on remediating three vulnerabilities, three of which were classified as critical or high severity, such as unrestricted file upload, insecure cookie configurations, and session fixation vulnerabilities. Each one required careful review of the application's PHP code, session logic, and browser behavior through DevTool. While fixing them, I gained a much deeper appreciation for how layered and detail-oriented cyber security really is.

Tools Used:

- The HusKey Password Manager (running on localhost)
- Google Chrome Developer Tools
- VS Code
    - PHP files within the app (index.php, vault_details.php, login.php, etc.)

Steps to Reproduce

Vulnerability 4: Missing CSRF Token in Login Form (Reported by Julie Ganbold)

CWE Reference:

- CWE-352: Cross-Site Request Forgery (CSRF)

Severity: Medium

Steps to Reproduce (Before Patch):

1. Navigate to the HusKey Password Manager login page at https://localhost/login.php.
2. Open Chrome DevTools and inspect the HTML structure of the login form.
3. Confirm that there is no hidden <input> field named csrf_token present in the form.
4. Create a malicious HTML form on an attacker-controlled domain:

Ex.

```
<form action="https://localhost/login.php" method="post">
```

```
<input name="username" value="admin">

<input name="password" value="password123">

<input type="submit">

</form>
```

5. Submit this form. The request is processed normally despite being forged, indicating a successful CSRF attack.

6. Confirm that login.php accepts the request without verifying any CSRF token.

Screenshot: DevTools showing no CSRF token present in the form before patching.

Remediation:

To mitigate this vulnerability, I implemented full CSRF protection in login.php using the following changes:

- Token Generation

At the top of login.php, I generated a token using:

```
if (empty($_SESSION['csrf_token'])) {

  $_SESSION['csrf_token'] = bin2hex(random_bytes(32));

}
```

- Token Injection into Form

In the login form HTML, I injected the token using:

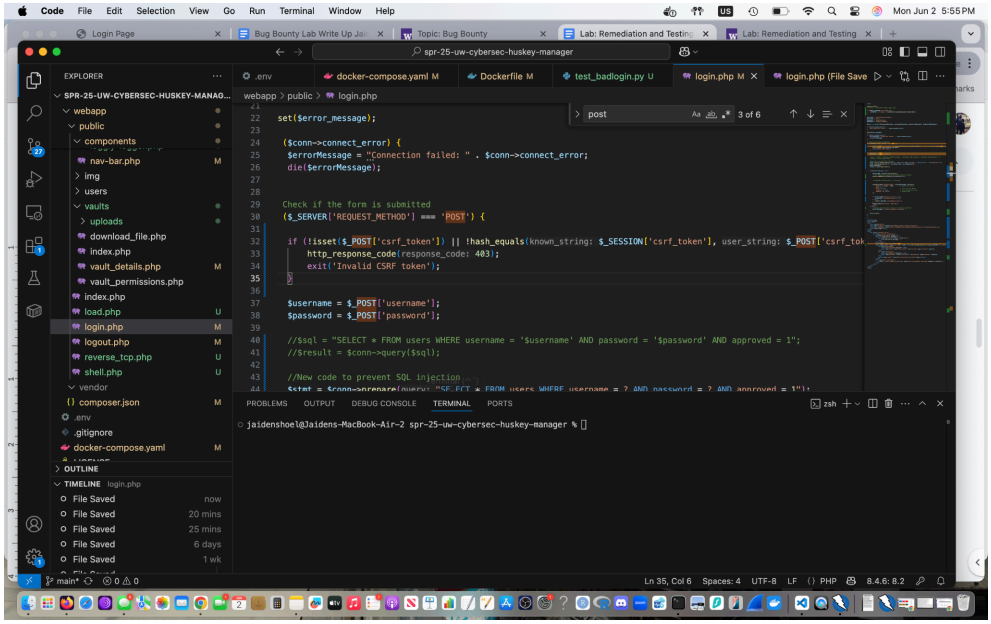
```
<input type="hidden" name="csrf_token" value="<?php echo htmlspecialchars($_SESSION['csrf_token']); ?>">
```

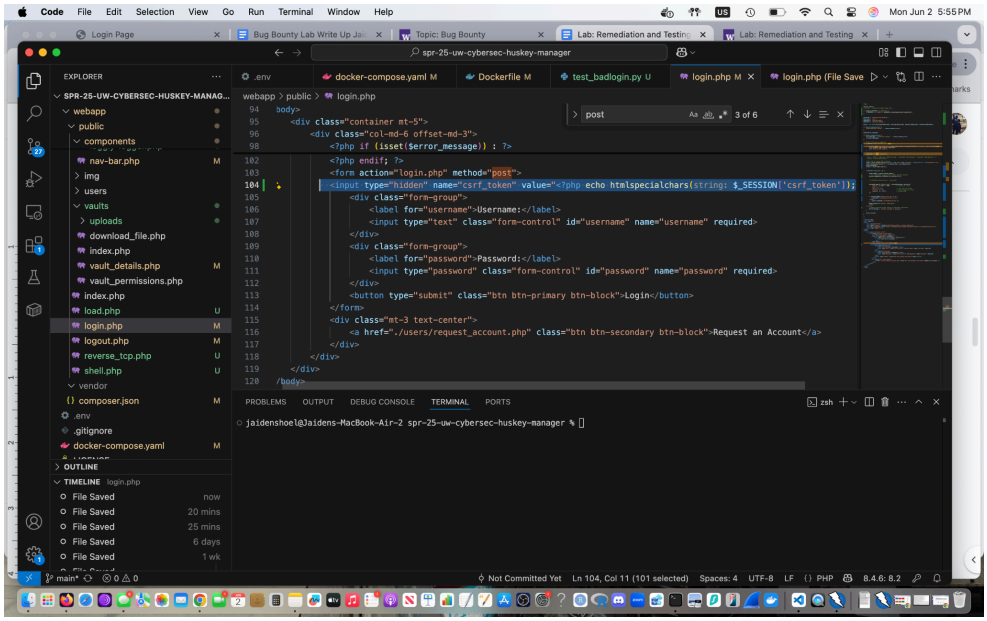
- Token Validation

At the beginning of the POST login logic, I added:

```
if (!isset($_POST['csrf_token']) || !hash_equals($_SESSION['csrf_token'], $_POST['csrf_token'])) {

    http_response_code(403);

    exit('Invalid CSRF token');

  }
```

- These steps ensure that any login attempt must come from a session that includes the correct token.



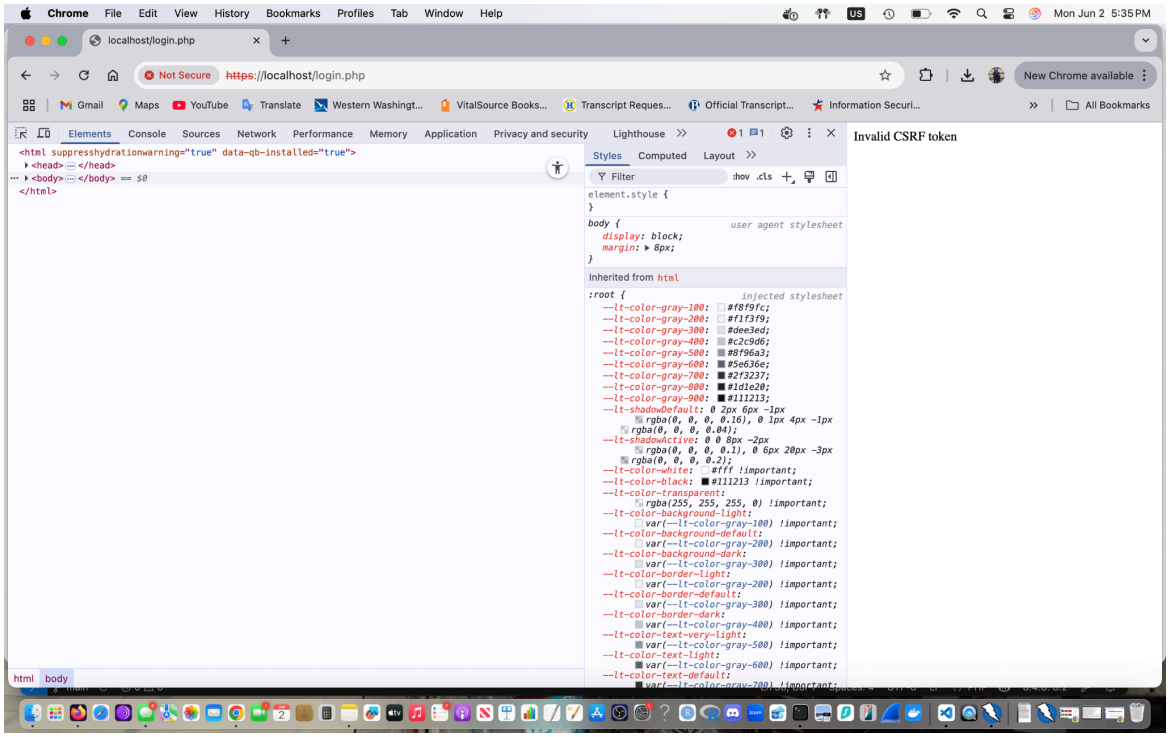


Steps to Reproduce (After Patch):

- Return to https://localhost/login.php and open DevTools to inspect the form.
- Confirm that a hidden CSRF token is present:
- <input type="hidden" name="csrf_token" value="...">
- Submit the form normally — login proceeds as expected.
- Open DevTools and delete the CSRF token field or change its value (example: to badtoken123).

- Submit the form — the server responds with the message “Invalid CSRF token,” confirming that the patch works as intended.

Screenshot:



Link to video:
https://drive.google.com/file/d/1puAzSs-DOhmVhN2IM7SxZ9WFK1mE3x8k/view?usp=sharing

Vulnerability 5: Missing X-Frame-Options Header (Reported by Me & Jeffrey Zeng)

CWE Reference:

- CWE-1021: Improper Restriction of Rendered UI Layers or Frames

Severity: Medium

Steps to Reproduce (Before Patch)

1. Open your browser and go to https://localhost/login.php.
2. Open **DevTools > Network tab**.
3. Refresh the page, and click on the login.php response in the left panel.
4. Look under the **Response Headers** section.
5. **Observe:** There is **no** X-Frame-Options header present.

6. Now simulate an attack by creating a malicious HTML file (Shown in Video):
7. Save this as clickjack_test.html and open it in Chrome.
8. While logged into the site in another tab, try clicking the fake button.
9. **Observe**: Your click is hijacked. You're unknowingly interacting with the login form behind the transparent iframe. In other words or simpler terms the fake site is on the actual site essentially.

Video: https://drive.google.com/file/d/1ybnzcRxxiv2vXNYmLw4igSnMct2_pWso/view?usp=sharing

Why This Is a Problem:

Without an X-Frame-Options, your page can be embedded in a malicious site as a hidden iframe. An attacker could:

- Trick a user into clicking a fake button while actually clicking "Delete Account" or "Transfer Funds" on your app. Again, because the fake site is on the actual site essentially
- Perform UI redressing or other clickjacking attacks.
- Compromise session integrity by exploiting trusted actions.

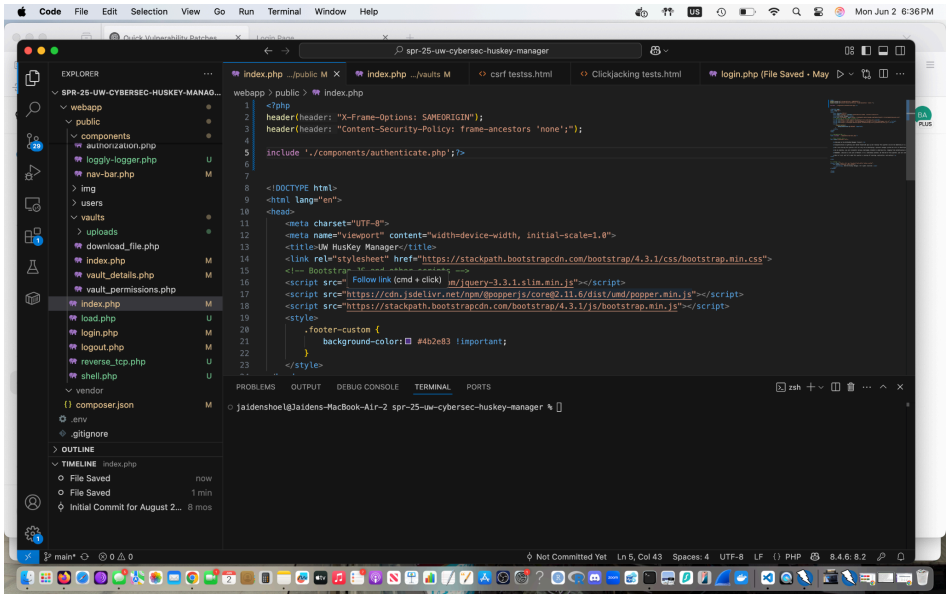
Remediation

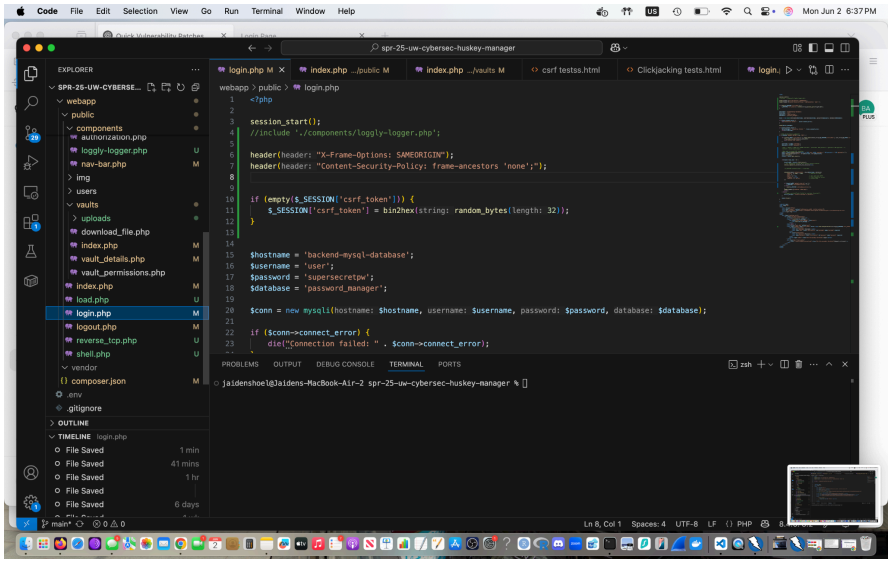
- To protect the application against clickjacking, I added two headers to every HTML-rendering PHP file:
- header("X-Frame-Options: SAMEORIGIN");
- header("Content-Security-Policy: frame-ancestors 'none';");

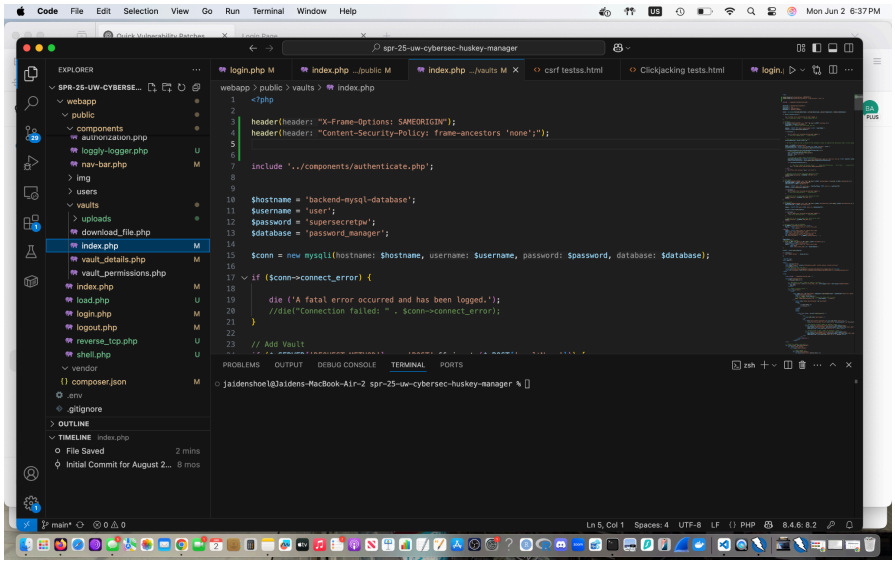
These headers were inserted at the **top of**:

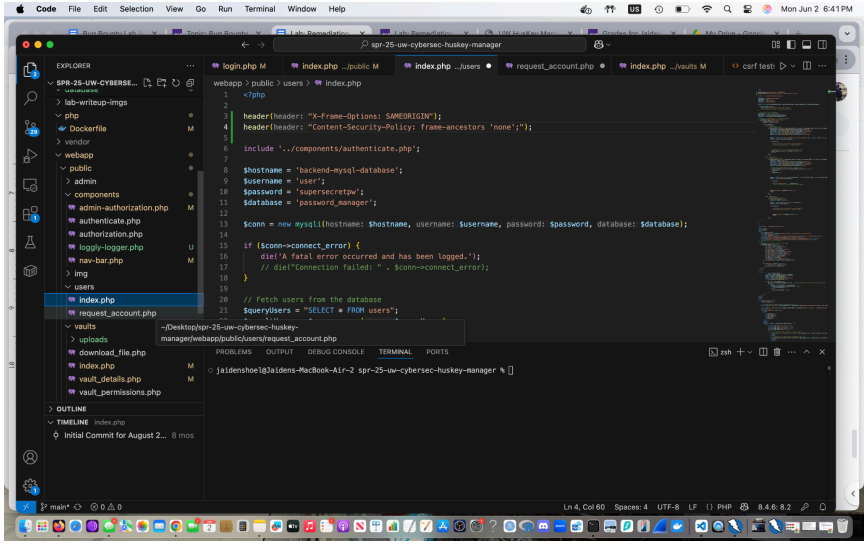
- login.php
- index.php
- vaults/index.php
- users/index.php
- users/request_account.php

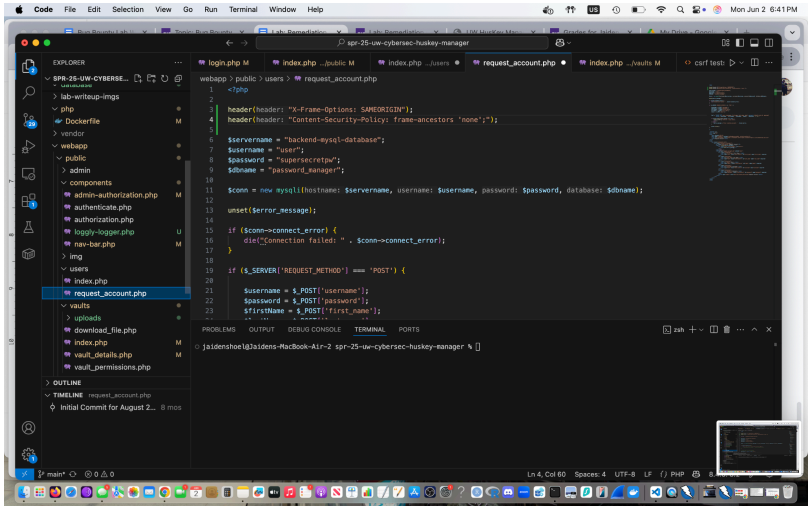
The X-Frame-Options header prevents the page from being embedded in iframes from other domains, and the stricter frame-ancestors 'none' directive blocks all embedding completely — even from the same site.


Code File Edit Selection View Go Run Terminal Window Help
Mon Jun 2 6:36 PM
spr-25-uw-cybersec-huskey-manager
EXPLORER
SPR-25-UW-CYBERSEC-HUSKEY-MANAG...
webapp
public
components
authorization.php
loggly-logger.php
nav-bar.php
img
users
vaults
uploads
download_file.php
index.php
vault_details.php
vault_permissions.php
index.php
load.php
login.php
logout.php
reverse_tcp.php
shell.php
vendor
composer.json
.env
.gitignore
OUTLINE
TIMELINE index.php
File Saved now
File Saved 1 min
Initial Commit for August 2... 8 mos
index.php .../public M
index.php .../vaults M
csrf testss.html
Clickjacking tests.html
login.php (File Saved • May
webapp > public > index.php
<?php
header(header: "X-Frame-Options: SAMEORIGIN");
header(header: "Content-Security-Policy: frame-ancestors 'none';");
include './components/authenticate.php';?>
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>UW HusKey Manager</title>
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
<!-- Bootstrap JS and jQuery scripts -->
<script src="
Follow link (cmd + click)
ap/jquery-3.3.1.slim.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>
<style>
.footer-custom {
background-color: #4b2e83 !important;
}
</style>
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
zsh
jaidenshoel@Jaidens-MacBook-Air-2 spr-25-uw-cybersec-huskey-manager %
main*
0 0
Not Committed Yet Ln 5, Col 43 Spaces: 4 UTF-8 LF PHP 8.4.6: 8.2


Code File Edit Selection View Go Run Terminal Window Help
Mon Jun 2 6:37 PM
spr-25-uw-cybersec-huskey-manager
EXPLORER
SPR-25-UW-CYBERSE...
webapp
public
components
authorization.php
loggly-logger.php
nav-bar.php
img
users
vaults
uploads
download_file.php
index.php
vault_details.php
vault_permissions.php
index.php
load.php
login.php
logout.php
reverse_tcp.php
shell.php
vendor
composer.json
.env
.gitignore
OUTLINE
TIMELINE login.php
File Saved 1 min
File Saved 41 mins
File Saved 1 hr
File Saved
File Saved 6 days
login.php M
index.php .../public M
index.php .../vaults M
csrf testss.html
Clickjacking tests.html
webapp > public > login.php
<?php
session_start();
//include './components/loggly-logger.php';
header(header: "X-Frame-Options: SAMEORIGIN");
header(header: "Content-Security-Policy: frame-ancestors 'none';");
if (empty($_SESSION['csrf_token'])) {
$_SESSION['csrf_token'] = bin2hex(string: random_bytes(length: 32));
}
$hostname = 'backend-mysql-database';
$username = 'user';
$password = 'supersecretpw';
$database = 'password_manager';
$conn = new mysqli(hostname: $hostname, username: $username, password: $password, database: $database);
if ($conn->connect_error) {
die("Connection failed: " . $conn->connect_error);
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
zsh
jaidenshoel@Jaidens-MacBook-Air-2 spr-25-uw-cybersec-huskey-manager %
main*
0 0
Ln 8, Col 1 Spaces: 4 UTF-8 LF PHP


Code File Edit Selection View Go Run Terminal Window Help
Mon Jun 2 6:37 PM
spr-25-uw-cybersec-huskey-manager
index.php
<?php
header(header: "X-Frame-Options: SAMEORIGIN");
header(header: "Content-Security-Policy: frame-ancestors 'none';");
include '../components/authenticate.php';
$hostname = 'backend-mysql-database';
$username = 'user';
$password = 'supersecretpw';
$database = 'password_manager';
$conn = new mysqli(hostname: $hostname, username: $username, password: $password, database: $database);
if ($conn->connect_error) {
die ('A fatal error occurred and has been logged.');
//die("Connection failed: " . $conn->connect_error);
}
// Add Vault
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
jaidenshoel@Jaidens-MacBook-Air-2 spr-25-uw-cybersec-huskey-manager %


Code File Edit Selection View Go Run Terminal Window Help
Mon Jun 2 6:41 PM
spr-25-uw-cybersec-huskey-manager
index.php
request_account.php
<?php
header(header: "X-Frame-Options: SAMEORIGIN");
header(header: "Content-Security-Policy: frame-ancestors 'none';");
include '../components/authenticate.php';
$hostname = 'backend-mysql-database';
$username = 'user';
$password = 'supersecretpw';
$database = 'password_manager';
$conn = new mysqli(hostname: $hostname, username: $username, password: $password, database: $database);
if ($conn->connect_error) {
die('A fatal error occurred and has been logged.');
// die("Connection failed: " . $conn->connect_error);
}
// Fetch users from the database
$queryUsers = "SELECT * FROM users";
~/Desktop/spr-25-uw-cybersec-huskey-manager/webapp/public/users/request_account.php
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
jaidenshoel@Jaidens-MacBook-Air-2 spr-25-uw-cybersec-huskey-manager %

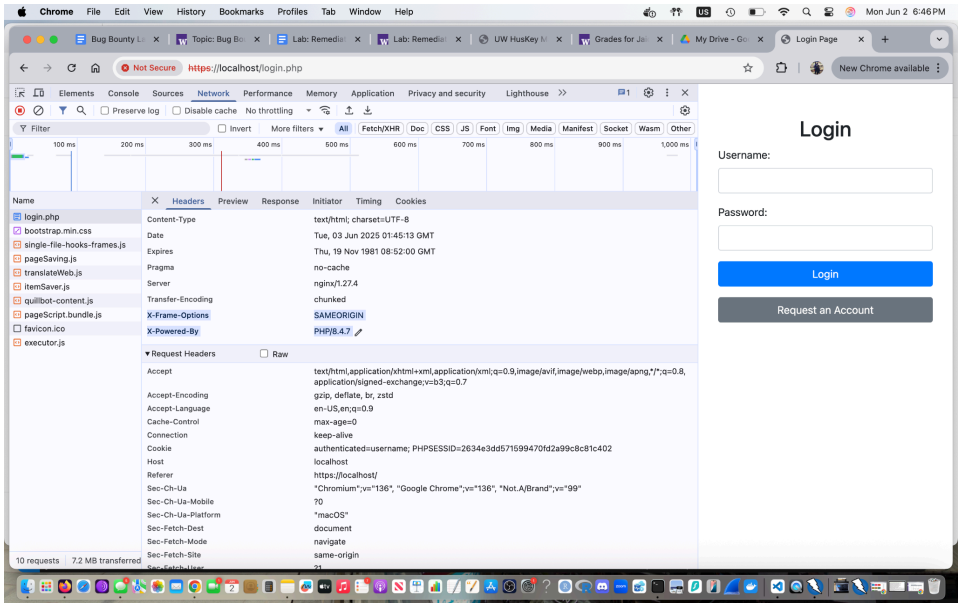

Code File Edit Selection View Go Run Terminal Window Help
Mon Jun 2 6:41 PM
spr-25-uw-cybersec-huskey-manager
request_account.php
<?php
header(header: "X-Frame-Options: SAMEORIGIN");
header(header: "Content-Security-Policy: frame-ancestors 'none';");
$servername = "backend-mysql-database";
$username = "user";
$password = "supersecretpw";
$dbname = "password_manager";
$conn = new mysqli(hostname: $servername, username: $username, password: $password, database: $dbname);
unset($error_message);
if ($conn->connect_error) {
die("Connection failed: " . $conn->connect_error);
}
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
$username = $_POST['username'];
$password = $_POST['password'];
$firstName = $_POST['first_name'];
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
jaidenshoel@Jaidens-MacBook-Air-2 spr-25-uw-cybersec-huskey-manager %

Steps to Reproduce (After Patch)

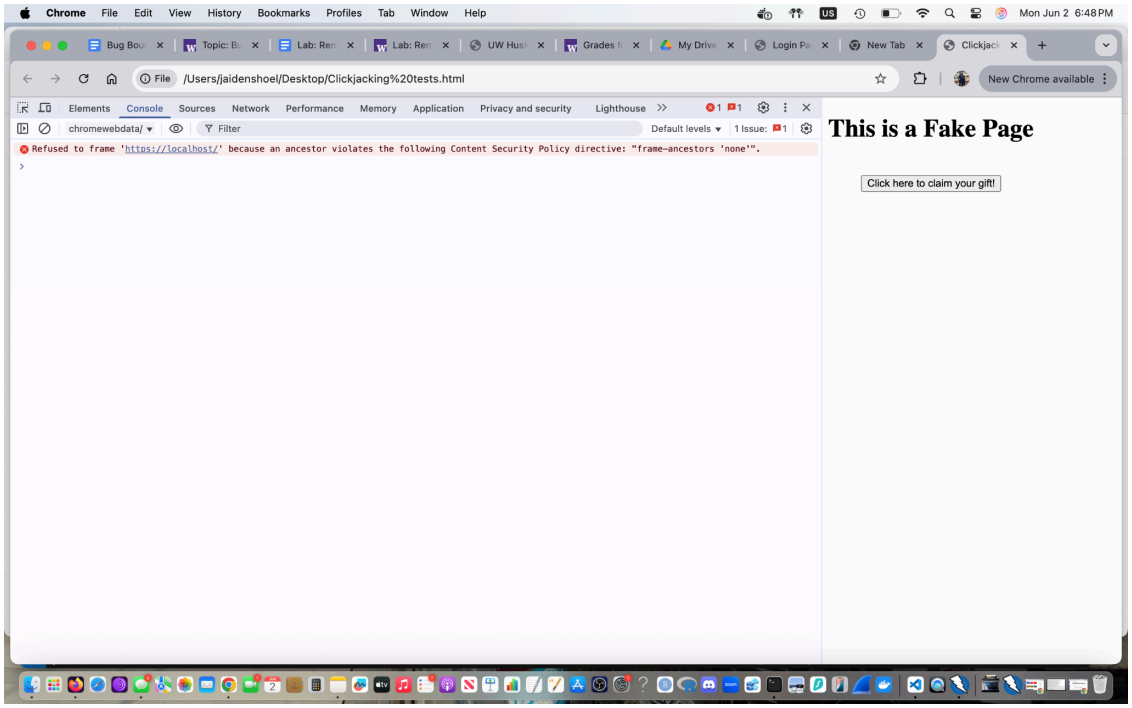
1. Open DevTools, go to Network tab on https://localhost/login.php or index.php.
2. Check the **Response Headers**.
3. Confirm that the following headers are now present:

X-Frame-Options: SAMEORIGIN

Content-Security-Policy: frame-ancestors 'none'



4. Reopen your clickjack_test.html file in Chrome.
5. Observe: The iframe **fails to load**, or is blocked entirely by the browser.
6. This confirms that **clickjacking is no longer possible**.



Class Principles

How did you prioritize the selection of your test cases?

I prioritized my test cases based on critical login functionality and security requirements directly tied to the vulnerabilities we patched in earlier labs. I started by testing the baseline behavior with valid (test_goodlogin) and invalid (test_badlogin) login credentials to ensure that authentication was working as intended. Essentially the tests Wyatt walked us through in Lab last week. From there, I prioritized testing for tampering via CSRF, as this directly impacted the security of user sessions and was identified as a previously unpatched vulnerability. Since CSRF was introduced as part of a new remediation in terms of my vulnerability pataches, confirming its effectiveness and test was essential before considering the application hardened. So even though this took me hours upon hours to create and debug. I ended up completing it and making it work successfully in the end.

Cybersecurity Triad Protections:

1. **test_goodlogin.py**
    - **Triad Property Protected:** *Availability*
    - **Explanation:** This test confirms that users with valid credentials are able to access the system without issue. It ensures that authentication is functioning correctly and the system is available for authorized use.

2. **test_badlogin.py**

    - **Triad Property Protected:** *Confidentiality*

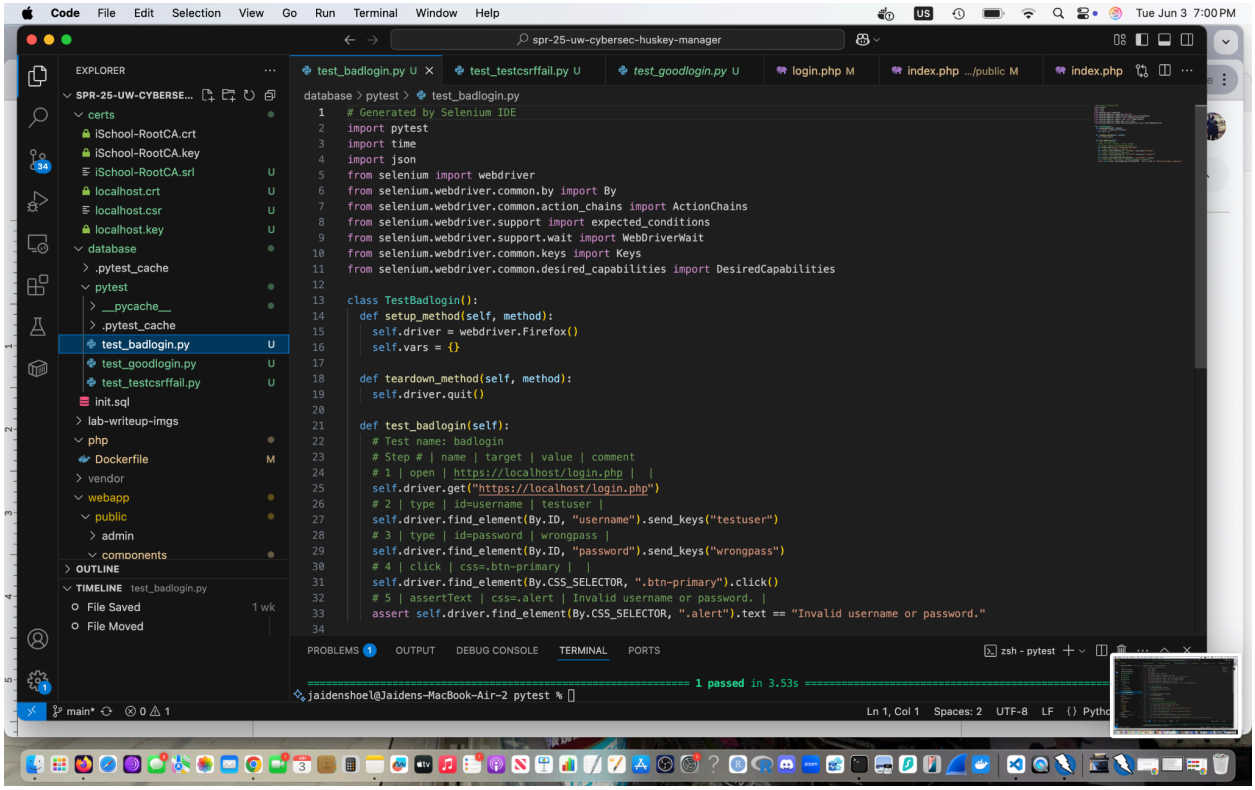
    - **Explanation:** By testing for failed login attempts with invalid credentials, this test ensures that unauthorized users cannot gain access to the system. It protects the confidentiality of user data by confirming the system does not leak or permit access without proper authentication.

3. **test_csrf_fail.py**

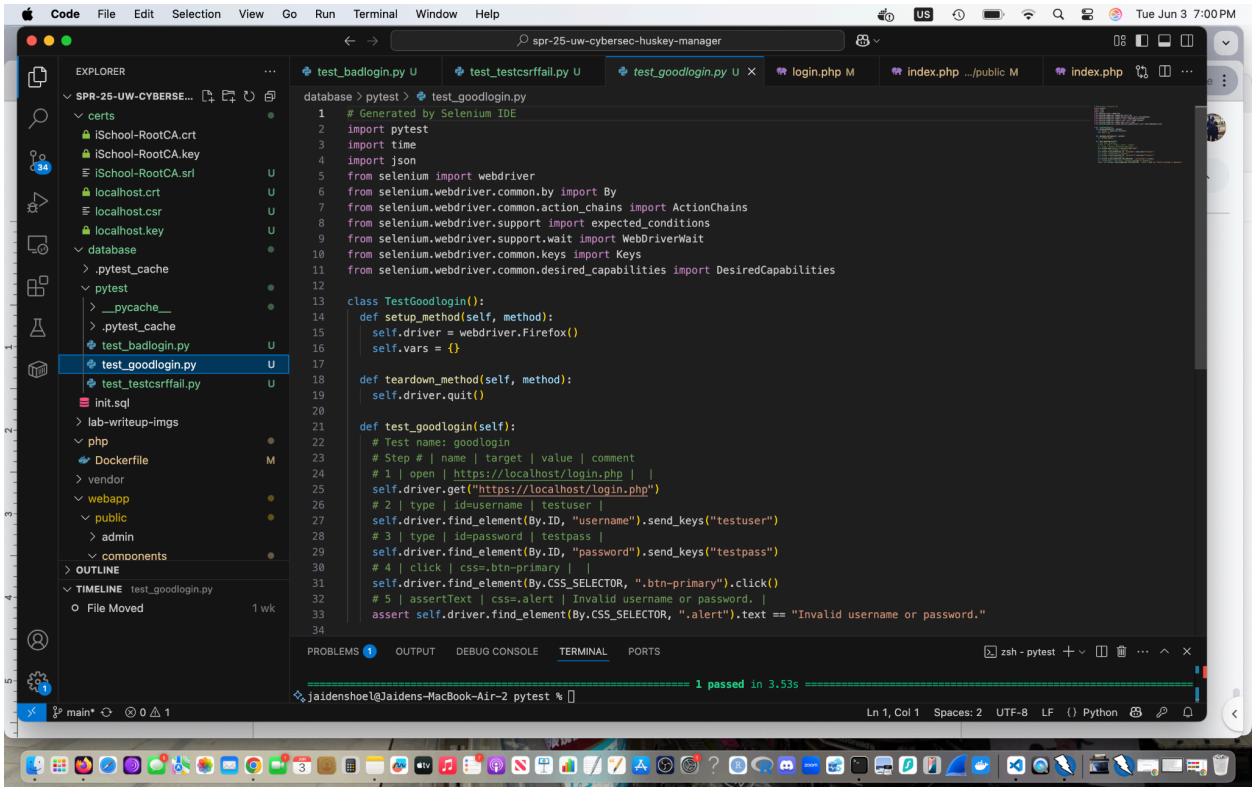
    - **Triad Property Protected:** *Integrity*

    - **Explanation:** This test verifies that a forged or tampered CSRF token results in a failed login attempt. It ensures that requests cannot be maliciously injected or modified, preserving the integrity of session-based form submissions and protecting against unauthorized actions.

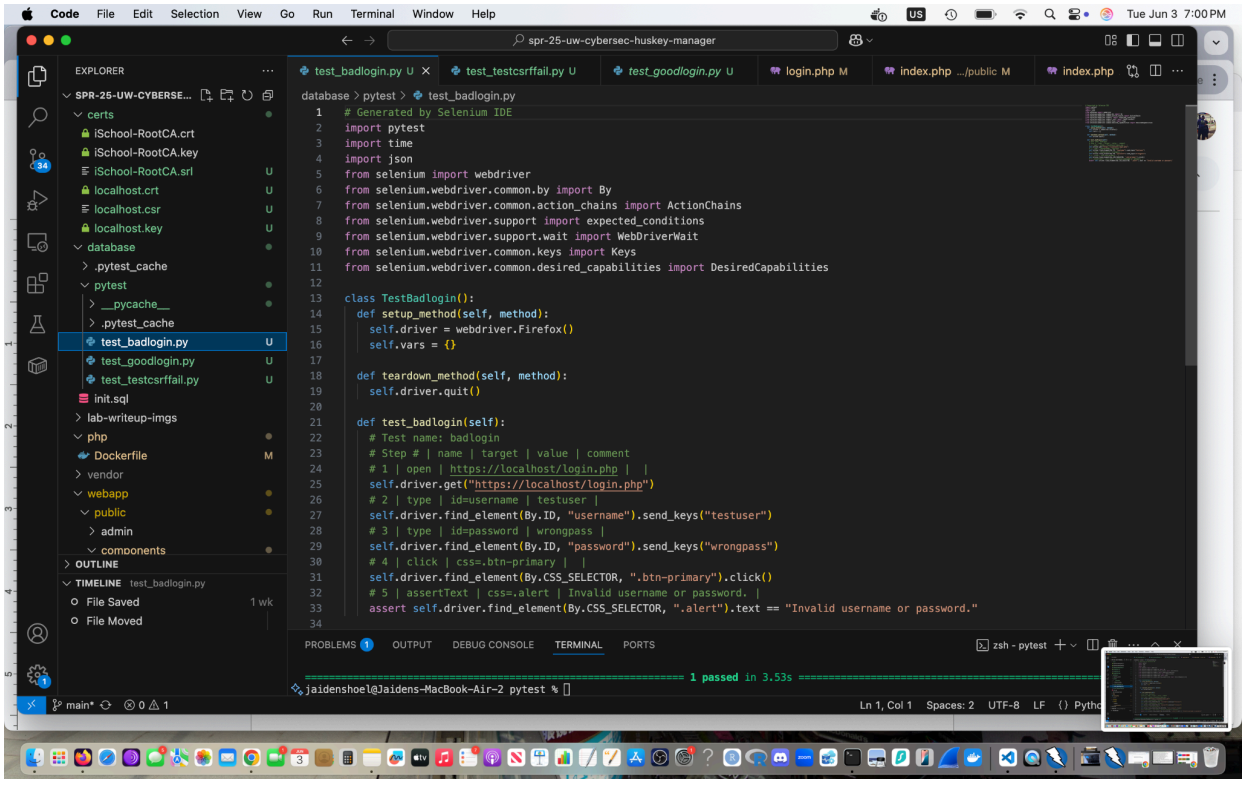
First Screenshot: Shows all the tests passed in the terminal.



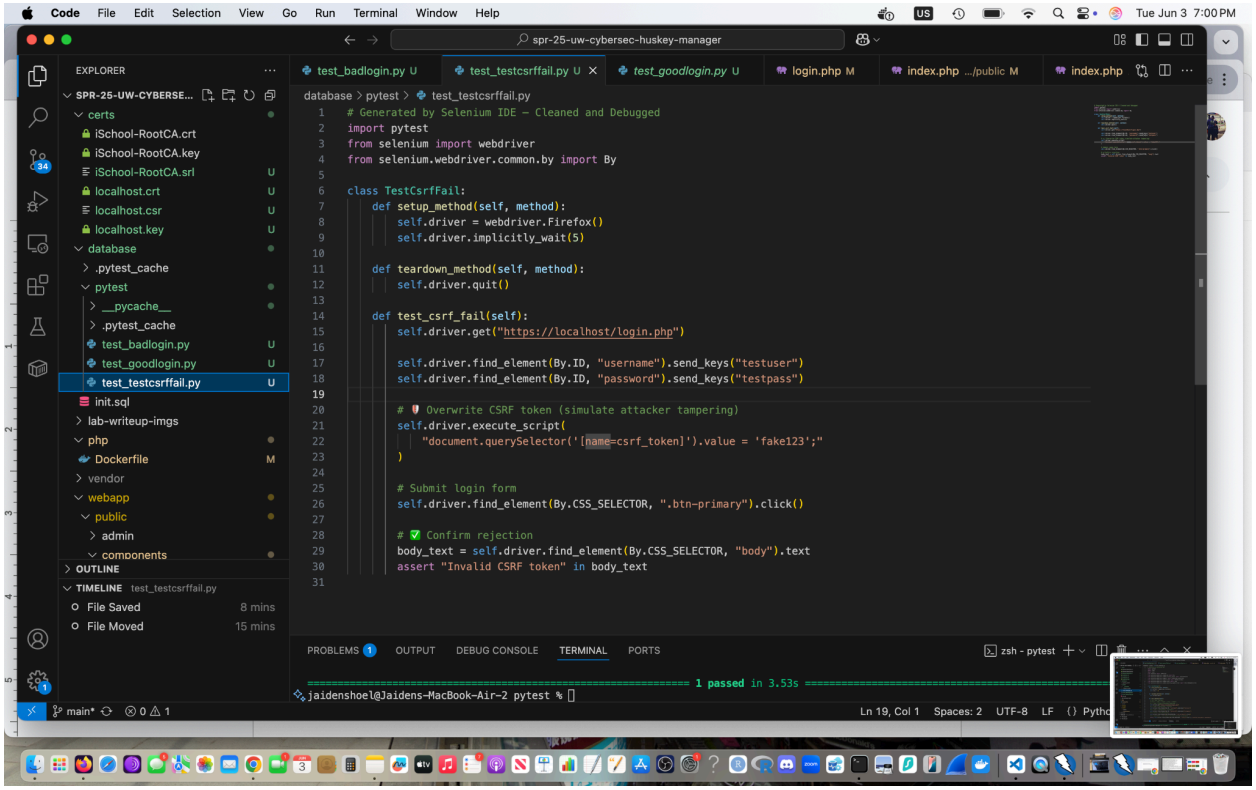
Second Screenshot: My goodlogin test:



Third Screenshot: My badlogin test:



Fourth Screenshot: My CSRFfail test: Again, test checks whether the login form correctly rejects a request with a tampered CSRF token, proving that CSRF protection is working as intended.



Hacker Mindset and Conclusion

Which of the identified issues do you believe are likely unaddressed by your peers' vault manager?

After taking a look at the entire Bug Bounty discussion once again, one vulnerability that remains unpatched in many of my peers' vault managers would be brute-force login protection. To start, while a lot of groups, including mine, focused on CSRF, session management, and file upload restrictions, very few seemed to mention anything about rate limiting or something like account lockout thresholds. All of this means that an attacker could still hammer the login form with repeated guesses, potentially using credential stuffing techniques with leaked passwords from past breaches.

Although, this vulnerability makes sense to not think about at first because it's not immediately obvious. It's one of those vulnerabilities that is so obvious but because it's so obvious it's invisible. I only thought about this by looking at my localhost login page one more time and thinking what would a hacker do aka tapping into the Hacker mindset. Then I realized I could just keep trying to guess login credentials because there isn't anything on our localhost that prevents this such as built-in rate limiting or CAPTCHA system. I bet even a basic script from Python, Java, etc. could test hundreds of passwords per minute until it gets lucky, especially with the fact that our credentials is literally just "username" and "password!"

What do you believe would be the most effective way for you to hack your way into one of your peers' vault manager, i.e. how would you exploit the vulnerabilities you previously stated are likely to still exist?

The most simple way to exploit this vulnerability that I previously stated (brute-force login) would be to literally do what I said in the last sentence. That is, creating a basic Python script using the requests library that just loops through a list of common passwords and tries them one by one against the login form. I most likely wouldn't even need to touch Selenium or anything fancy, just a loop that connects to /login.php with slightly different credentials each time.

The key thing here is that our localhost site does not have any lockout threshold or delay between attempts and the server ultimately gives me unlimited chances. I could run this script for minutes or hours if needed, and eventually I'd hit on the right combination. It's even more effective if I already suspect the username and password and in our case, again, that's not hard to guess.

What makes this attack dangerous isn't that it's technically advanced. It's literally something that is low-effort and high-reward. Anyone with basic scripting skills could pull it off or even using AI could pull off, and the only thing stopping them would be some kind of throttle or CAPTCHA, which almost none of us implemented. I think this is a perfect example of how security isn't just about patching flashy bugs but also making sure everything, especially the obvious things are patched and secured.

How would you go about detecting if one of your peers was attempting to run a similar exploit against your vault manager?

In terms of **detecting** this exploit/vulnerability, I would rely heavily on logging, especially since our vault manager currently doesn't implement rate limiting or CAPTCHA protections. If we can't actively prevent brute-force attempts at the application level, then we at least need to be able to spot them when they happen.

One concrete way to do this would maybe be by setting up detailed logs that capture each login attempt, including the timestamp, IP address, username used, and whether the attempt succeeded or failed. If I noticed a high volume of failed attempts coming from the same IP address in a short period of time, say, 50 failed logins within a minute, that would be a huge red flag. Even a pattern like 10 failed attempts for the same username with slight variations in password could suggest a credential stuffing attack. Additionally, I could forward logs to a centralized logging service like Loggly and use alerting rules to notify me by email or Slack if someone's hammering and trying to bruce forcing the login page. Ideally, this would give me a chance to block the IP address manually or put temporary measures in place before any real damage is done. However, even though logging gives us a solid second line of defense and a way to spot and respond to malicious behavior if it slips past the front gate, my first instinct for addressing this vulnerability would still be to implement actual rate limiting and CAPTCHA protections.