Roberto Reade

CSEC 380

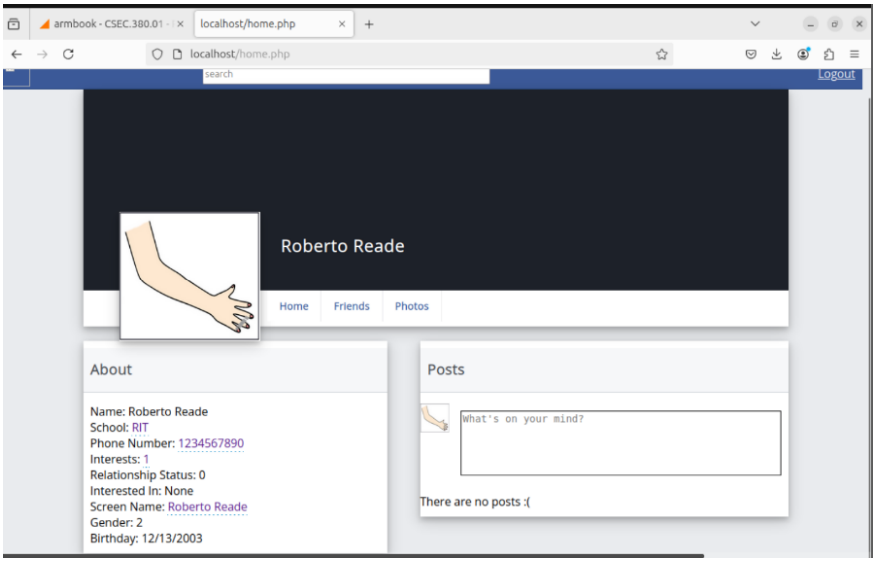
Practical Final

Section 1: Documenting Armbook Setup

To set up the Arm Book web application, I first installed the necessary components on my Ubuntu VM. This included Apache, PHP, MySQL, and PHP-MySQL. Once these were installed, I went into MySQL and created a new database named "armbook". I also made a user account to manage the database. I assigned privileges to this user to allow full access to the armbook database. After this, I downloaded and extracted the Arm Book source code and the SQL file. I used the SQL file to create the database with the tables and data that are needed for the application to work. I then copied the Arm Book files into the Apache web server's root directory. Next, I edited the application's configuration file to include the database credentials I created earlier. I did this to ensure the web application could connect to the database. Finally, I restarted Apache and accessed the Arm Book application through a web browser. I made sure the web application was set up correctly by creating a new user account through the web application's interface.

Screenshot creating new user:

Screenshot new user logged in:



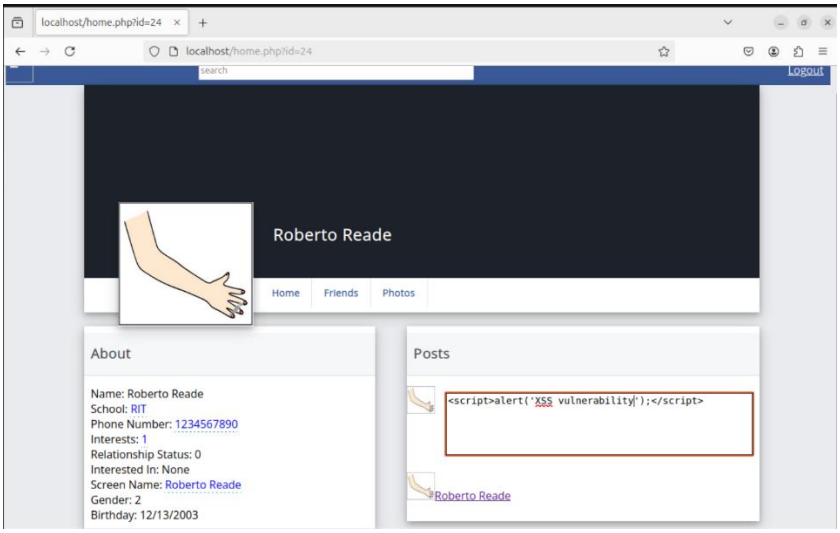
Screenshot showing user is in database:

```
practical-lab@practical-lab:~$ sudo mysql -u armbookadm -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 60
Server version: 8.0.41-0ubuntu0.24.04.1 (Ubuntu)

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE armbook;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SELECT * FROM users;
+---------+--------------------------+----------+-----------+-----------+-----+----------------+--------------+---------------+
| user_id | email                    | password | firstname | lastname  | sex | birthday_month | birthday_day | birthday_year |
+---------+--------------------------+----------+-----------+-----------+-----+----------------+--------------+---------------+
|       1 | ces1509@rit.edu          | password | Jon       | Doe       |   1 |             10 |            2 |          1988 |
|       2 | neil@neil.com            | apple    | password  | zimmerman |   1 |             10 |            2 |          1988 |
|       6 | csanders@sparsa.org      | password | Grant     | Batchlor  |   2 |             -1 |           -1 |            -1 |
|       4 | chaim.sanders@gmail.com  | password | Jon       | Mccall    |   2 |             -1 |           -1 |            -1 |
|       5 | jruppal@gmail.com        | password | Jacob     | Ruppal    |   2 |             -1 |           -1 |            -1 |
|       7 | griffith.chaffee@gmail.com | password | griffith  | chaffee   |   2 |              1 |            2 |          1982 |
|       8 | andy@culler.com          | password$ | Andy      | Culler    |   2 |             -1 |           -1 |            -1 |
|       9 | test@test.com            | password | test      | test      |   2 |             -1 |           -1 |            -1 |
|      10 | bsmith@gmail.com         | password | Bob       | Smith     |   1 |              1 |            1 |          1950 |
|      11 | test2@test.com           | password | test2     | test2     |   2 |             -1 |           -1 |            -1 |
|      13 | jrr@foobar.com           | password | Jamie     | Richard   |   2 |              3 |           28 |          1987 |
|      14 | rbower@sparsa.org        | password | Rusty     | Bower     |   2 |              1 |            1 |          1950 |
|      12 | chaim@chaim.com          | password | chaim     | sanders   |   1 |              3 |            2 |          1988 |
|      18 | jon@jon.com              | password | Jon       | Jon       |   1 |              1 |            1 |          2011 |
|      24 | rar9027@rit.edu          | rar9027  | Roberto   | Reade     |   2 |             12 |           13 |          2003 |
+---------+--------------------------+----------+-----------+-----------+-----+----------------+--------------+---------------+
```

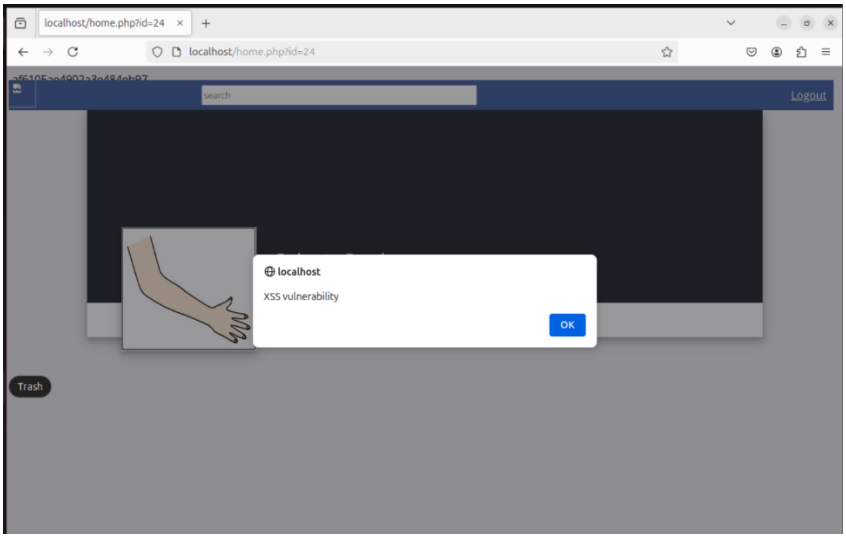
Section 2: Web Application Penetration Testing and Vulnerability Assessment

Cross-site Scripting Vulnerability		
Risk: Medium	**Impact**: Medium	**Exploitability**: High
CVSS SCORE: 6.1	**CVSS String:** CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:L/A:N	
OWASP Top 10: A03:2021 - Injection		
Component(s): timeline.php and home.php		
Impact: A user can inject a script into the timeline. This can be executed by anyone who views the post. This vulnerability can cause session hijacking or credentials / other information to be stolen.		
Description: The timeline allows users to make posts. These posts are not sanitized. This can allow an attacker to insert scripts that run in the users browser.		
Steps to Replicate: 1. Log into Arm Book 2. Go to the timeline / posts section 3. Post a malicious script (<script>alert('XSS vulnerability')</script>) 4. Refresh the page or log in again		
Solution: To mitigate this vulnerability you can use htmlspecialchars() which would sanitize input before executing it. You can also implement Content Security Policy headers which would prevent running embedded scripts.		

Creating the post to test the Cross-site Scripting vulnerability:



Proof the Cross-site Scripting vulnerability works:

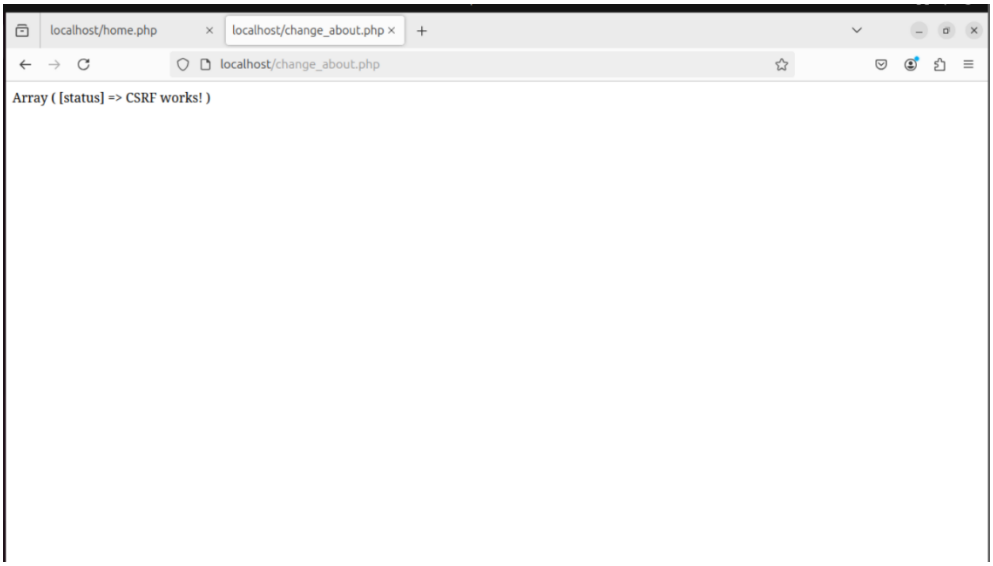


Cross-site Request Forgery		
Risk: High	**Impact**: High	**Exploitability**: Medium
CVSS SCORE: 8.8	**CVSS String:** CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:N	
OWASP Top 10: A01:2021 - Broken Access Control		
Component(s): index.php, home.php, and change_about.php		
Impact: An attacker can trick a real user into performing actions such as changing account details.		
Description: There is no CSRF token on the home page. Posting or interacting with the web application can be done from another site.		
Steps to Replicate: 1. Create a HTML file on a different server 2. Log into Arm Book 3. While logged into Arm Book, visit the HTML file in a different window		
Solution: In order to mitigate this vulnerability, you can use anti-CSRF tokens for all requests. You can also validate the HTTP Origin and Referrer headers. Another way to mitigate this would be to implement SameSite cookies.		

Here is the code for my malicious HTML file:

```
 GNU nano 7.2                    csrf_attack.html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>CSRF Attack</title>
</head>
<body>
  <form id="csrfForm" action="http://localhost/change_about.php" method="POST">
    <input type="hidden" name="status" value="CSRF works!">
  </form>
  <script>
    window.onload = function() {
      document.getElementById('csrfForm').submit();
    };
  </script>
</body>
</html>
```

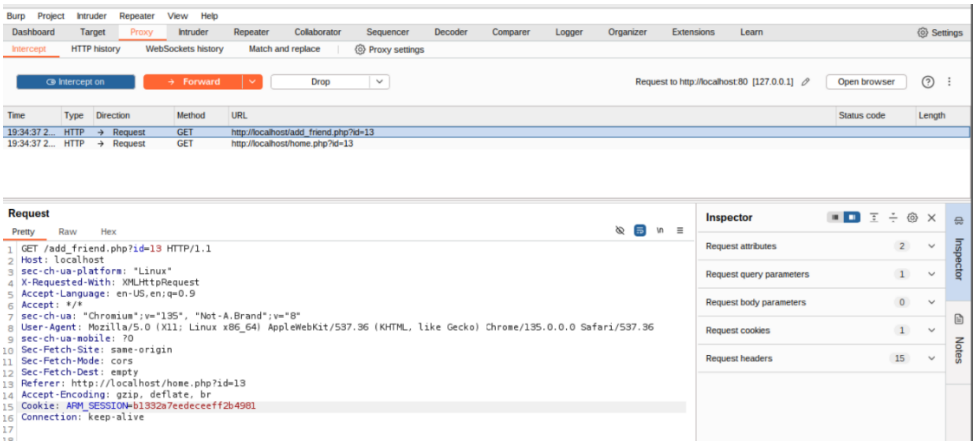
Here is me looking up the HTML file in a different window showing it works:



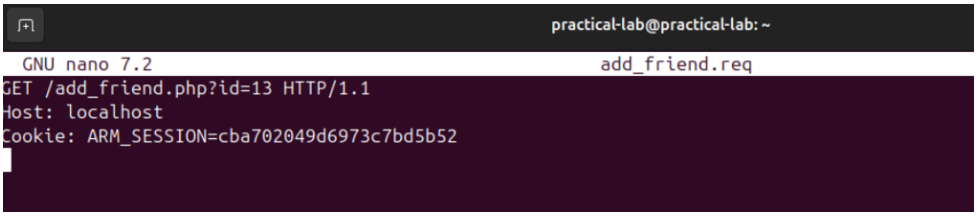
SQL Injection

Risk: High	**Impact**: High	**Exploitability**: High
CVSS SCORE: 9.1	**CVSS String:** CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:L	
OWASP Top 10: A03:2021 – Injection		
Component(s): add_friend.php		
Impact: This vulnerability allows an attacker to extract sensitive information from the armbook database. This can result in the Arm Book database being fully compromised.		
Description: The cookie parameter in the add_friend.php page is vulnerable. The application does not properly sanitize or validate user input. This can allow attackers to inject SQL code into a request and forward it along. This can lead to an attacker extracting content of the database.		
Steps to Replicate: 1. Login to Arm Book 2. Capture the GET message in Burp Suite once add friend button is clicked 3. Copy GET message into a file 4. Run sqlmap with file from above step to find vulnerability 5. Sqlmap should provide a payload to inject into message 6. Inject payload and forward message 7. You can also mess around and extract information from the armbook database by trying different sqlmap commands still using the same file from step 3		
Solution: In order to mitigate this vulnerability, Arm Book should use prepared statements. Arm Book should also implement sanitization of input to ensure a payload has not been injected.		

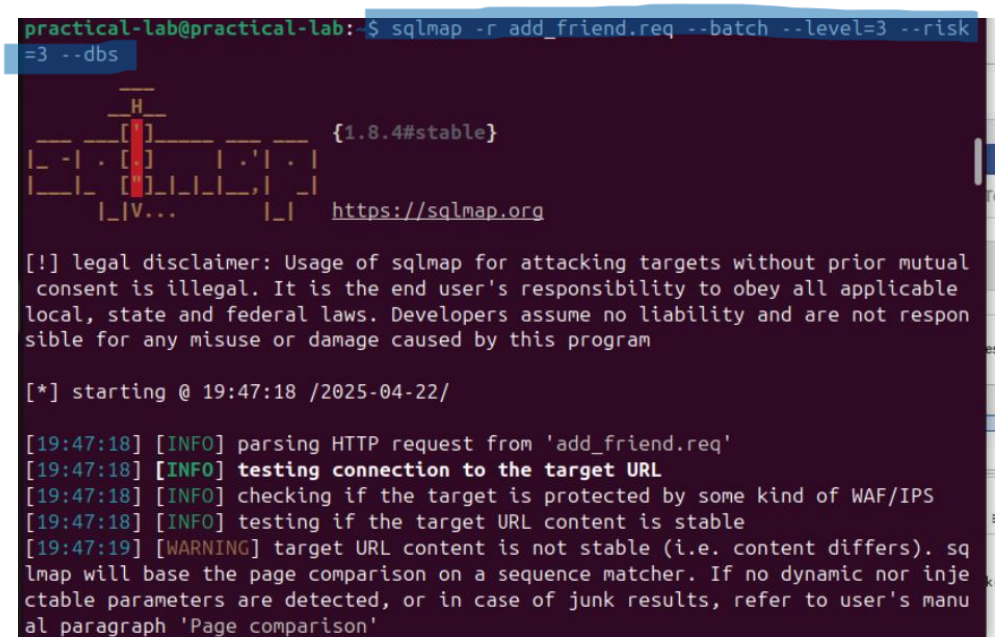
Message captured in Burp Suite:



Contents of GET message in a file used for sqlmap:



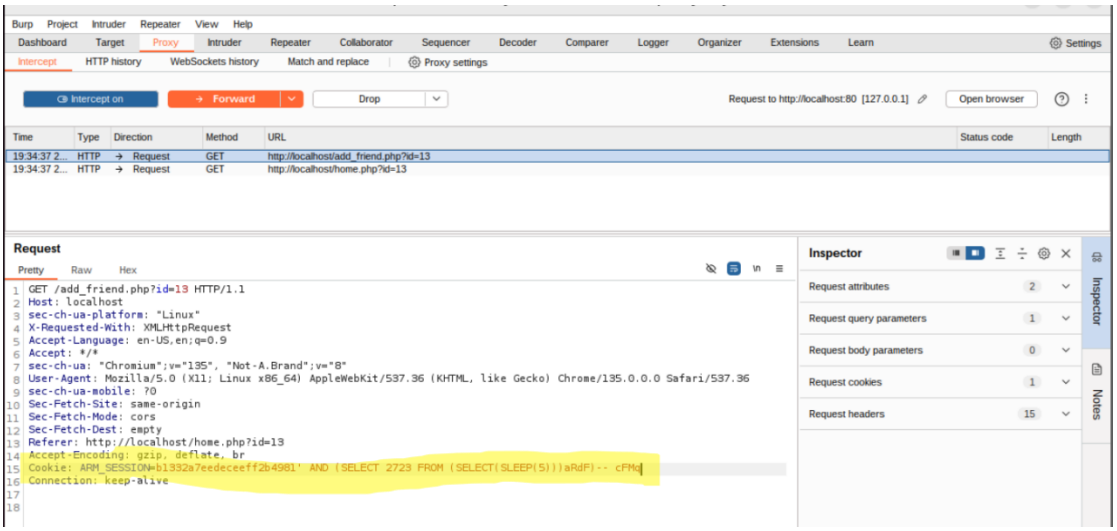
Running sqlmap:



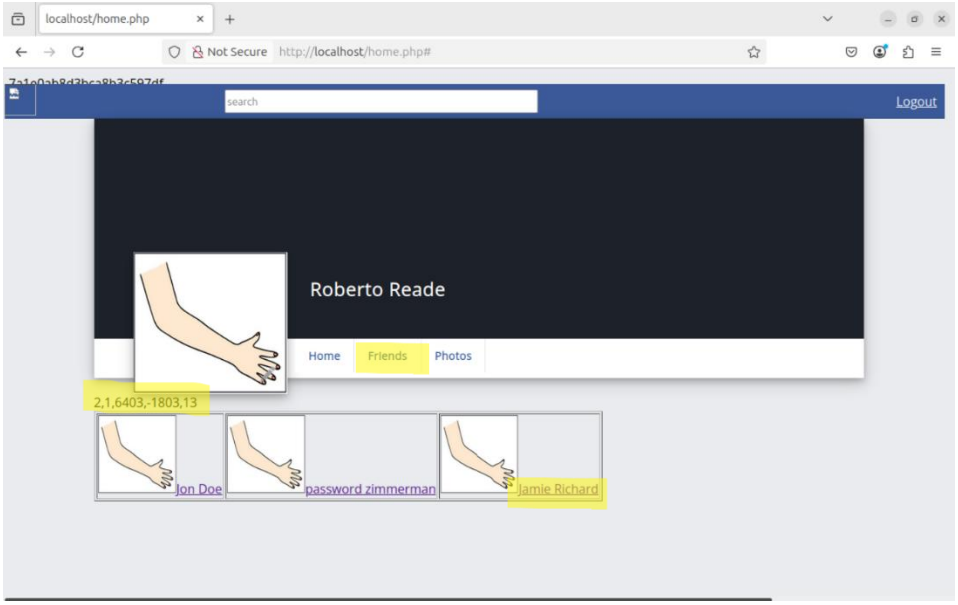
Results and payload from sqlmap:

```
for the remaining tests, do you want to include all tests for 'MySQL' extending
provided level (3) value? [Y/n] Y
[19:48:20] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[19:48:20] [INFO] automatically extending ranges for UNION query injection techn
ique tests as there is at least one other (potential) technique found
[19:48:20] [INFO] testing 'Generic UNION query (random number) - 1 to 20 columns
'
[19:48:20] [INFO] testing 'Generic UNION query (NULL) - 21 to 40 columns'
[19:48:20] [INFO] testing 'Generic UNION query (random number) - 21 to 40 column
s'
[19:48:20] [INFO] testing 'Generic UNION query (NULL) - 41 to 60 columns'
[19:48:20] [INFO] checking if the injection point on Cookie parameter 'ARM_SESSI
ON' is a false positive
Cookie parameter 'ARM_SESSION' is vulnerable. Do you want to keep testing the ot
hers (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 4244 HTTP(s)
requests:
---
Parameter: ARM_SESSION (Cookie)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: ARM_SESSION=cba702049d6973c7bd5b52' AND (SELECT 2723 FROM (SELECT(S
LEEP(5)))aRdF)-- cFMq
```

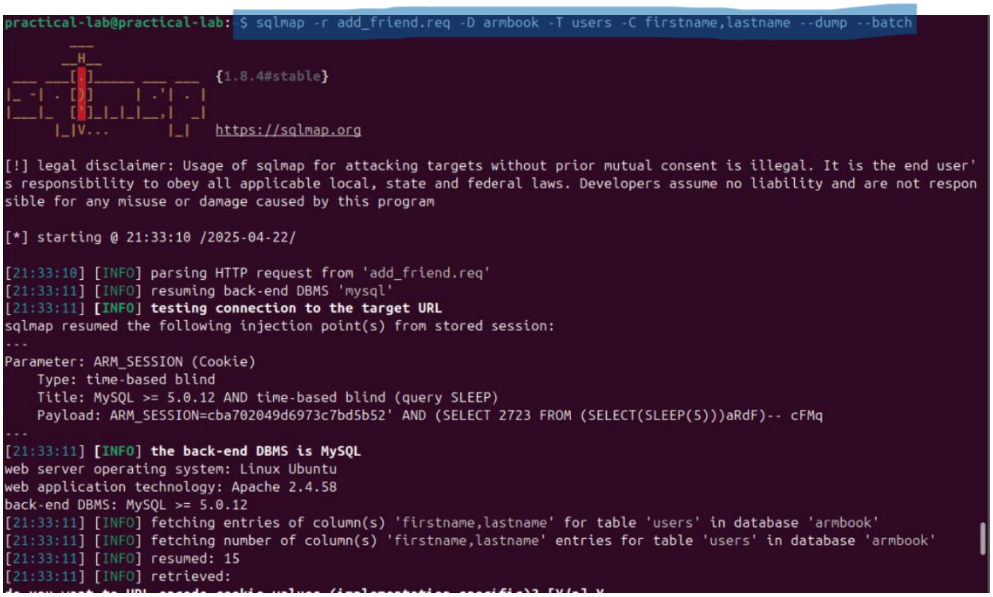
Adding exploit to captured GET message before forwarding it:



What I saw after the vulnerability was exploited in Arm Book:

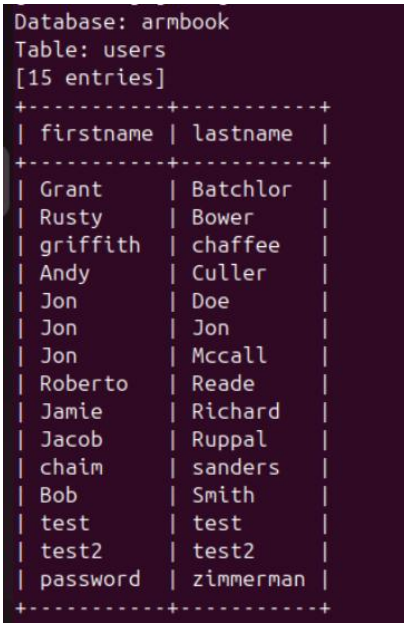


Command used in sqlmap to get information from the armbook database:



There are many other commands that can get all information from the armbook database. This is one of many that I tested out.

Results of the above command (information from armbook database gathered by sqlmap):



```
Database: armbook
Table: users
[15 entries]
+-----------+-----------+
| firstname | lastname  |
+-----------+-----------+
| Grant     | Batchlor  |
| Rusty     | Bower     |
| griffith  | chaffee   |
| Andy      | Culler    |
| Jon       | Doe       |
| Jon       | Jon       |
| Jon       | Mccall    |
| Roberto   | Reade     |
| Jamie     | Richard   |
| Jacob     | Ruppal    |
| chaim     | sanders   |
| Bob       | Smith     |
| test      | test      |
| test2     | test2     |
| password  | zimmerman |
+-----------+-----------+
```

Section 3: Automated Web Application Vulnerability Scanning

Question 1: In your opinion, which OWASP ZAP finding is the most significant and why?

In my opinion, the most significant OWASP ZAP finding was the Reflected Cross-site Scripting. It was the only alert marked as a high risk, and it had medium confidence. I also believe this is a significant finding because this can allow an attacker to hijack a user session, steal a user's credentials, or can lead to other attacks.

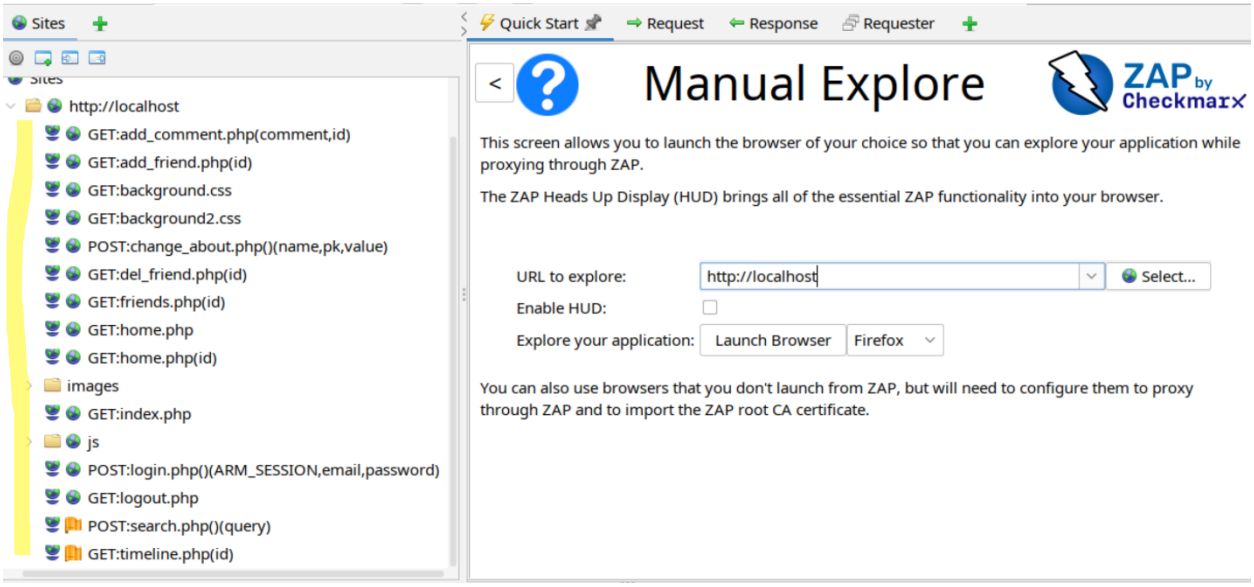
Question 2: Are any of the high-priority OWASP ZAP findings likely to be false positives? If so, why?

No, I don't believe that the high-priority OWASP ZAP findings are false. In my report, there was only one high risk finding which was Reflected Cross-site Scripting. This finding is valid and exploitable. However, I do believe there might be other findings that may be false.

Question 3: Did OWASP ZAP indicate any other vulnerabilities found in the previous section?

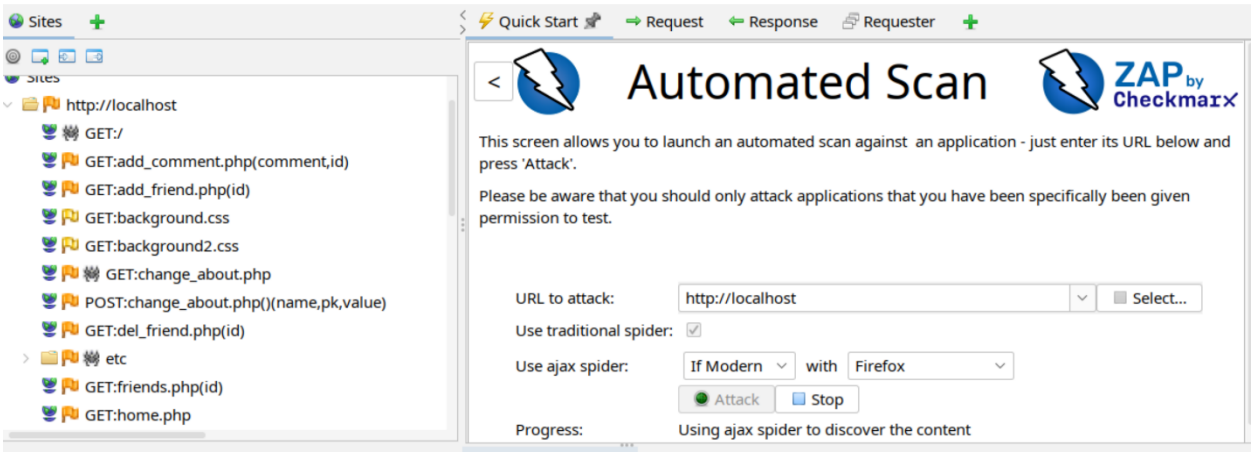
Yes, the OWASP ZAP scan identified two of the three vulnerabilities from the previous section. The report stated the Cross-site Scripting attack was a high-risk vulnerability. The report also stated that the CSRF attack was a medium risk vulnerability due to absence of anti-CSRF tokens. However, OWASP ZAP did not detect the SQL injection attack in the add friend's workflow. In addition to this, the OWASP ZAP scan found multiple other vulnerabilities in the Arm Book web application.

Screenshot showing the manual scan I did in OWASP ZAP:



The sites are everything I explored manually while going through Arm Book.

Screenshot showing my automated scan running in OWASP ZAP:



I am running an automated scan on all the sites I explored while doing my manual scan.

The HTML file from the OWASP ZAP automated scan report is turned in separately.

Section 4: ASVS Web Application Security Auditing

V3: Session Management	
Overall Maturity Level (L1, L2, L3)	Level 1
Justification: Arm Book only implements the basics for session support. For example, cookies are used and there are no tokens in the URL. However, Arm Book fails to implement other security features such as regenerating IDs when a user logs in, use secure, HttpOnly, and SameSite cookie flags, invalidate sessions when a user logs out, and require re-authentication or secondary authentication. These vulnerabilities plus some more on this web application is what earns it a level 1 rather than level 2 or 3. Even though the basics are implemented, there must be more of these vulnerabilities taken care of for this web application to reach a higher maturity level.	
Criteria: Verify that the application never reveals session tokens in URL parameters	**Status:** Pass
I ran a grep command that searched for PHPSESSID in any of the Arm Book code that was provided. I didn't get any matches which means that the web application never reveals session tokens in URL parameters.	
Criteria: Verify that the application generates a new session token on user authentication	**Status:** Fail
I logged in and checked my session cookie, I then logged out and relogged in and the session cookie was the same. I also ran a grep command that searched for session_regenerate_id in the Arm Book code and I didn't get any matches.	
Criteria: Verify that session tokens possess at least 64 bits of entropy	**Status:** Pass
I first created a PHP function that checks the size of the session tokens. I then checked the cookie for the session. I found that the session tokens have over a 64-bit entropy.	
Criteria: Verify the application stores session tokens in the browser using secure methods such as appropriately secured cookies or HTML 5 session storage	**Status:** Pass

I went into the DevTools section in Firefox. From there I went into storage then cookies. Under the cookies section I saw the session cookie. This confirms it is using proper storage.	
Criteria: Verify that the session tokens are generated using approved cryptographic algorithms	**Status:** Unknown
I inspected the php.ini file. I checked to see if session.entropy_file = /dev/urandom and session.sid_bits_per_character >= 6. In the file there was no variable session.entropy_file and the session.sid_bits_per_character was set to 5. Due to this I am unsure is the tokens are generated using an approved cryptographic algorithm.	
Criteria: Verify that logout and expiration invalidate the session token, such that the back button or a downstream relying party does not resume an authenticated session, including across relying parties.	**Status:** Fail
I logged in to Arm Book. I then logged out and pressed the back arrow in the browser. This brought me back into my home page of Arm Book.	
Criteria: If authenticators permit users to remain logged in, verify that re-authentication occurs periodically both when actively used or after an idle period	**Status:** N/A
There is no remember me checkbox in the login page of Arm Book. There is also no long-lived login option in Arm Book. This does not apply.	
Criteria: Verify that the application gives the option to terminate all other active sessions after a successful password change (including change via password reset/recovery), and that this is effective across the application, federated login (if present), and any relying parties.	**Status:** N/A
The application does not offer the option to change your password when logged in or while logging in. The only way to do that is to go through the database. Therefore, this does not apply.	
Criteria: Verify that cookie-based session tokens have the 'Secure' attribute set.	**Status:** Fail
Once logged into Arm Book, I went to DevTools and into the storage section. I then looked at the cookie-based session token. This token said false under the Secure section meaning it is not set.	
Criteria: Verify that cookie-based session tokens have the 'HttpOnly' attribute set.	**Status:** Fail

Once logged into Arm Book, I went to DevTools and into the storage section. I then looked at the cookie-based session token. This token said false under the HttpOnly section meaning it is not set.	
Criteria: Verify that cookie-based session tokens utilize the 'SameSite' attribute to limit exposure to cross-site request forgery attacks.	**Status:** Fail
Once logged into Arm Book, I went to DevTools and into the storage section. I then looked at the cookie-based session token. This token said none under the SameSite section meaning the token does not utilize SameSite to limit exposure to CSRF attacks.	
Criteria: Verify that cookie-based session tokens use the "__Host-" prefix so cookies are only sent to the host that initially set the cookie.	**Status:** Fail
Once logged into Arm Book, I went to DevTools and into the storage section. I then looked at the cookie-based session token. The name of the token was ARM_SESSION. It did not utilize the __Host- prefix.	
Criteria: Verify that if the application is published under a domain name with other applications that set or use session cookies that might disclose the session cookies, set the path attribute in cookie-based session tokens using the most precise path possible.	**Status:** Pass
Once logged into Arm Book, I went to DevTools and into the storage section. I then looked at the cookie-based session token. I looked at the domain which was /localhost and the path which was /. This is the most precise path possible for Arm Book on my VM since all the code for Arm Book is in that path and there are no other applications in that path.	
Criteria: Verify the application allows users to revoke OAuth tokens that form trust relationships with linked applications.	**Status:** N/A
This is not applicable since Arm Book does not have an OAuth or federated login functionality.	
Criteria: Verify the application uses session tokens rather than static API secrets and keys, except with legacy implementations.	**Status:** Pass
I ran a grep command that checked for $_SESSION in any of the Arm Book code and it was in a majority if not all the files. This shows it is used for authentication instead of static API secrets and keys.	

Criteria: Verify that stateless session tokens use digital signatures, encryption, and other countermeasures to protect against tampering, enveloping, replay, null cipher, and key substitution attacks.	**Status:** N/A
This is not applicable since there is no stateless token mechanism anywhere in the code for Arm Book.	
Criteria: Verify that Relying Parties (RPs) specify the maximum authentication time to Credential Service Providers (CSPs) and that CSPs re-authenticate the user if they haven't used a session within that period.	**Status:** N/A
This again is not applicable since there is no single sign on or federated login in Arm Book.	
Criteria: Verify that Credential Service Providers (CSPs) inform Relying Parties (RPs) of the last authentication event, to allow RPs to determine if they need to re-authenticate the user.	**Status:** N/A
There is no federated authentication framework implemented in Arm Book so this is not applicable.	
Criteria: Verify the application ensures a full, valid login session or requires re-authentication or secondary verification before allowing any sensitive transactions or account modifications.	**Status:** Fail
I logged into Arm Book and attempted to edit my profile details. I was successfully able to do so without having to enter my password. There is no re-authentication or secondary verification implemented.	