Cyber Security Base Project I

1 Introduction

This is a simple message board application. It intentionally contains several security flaws from the OWASP Top 10 2013 list. This document explains the vulnerabilities and how to find and fix them.

1.1 Tools Used

The application is written in PHP. No external libraries or frameworks are used. Data is stored in a SQLite3 database. The application was tested – and is meant to be run on – PHP's built-in webserver. I tried to make the application as simple as possible, yet functional, to highlight the security issues.

1.2 Structure and Files

The file structure is the following:

config/ contains configuration files

- **database.ini** contains configuration for the database
- **database_schema.sql** contains the schema of the database

include/ contains files that implement functionality for other files

- **authenticate.php** is included for pages that require authentication
- **Database.php** is a wrapper class for database connections
- **MessageAccess.php** contains a class for querying the Message table
- **UserAccess.php** contains a class for querying the User table

templates/ contains html files that are included for other pages

- **header.html** contains the navigation bar

/ (document root) contains the PHP files that implement most of the user interface and business logic

2 Vulnerabilities and Fixing Them

2.1 Injection

MessageAccess and UserAccess classes are used to query the database. However, their implementations are flawed and user input is inserted directly into the query, making SQL injections possible.

Stored procedures or prepared statements should be used to avoid SQL injection. Using a framework or an ORM library that abstracts the database operations might be a good idea, instead of implementing everything manually.

2.2 Broken Authentication and Session Management

User passwords are stored into the database in plain text. The application also enforces bad username/password policies – they are limited to 8 characters and alphanumeric characters but only on the client side. The client side restriction can be bypassed easily to e.g. store a script as the username.

To fix these flaws, user credentials should be salted and hashed before storing into the database. Username/password validity checks should be done on server, not on the client side.

2.3 Cross-Site Scripting

Malicious scripts can be inserted into messages or usernames. The message board displays usernames and messages, causing any script stored to be executed. HttpOnly cookies are not enabled. Consequently, user sessions can be hijacked by stealing session IDs with XSS.

To avoid XSS, all untrusted data should be properly escaped. Some frameworks or templating engines do this automatically. HttpOnly cookie flag should be used as well as X-XSS-Protection header.

2.4 Insecure Direct Object Reference and Missing Function-Level Access Control

Users are shown a "Delete" button for their own messages in the message board. However, the authorization check is only done for the user interface. Any message can be deleted by changing the ID parameter for deletemessage.php.

All authorization checks should be done for the functionality, not just the user interface. Access should be denied by default and allowed only as the special case. For example, an access check should be implemented for deletemessage.php.

2.5 Security Misconfiguration

PHP's built-in webserver is not meant for production use and thus could be seen as a security flaw. File and directory permissions(.htaccess) are not supported which means users can simply download any file in the web root, including the database. Private files should not be in the document root, accessible by the webserver.

To get rid of these flaws, a proper webserver, e.g. Apache, should be used and configured. Access for files and directories should be limited so that only necessary files are publicly available.

2.6 Cross-Site Request Forgery

The application contains multiple pages that allow users to give input which results in state-changing operations, e.g. changing their own password. This means that malicious JavaScript code could be used to make the user's browser do something without the user's knowledge. For example, the user could tricked to click a link to 'myaccount.php?password=csrf!', resulting in their password being changed.

Unique tokens that must be submitted along with data, i.e. CSRF tokens, should be used. Request origins should be checked. Alternatively, a human-check(CAPTCHA) can be used to verify that the request was made by the user. In addition, using GET for sensitive information such as usernames and passwords should not be done, as it can expose those details.

3 Discovering and Exploiting the Vulnerabilities

3.1 Finding the Vulnerabilities

3.1.1 OWASP ZAP

OWASP ZAP can be used to find most of the flaws in this application, although not all of them. Injection, XSS and misconfiguration(HttpOnly, X-XSS) can be found fairly easily. Fuzzing can be used to find the Direct Object Reference vulnerability in deletemessage.php.

To test the application with ZAP, a login session needs to be included to the ZAP context and the logout page must be excluded from scans.

3.1.2 Code Review

Many of the vulnerabilities in the application are a result of poor programming practices. They can be found from the source code fairly easily.

3.2 Exploiting the Vulnerabilities

3.2.1 Injection

1. Register and go to the login page

2. Type in your username

3. Use the following as the password:

```
x’ OR ’1’=’1’;
```

4. You will be logged in without the password
5. (Alternatively, you can log in as some user by using the string from 3. as username)

3.2.2 XSS

1. Register and login
2. Go to the message board (/messages.php)
3. Type:

   ```
   <script> alert(document.cookie);</script>
   ```

   to the message box and add the message
4. The script will be executed when the message board is viewed

3.2.3 Insecure Direct Object Reference and Missing Function-Level Access Control

1. Register two users and login as the first one
2. Add a message to the message board and view the ”Delete” button’s HTML source
3. Take the ID parameter from the link in the HTML(”/deletemessage?id=x”)
4. Log in as the second user
5. Go to /deletemessage?id=x where x is the id from step 3
6. The first user’s message will be deleted
7. (Alternatively, do steps 1-2 and use OWASP ZAP to fuzz deletemessage?id=x with random integers)

3.2.4 Security Misconfiguration

1. Go to the URL /database.sqlite
2. The server will allow you to download the database

3.2.5 Broken Authentication and Session Management

1. Download the database
2. Access it with sqlite3 and query the user table(SELECT * FROM user;)
3. You'll get all users and plain text passwords
4. (Alternatively, exploit the XSS vulnerability to obtain valid session IDs to hijack user sessions)

3.2.6 Cross-Site Request Forgery

1. Register and login
2. Go to the URL /myaccount.php?password=csrf
3. Your password will be changed