Assignment 2
CS-455
Spring 2019

Due Date: 4/19/2019 at 11:55 pm (No extensions)
You may work in groups of 5

Goals:

1. **Broader goal:** Utilize secure web application development and deployment practices.
2. **Specific goals:** Implement an online banking application that:
   (a) utilizes secure coding practices described in class (e.g., strict mode, object security, etc)
   (b) is secure against cross-site scripting (XSS) attacks
   (c) is secure against broken authentication attacks
   (d) uses XML encoding for passing data between front-end and back-end

Outcomes:

After completing this assignment you will able to:

1. Describe approaches used in the development and deployment of secure web applications
2. Utilize secure design and coding practices to defend against XSS and broken authentication attacks
3. Secure JavaScript code using strict mode
4. Identify and fix XSS and broken authentication vulnerabilities in web applications
5. Implement secure session management using cookies
6. Implement secure data encoding schemes for different web application contexts
7. Implement defense in depth strategies against XSS attacks using Content Security Headers
8. Encode data using XML
9. Document security mechanisms in a web application

OVERVIEW

You will implement a web banking application. The application shall allow users to create multiple bank accounts and then deposit and withdraw money to/from them. The user shall be able to access all of their accounts using a single set of credentials similar to a real-world bank.

Transferring of money between the accounts belonging to the same user shall also be supported. Multiple users shall be able to register and use the application simultaneously.
The application shall feature a secure login, a database (any kind; text files are fine too), and shall use a secure cookie-based session management scheme that allows the user to remain logged in for a period of time. Session management must use secure session handling and avoid **all** flaws discussed in OWASP Top 10 A2:2017.

WEB APP SPECIFICATIONS

Your web app architecture shall comprise a front-end and back-end.

Front-end: The front-end shall feature interfaces for:

- Register as a customer (requiring first and last name, choice of a strong password, and address)
- Logging in with existing user name and password and remain logged in for a period of time. You may find this resource helpful.
- Creating multiple bank accounts and managing all of them at the same time by logging in (similar to a real bank). This includes features for viewing account balances, depositing and withdrawing money to/from accounts, and transferring money between accounts of the same customer
- Logging out
- Functionality for front-end data validation
- Defenses against DOM based XSS. Please follow practices described in OWASP DOM based XSS Prevention Cheat Sheet

Back-end: The back-end shall implement the following functionality:

- A database of all registered users and their account information (e.g., how much money each account has)
- Cookies that allow users to remain logged in. If the user remains idle, the cookie is invalidated after **3 mins** and the user is automatically logged out
- Back-end data validation
- Proper defenses against XSS attacks including input data validation, query escaping, HTTP-only cookies, and Content Security Policy (CSP) headers. It must also follow **all** recommendations outlined in OWASP Cross-site Scripting Cheat Sheet. Please make sure that your escaping is done correctly and appropriately for each context.
- Implements all proper defenses against broken authentication attacks outlined in OWASP Top Ten Project A2:2017.
- Utilizes XML to pass account data between front-end and back-end

All implementations must use strict mode and use other secure JavaScript coding practices discussed in class. You may use any reasonable front-end and back-end design that meets the above functional and, above all, security requirements. The resources below may help you achieve some of the defenses against XSS and broken authentication attacks. All information between back-front and back-end must be encoded using XML (e.g., user information, account information, etc).

Please note: in this assignment you cannot use ejs or other templating engines. This will ensure that you do not solely rely on templating engines to perform sanitization (and they have their limitations too).

README File

Your README file must contain a **detailed** description of your application design and must document the **specific** security design principles used for defending against XSS and broken authentication attacks. You may provide design diagrams showing the application architecture, use cases, etc, as well as their descriptions.

For secure implementation, you can document, for example, "on the login page the application accepts a user name input. The user names are being used in HTML and JavaScript contexts. To protect against these they are being escaped as follows...".

Also, please include the names and emails of group members and anything special about your program.

RESOURCES

- OWASP Top Ten Project A2:2017
- Cross Site Scripting Prevention Cheat Sheet
- OWASP DOM based XSS Prevention Cheat Sheet
- Secure Session Management in Node.js
- xss-filters encoding library
- Bleach library
- DOM Purify
- Content Security Policy (CSP) headers in Node.js
- Node.js XML parser
- Front-end XML parsing

SUBMISSION GUIDELINES:

- This assignment must be completed using Node.js. **You may use `http` or `express` packages. However, no other non-standard packages aside from data sanitization and XML parsing should be used**.

- Please hand in your source code for the html page implementing the front-end and your Node.js JavaScript code. To TITANIUM.
- **If you worked in a group, only one person within each group should submit.**
- Write a README file (text file, do not submit a .doc file) which contains
  - Names and email addresses of all group members
  - How to execute your program
  - Whether you implemented the extra credit
  - Anything special about your submission that we should take note of
- Place all your files under one directory with a unique name (such as `p1-[userid]` for assignment 1, e.g. `p2-mgofman1`).
- Tar the contents of this directory using the following command. `tar cvf [directory_name].tar [directory_name]` E.g. `tar -cvf p1-mgofman1.tar p1-mgofman1/`
- Use TITANIUM to upload the tared file you created above.

Grading guideline:

- Node.js code starts without errors: 5'
- Front-end correctly implements all the required functional features 10'
- Back-end correctly implements all the required functional features 10'
- Data between front-end and back-end is passed using XML encoding 10'
- All required OWASP XSS defenses are correctly implemented 30'
- All OWASP secure session management techniques are correctly implemented 30'
- README file included: 5'
- Late submissions shall be penalized 10%. No assignments shall be accepted after 24 hours.

Academic Honesty:

All forms of cheating shall be treated with utmost seriousness. You may discuss the problems with other students, however, you must write your **OWN codes and solutions**. Discussing solutions to the problem is **NOT** acceptable (unless specified otherwise). Copying an assignment from another student or allowing another student to copy your work **may lead to an automatic F for this course**. Moss shall be used to detect plagiarism in programming assignments. If you have any questions about whether an act of collaboration may be treated as academic dishonesty, please consult the instructor before you collaborate. Details posted at http://www.fullerton.edu/senate/documents/PDF/300/UPS300-021.pdf.