Project 4:

Web Security Report Entry

Spring 2022

Task 1 – Warm Up Exercises

Activity 1 - The Inspector & Console tabs

1. What is the value of the 'CanYouSeeMe' input? *Do not include quotes in*

   `A_Value_Between_One_And_Ten`

2. The page references a single JavaScript file in a script tag. Name this file including the file extension. *Do not include the path, just the file and extension. Ex: "ajavascriptfile.js".*

project4.js

3. The script file has a JavaScript function named 'runme'. Use the console to <u>execute</u> this function. What is the output that shows up in the console?

   There are 42 bugs on the wall.

Activity 2 - Network Tab

1. What request method (http verb) was used in the request to the server?

POST

2. What status code did the server return?. *Ex: "200"*

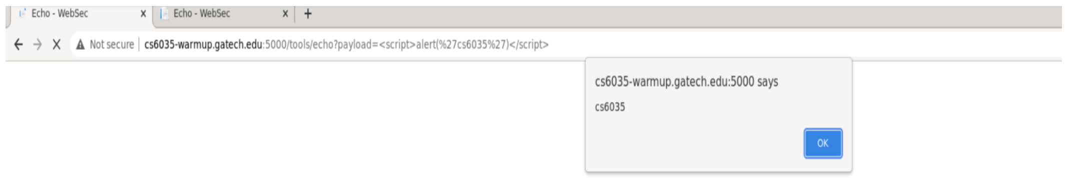
499

3. The server returned a cookie named 'Samy' for the browser to store. What is the value of this cookie? *Do not include quotes in your answer.*

   but_most_of_all_samy_is_my_hero

Activity 3 - Built-in browser protections

1. You can do more than just echo back text. Construct a URL such that a JavaScript alert dialog appears with the text cs6035 on the screen. Upload **activity3.html** and paste in a screenshot of the page with the dialog as your answer below. Be sure to include the URL of the browser in your screenshot.



cs6035-warmup.gatech.edu:5000/tools/echo?payload=<script>alert(%27cs6035%27)</script>

cs6035-warmup.gatech.edu:5000 says

cs6035

Activity 4 - Submitting forms

1. Copy and paste below the entire output message you see and submit that as your answer to this activity. Upload **activity4.html** which is the form that you constructed.

Congratulations!, you've successfully finished this activity. The answer is Stuxnet (2010)

Activity 5 - Accessing the DOM with JavaScript

1. Upload **activity5.html** which is the form that you constructed. No other answers are required for this activity.

Task 5 – Epilogue Questions

Target 1 -- Epilogue

1. List the PHP page and lines that should be changed to fix the vulnerability.

   Page: account.php

   Lines: 12, 27

2. Describe in detail why the code listed in the line numbers above are vulnerable. You're free to use generalized concepts to help show your understanding but we also need to know details that pertain to this target and assignment. A definition of XSRF is not what we're looking for.

   While the code is verifying that a random CSRF token exists in the request, it compares the expected value with a field in the form. Therefore, the attacker could generate a valid token as done in the attack and bypass the verification. Instead the submitted value should be compared with a value stored on the server, like the user session.

3. Explanation of how to fix the code. Feel free to include snippets and examples. Be detailed!

   The key line that should be changed is line 27. Instead of comparing with $_POST['response'] we should compare with $_SESSION['csrf_token']

The token used can be a 128 bit GUID instead of a random number to make it hard to brute force or exploit weak random number generators. OWASP CSRF prevention cheat-sheet have detailed guidelines on the attack prevention with additional techniques like target origin verification and same origin policy (Cross-Site Request Forgery Prevention - OWASP Cheat Sheet Series, n.d.). The guidelines for CSRF tokens are:

- Unique per user session
- Secret
- Unpredictable

Target 2 Epilogue

1. List the PHP page and lines that should be changed to fix the vulnerability.

Page: Index.php

Line: 34

2. Describe in detail why the code listed in the line numbers above are vulnerable. You're free to use generalized concepts to help show your understanding but we also need to know details that pertain to this target and assignment. A definition of XSS is not what we're looking for.

The attribute value of the login input is taken from the request without validation. As an attacker I was able to close the value quote and inject a script that listens to the form submission event and executes the code to steal the user credentials.

3. Explanation of how to fix the code. Feel free to include snippets and examples. Be detailed!
   a. Be careful with your explanation here. There are wrong ways to fix this vulnerability. Hint: Never write your own crypto algorithms. This concept extends to XSS sanitization.
   b. Warning: Removing site functionality will not be accepted here.

The approach is to encode the values coming as inputs with HTML encoding to make sure that there are no special characters that can be used to escape the intended use. PHP has the function htmlspecialchars (GeeksforGeeks, 2021) that can perform this. The code should be:

<input type="text" name="login" value="<?php echo

```
htmlspecialchars(@$_POST['login']) ?>">
```

Target 3 Epilogue

1. List the PHP page and lines that should be changed to fix the vulnerability.

Page: auth.php

Lines: 30-53

2. Describe in detail why the code listed in the line numbers above are vulnerable. You're free to use generalized concepts to help show your understanding but we also need to know details that pertain to this target and assignment. A definition of SQL Injection is not what we're looking for.

The sql injection attack depends on the use of special characters to inject code into the sql query to bypass conditions, steal or corrupt data. The page developer wrote some defense to sanitize the input strings, but the sanitization code itself was vulnerable as well. The code removes some known bad string patterns in order, but with a trick we could inject a combination of the desired pattern separated by another pattern so when the code was executed the end result was the desired pattern. Specifically we wanted to inject a '--' comment pattern (*SQL Injection Prevention - OWASP Cheat Sheet Series*, n.d.), but just appending it to the login name was defended. Instead the injected string was '-#-' so the comment pattern was not detected, but formed after the # character was removed by the defense code.

3. Explanation of how to fix the code. Feel free to include snippets and examples. Be detailed!
   a. Be careful with your explanation here. There are wrong ways to fix this vulnerability. Hint: Never write your own crypto

algorithms. This concept extends to SQL sanitization.

The recommended method to handle SQL injection with PHP is to use prepared statements if supported by the driver (Marin, 2020), and the SQLite driver supports prepared statements PHP: SQLite3::prepare - Manual. Another option is to use the escapeString method in the Database class in salite.php file. The escapeString is designed to escape the string of any special character to prevent injection attacks PHP: SQLite3::escapeString - Manual.

Additional Targets

1. Describe any two additional issues (they need not be code issues) that create security holes in the site.

Issue 1: The website is using clear HTTP for communication. This opens the website to multiple issues including information theft. An attacker can listen to the information and steal sensitive information like user credentials and authentication cookies. The attacker also can modify the information either in the request or response. The attacker can modify the request to modify information like the bank routing information, and modify the response as well to conceal the modification to the user.

Issue 2: Weak password tolerance, and use of MD5 hashing. The website does not have any requirements on the password length or complexity opening it for brute force attacks. The website is also using MD5 hashing algorithm for storing the passwords while MD5 is considered broken (M5: Insufficient Cryptography | OWASP Foundation, n.d.) and exposes the clear users passwords for leakage.

2. Provide an explanation of how to safely fix the identified issues. Feel free to include snippets and examples. Be detailed!

For issue 1: Use HTTPS for encrypting all the requests and responses including non sensitive information, because an attacker can act as a man in the middle and inject malicious content to the unencrypted content even if it is not sensitive (M5: Insufficient Cryptography | OWASP Foundation, n.d.)

For issue 2 use a strong hashing algorithm like SHA256 and enforce more complex password requirements like requiring minimum length and use of numbers and/or special characters (Authentication - OWASP Cheat Sheet Series, n.d.).

Works Cited

Cross-site request forgery prevention cheat sheet. Cross-Site Request Forgery Prevention - OWASP Cheat Sheet Series. (n.d.). Retrieved April 2, 2022, from https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

Cross Site Scripting Prevention - OWASP Cheat Sheet Series. (n.d.). OWASP XSS. https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

GeeksforGeeks. (2021, December 17). How to prevent XSS with HTML/PHP ? https://www.geeksforgeeks.org/how-to-prevent-xss-with-html-php/

SQL Injection Prevention - OWASP Cheat Sheet Series. (n.d.). OWASP. https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

Marin, D. (2020, June 15). SQL Injection in PHP: Practices to Avoid. Okta Developer. https://developer.okta.com/blog/2020/06/15/sql-injection-in-php

M5: Insufficient Cryptography | OWASP Foundation. (n.d.). OWASP. https://owasp.org/www-project-mobile-top-10/2016-risks/m5-insufficient-cryptography

Authentication - OWASP Cheat Sheet Series. (n.d.). OWASP.

https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

When complete, please save this form as a PDF and submit with your HTML files as "report.pdf". Do not zip up anything!