Report

CCSEP Assignment Semester 2 2019

ISEC3004

Kevin Le – 19472960

1. System Overview

System is built on a LAMP stack, consisting of Linux, Apache Web Server, MySQL and PHP technologies.

File structure:

```
/website/access.php
         adminpanel.php
         db.php
         index.php
         item.php
         login.php
         register.php
         sell.php
         user.php

/website/assets/css/form.css
                    styles.css

/website/assets/img/favicon.png
                    logo.png
                    userImage.png

/website/assets/js/jquery.min.js

/website/assets/bootstrap/css/bootstrap.min.css
/website/assets/bootstrap/js/bootstrap.min.js
```

2. Vulnerabilities

2.1. XSS

2.1.1. Reflected (Non-Persistent) XSS

Description: Reflected XSS is when user input parameters containing code is run (displayed/ reflected) on the victim's webpage. The input containing code is not stored in the application and is non-persistent, only showing to users who have clicked a link containing the submitted query string. The attackers input parameters containing a malicious script is interpreted and executed by the browser, which it assumes to be genuine code as it cannot determine which code is which. This can allow attackers to steal cookies containing session IDs, redirecting to a fake website, or many other things that can be done with a browser. This method of XSS spreads by sharing links containing the submitted query string. In this case, the users search query is reflected on the webpage without any sanitization, allowing it to contain executable code.

Location: `index.php` displays the users submitted search query without any sanitization.

- Offending input field: Line 70 in `index.php`
- Reflected at: Line 78 in `index.php`

```
if(isset($_GET["search"]))
{
    $name = $_GET["search"];
    echo "<p style=\"padding-top: 10px;\">Searching for: $name</p>";
}
```

Trigger: Any standard JavaScript.

- Inside search field enter: `<script>alert('Hello')</script>`
- Click "Search".
- Script will be run displaying alert box with message "Hello" on webpage.
- Resulting URL: `___/?search=<script>alert('hi')<%2Fscript>`

Removal/ Mitigate: Modify the program so it does not reflect user search input at all by removing the reflection code, and never insert untrusted data. If the input must be reflected, sanitize and escape user input such as removing < and > characters and stripping any other HTML tags, this can be done with the php function `strip_tags()` which strips all HTML and PHP tags from a string. Further prevention can be checking for any malicious input, and if any strings deemed suspicious or trigger a filter/ keyword then outright reject the string in a paranoid approach, it is also possible to encode input to prevent the browser from executing any unwanted HTML, especially if HTML code is meant to be displayed (E.g. forums). [1] In php, this can be done with `htmlspecialchars()` which converts HTML characters to HTML entities. The potential severity can be reduced if a successful

attack were to occur by using the content security policy (CSP). This specifies domains that the browser should consider as valid and will only execute scripts loaded from those whitelisted domains. [2] This also prevents scripts opening connections to unauthorized locations.

2.1.2. Stored (Persistent) XSS

Description: User input parameters containing code is stored by the website and displayed to all visitors as a normal page, which the browsers render/ execute as trusted code. More damaging due to potential rate of spread without need to spread a submitted query string. In this case, an item listings description is not sanitized for code allowing for code to be stored and rendered by a victim when the item listing is accessed.

Location: `sell.php` accepts input for an item listing with fields: name, description, and price. The description is accepted without any sanitization and is stored in the database table “items” which contains items for sale. When a victim goes to view an items details, the item description is fetched and rendered on the `item.php` page, executing the code.

- Offending input field: Line 55 in `sell.php`
- Reflected at: Line 117 in `item.php` when page loaded for item.

Trigger: Any standard JavaScript.

- Login to a user account.
- Click “Sell Something!” to begin an item listing.
- Set Name to anything.
- Inside Description field enter: `<script>alert(‘Hello’)</script>`
- Set Price to anything.
- Click “Sell”.
- Click on item that was just listed.
- Script will be run displaying alert box with message “Hello” on item page load.

Removal/ Mitigation: Do not accept user input without proper sanitization. Perform sanitization before storing item listing in database on the `sell.php` page. Escape dangerous characters. Follow mitigation steps as in (2.1.1).

2.2. SQL Injection

2.2.1. SQL Injection – Regular

Description: User input is not properly sanitized, and the input given contains potentially malicious SQL statements which are then executed on the database. Allows tampering with data, destroying

data, accessing restricted data, bypassing authentication, etc. In this case, user authentication is based on if an email and password combination return a row, this opens the possibility of accessing any account by performing a SQL injection by selecting the desired accounts row.

Location: User logs in via the login page located at `./access?page=login.php`. This page requests an email and password for authentication. The authentication is performed by building a query after hashing the entered password and trying find a row where the username and password match, calling `mysqli_query` to perform the query using the built statement, and then retrieving the number of rows returned by calling `mysqli_num_rows`.

- SQL Injection Location: Line 55 and 61 in login.php.

```
$sql = "SELECT * FROM users WHERE email = '$email' AND password = '$hashpwd'";
```

```
"SELECT uid, name, admin, enabled FROM users WHERE email = '$email' AND password = '$hashedpwd "
```

Some Other Locations:

- Line 101: Search Field in index.php
- Line 44, 54, 65, 66, 100 in item.php
- Line 59 in register.php
- Line 73 in sell.php
- Line 104 in user.php
- Line 40, 50, 60 in adminpanel.php

Trigger: All SQL Operations:

- Go to login page.
- For email enter: `anEmail' OR uid=1#`
    - This comments out the rest of the SQL statement and hijacks the WHERE clause. Making it return the row for WHERE UID 1.
- No need to enter anything for password.
- Click Login.
- Bypasses authentication and now have access to Administrator account.

Removal/ Mitigation: Use prepared statements to build a query. A template where constant values are substituted into placeholders during execution. This separates SQL from user input parameters, allowing the database to distinguish between actual code and user input. Interpreting any SQL input as literally the value entered. Sanitizing input is not worth the danger with prepared statements being much better for readability and safety. Further prevention can be done in the input, before allowing submission, check if a valid email has been entered. Can be done using the form type=”email”. This doesn’t always prevent SQL Injection for many fields taking any entry but helps.

```
$stmt = $db->prepare("SELECT * FROM users WHERE email = '?' AND password = '?'");
$stmt->bind_param("ss", $email, $hashedpwdentry);

$stmt->execute();
```
[3]

2.2.2. SQL Injection – Blind

Description: User input not properly sanitized, input contains potentially malicious SQL statements. Allowing dangerous/ destructive database operations. Like regular SQL injection but attacker does not receive any error messages and/ or other sources of information to tailor a special attack. This requires an attacker to perform trial and error by asking TRUE/ FALSE questions to probe the pages vulnerability.[4] In this case, an item listing information page is fetched using the URL: `/item.php?item=ITEMID` allowing an attacker to do just that.

This sends a query to the database where $iid = ITEMID:

```
$sql = "SELECT items.itemname, items.price, items.description, users.name, users.uid FROM items
INNER JOIN users ON items.sellerid = users.uid WHERE items.iid = $iid";
```

The attacker may then try attach some conditions to the URL parameters to see if the page changes.

Location: When `item.php` takes the URL parameters and performs a query to return the data for the specified item id.

- SQL Injection Location: Line 100 in `item.php`

Trigger: All SQL Operations.

- Use normal item page URL: /item.php?item=3 (assuming valid iid = 3) and after 3 add:
- AND 1=2
  - `/item.php?item=3 AND 1=2`
  - Observe page no longer returns result for ITEM
- AND 1=1
  - `/item.php?item=3 AND 1=1`
  - Observe page returns item again.
- AND IF(1=1, SLEEP(5), FALSE) [5]
  - `/item.php?item=3 AND IF(1=1, SLEEP(5), FALSE)`
  - Observe it takes webpage 5 seconds to load. Also known as time-based SQL Injection making the database perform long wait operations.
- All of these proves indicates page is vulnerable to Blind SQL Injection.

Removal/ Mitigate: Use prepared statements to build a query. Follow mitigation for Regular SQL (2.2.1.)

2.3 Broken Access Control

Description: The application gives users access to areas that they should not have access to. Verification of permissions occurs after the user follows a login sequence which determines what the user can do in terms of their account permission level. In this case the permission checks are incorrectly implemented, therefore failing and allowing users with insufficient permissions to access areas beyond their permission levels. In this case the developer likely forgot to check if the current logged in account had admin rights before giving access to the admin panel. This error often occurs due to access control checks being inserted all over the system making it difficult to detect. This system identifies admin user accounts by checking the database if ADMIN = TRUE (1). Another failure of the system is allowing path traversal, insecure IDs (database uses auto increment by 1 for user IDs) and improper file permissions which are more forms of broken access control.[16]

Location: where program fails to check if user logged in has admin rights and server setup allows path traversals.

- Line 23 in `adminpanel.php` with lack of admin right check.

Trigger: User logged in can access admin panel by simply entering the path to page even without adequate permission level. User can also traverse files on the server

- `/adminpanel.php`
- `/assets/`
    - Able to list files and access files which are all unprotected. Could possibly access sensitive content.

Removal/ Mitigate: Clearly specify pages and what s*ecure means* for the system with a security policy which specifies what secure behaviours the system must have; the policy should specify user types/ levels and what functions or information each user level can access. This will reduce the likelihood of human error in terms of implementing access controls in the system, especially when done in conjunction with testing access control systems in place to ensure its strength. Utilising an access control matrix to clearly specify user rights would also help prevent broken access control.[16]

2.4. Broken Cryptographic Algorithm

Description: Using hashing algorithms classified as broken or risky (e.g. MD5 or DES). The hash functions in question have been proven to be “breakable” where an attacker is able to find the hashed data. This often occurs when there are successful collision attacks proving the hash

functions failed resistance against such attacks, where the attacker finds two inputs hashing to the same value. In this case, the website and database use MD5 hashes for passwords which has been proven to fail collision attacks[6], potentially taking seconds on a regular PC to find a collision. [7][8]

Location: When passwords are hashed to store in the database (on registration) and when entered passwords are hashed to check for match (on login).

- Database
- Registering using MD5 hash to store new password: Line 57 in `register.php`
- Logging in using MD5 hash to compare entered password hash with stored password hash: Line 53 in `login.php`

Trigger: When registering for account with new password or logging in with existing password the insecure MD5 hash is used.

- On the Login page.
- Click “Register”.
- Create a new account filling in Name, Email, and both Password fields.
- New account will be made using MD5 hash to store the passwords in the users table.
- Click “Login”.
- Login using valid Email and Password.
- Login will be validated by converting MD5 hash to password and comparing to hash stored in users table.

Removal/ Mitigate: Use unbroken, up to date cryptographic algorithms. Currently one of the best is Bcrypt which is slow enough to make brute forcing unrealistic on a large scale.[9] This should also be backed up by having strong password requirements such as length. In this case the MD5 hashing would be replaced with PHPs `password_hash()` function, where `PASSWORD_DEFAULT` is currently the BCrypt algorithm (As of PHP5.5.0). [10][11]

```
password_hash("mypasswordisgreat", PASSWORD_DEFAULT)
```

2.5. PHP File Include

Description: Webpage has a user modifiable input that can change the path of a file to include. Allows an attacker to override the included file and perform Local File Inclusion, where they can choose files currently on the server to include or Remote File Inclusion where the website loads the file they want to include from an FTP server or another website. This means an attacker can control what file they want to execute. Enables code execution on client viewing page or webserver, denial of service and disclosure of sensitive information if file permissions aren’t set up properly. [12] In this

case the application has an access page containing a URL which specifies which page to load, this can be hijacked.

Location: When `access.php` is given a page to load for managing new/ existing users depending on what is called. The URL loads either `/access.php?page=login.php` or `/access.php?page=register.php`. This indicates the page may be vulnerable, if any other input is not validated.

- Offending Include Statement: Line 23 in `access.php`

```
if(isset($_GET['page']))
{
    $accesspage = $_GET['page'];
    include "$accesspage";
}
```

Trigger: Any compatible file.

- Observe URL: `/access.php?page=login.php`
- Appears to be vulnerable. Test if script directly includes the input by trying to access files on server (directory traversal):
- `/access.php?page=../../../etc/passwd`
- Observe it returns the passwd file.
- Vulnerable to PHP Include attack.

Removal/ Mitigate: Do not pass any user input into a part of code. If not possible maintain a list of allowed files in access.php, then use an index/id to access the files. [12]

```
if(isset($_GET['page']))
{
    $allowedpages = array('login', 'register');
    $pageidx = $_GET['page'];
    include "$allowedpages[$pageidx].php";
}

/access.php?page=0
```

2.6. Server Misconfiguration

Description: Flaws in server setup such as badly configured permissions, unnecessary features enabled (e.g. open ports, services), default accounts enabled, overly detailed error messages, etc. [13] In this case the website displays too much information on the login page, identifying which email or password field is incorrect. This gives an attacker much more information and makes it easier to brute force/ guess the email or password. This doesn't make much functional sense as users can have same passwords.

Location: Error outputs on Line 100, 104 and 108 in login.php where login doesn't match. Visible to user on the login page.

Trigger: When user enters incorrect login, page will specify what field is incorrect.

- Create account on register page.
- Attempt to login to account with correct email but incorrect password.
    - Observe message "Incorrect Password".
- Attempt to login to account with correct password but incorrect email.
    - Observe message "Incorrect Email".
- Attempt to login to account with incorrect password and email.
    - Observe message "Incorrect Email *and* Password".
- Server misconfiguration.

Removal/ Mitigation: Don't output verbose error messages. Instead state "Email or Password is Incorrect". This increases the difficulty of brute forcing/ dictionary attacks (but does reduce UX to some extent). It also makes it easier on the programmer as there is no need to determine what is wrong, since this query is enough:

```
$sql = "SELECT * FROM users WHERE email = '$email' AND password = '$hashedpwdentry'";
```

2.7. Use of Hard-Coded Passwords

Description: Using a hard code password stored in the source code for inbound or outbound communication to external components [14]. In this case the application makes a connection to the database whenever any data is to be stored/ retrieved. The connection to the database has the server, username, password and database name hardcoded. This allows an attacker to potentially discover these credentials and gain access to unauthorized data (e.g. if the page fails to load the data could be dumped).

```
define('DB_SERVER', 'localhost');
define('DB_USERNAME', 'student');
define('DB_PASSWORD', 'CCSEP2019');
define('DB_DATABASE', 'assignment');
```

Location: db.php file is included in every PHP file that needs a connection to perform operations on the database.

- Vulnerable Hardcoded Credentials: Line 18-21 in `database.php`
- Used in: `access.php, adminpanel.php, index.php, item.php, sell.php, user.php`

Trigger: Whenever database connection is required, it uses the connection in database.php which was made with a hard-coded password.

Removal/ Mitigation: Use an encrypted configuration file stored outside the server directory.[15] Use a PHP file that sets the server, username, password and database variables. This file is then included at the top of the page. Ensure access to the configuration file is restricted and not in root directory.

3. Setting up Application

Assuming Linux, Apache Web Server, MySQL and PHP already setup on server.

Creating the database:

1. Check MySQL is running
2. Check MySQL settings in /src/database.php are correct.
3. Login to MySQL
4. Run commands to create a new database. (Easy to copy file located in assignment ZIP root, called schema.txt):

```
CREATE DATABASE assignment;
USE assignment;

CREATE TABLE `users` (
  `uid` int(50) unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(255) NOT NULL,
  `email` varchar(320) NOT NULL,
  `password` varchar(500) NOT NULL,
  `balance` double(255,2) DEFAULT '0.00',
  `admin` BOOLEAN DEFAULT FALSE,
  `enabled` BOOLEAN DEFAULT TRUE,
  PRIMARY KEY (`uid`)
);

INSERT INTO `users` (`name`, `email`, `password`, `balance`, `admin`) VALUES ('admin',
'admin@ebuy.com', '5f4dcc3b5aa765d61d8327deb882cf99', '1000.00', TRUE); #Default password:
password

CREATE TABLE `items` (
  `iid` int(50) unsigned NOT NULL AUTO_INCREMENT,
  `sellerid` int(50) unsigned NOT NULL,
  `itemname` varchar(255) NOT NULL,
  `price` double(255,2) DEFAULT '0.00',
  `description` varchar(2000) DEFAULT 'Seller has no description',
  `sold` BOOLEAN DEFAULT FALSE,
  PRIMARY KEY (`iid`),
  FOREIGN KEY (`sellerid`) REFERENCES `users` (`uid`)
);

#Some data to insert

INSERT INTO `users` (`name`, `email`, `password`) VALUES ('william', 'willy@email.com',
'32250170a0dca92d53ec9624f336ca24'); #Password: pass123, id = 2
INSERT INTO `users` (`name`, `email`, `password`) VALUES ('jon', 'jon@email.com',
'b4af804009cb036a4ccdc33431ef9ac9'); #Password: pass1234, id = 3

INSERT INTO `items` (`sellerid`, `itemname`, `price`, `description`) VALUES
  ('2', 'iPhone 7 32gb', '300', 'Good condition, no scratches'),
  ('2', 'Trampoline', '100', 'Disassembled'),
  ('3', 'Helicopter', '100000', 'Need large enough trailer'),
```

```
('3', 'Desktop Computer', '600', 'Selling due to upgrade');
```

Loading web application files:

1. Apache files located at:
    a. Default directory located at /var/www/html **OR,**
    b. VM SMB server in the website folder.
2. Upload all files in /src/ to the Apache folder.

Connect to webpage using Virtual Machine host adapter IP, enter IP in browser address field. (Application tested on Firefox).

4. Known Defects

Currently none found.

5. References

[1] Obtained from *Rapid7* at “Cross Site Scripting”. Retrieved from: https://www.rapid7.com/fundamentals/cross-site-scripting/ Accessed 30/09/19.

[2] Obtained from *Mozilla* at “CSP”. Retrieved from: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP Accessed 30/09/19.

[3] Obtained from *W3* at “PHP MySQL Prepared Statements”. Retrieved from: https://www.w3schools.com/php/php_mysql_prepared_statements.asp Accessed 30/09/19.

[4] Obtained from *OWASP* at “Blind SQL Injection”. Retrieved from: https://www.owasp.org/index.php/Blind_SQL_Injection Accessed 30/09/19.

[5] Obtained from *Acunetix* at “Blind SQL Injection”. Retrieved from: https://www.acunetix.com/websitesecurity/blind-sql-injection/ Accessed 30/09/19.

[6] Obtained from *Carnegie Mellon University* at “MD5 vulnerable to collision attacks - VU#836068”. Retrieved from: https://www.kb.cert.org/vuls/id/836068 Accessed 30/09/19.

[7] Obtained from *Wikipedia* at “Hash Function Security Summary”. Retrieved from: https://en.wikipedia.org/wiki/Hash_function_security_summary Accessed 30/09/19.

[8] Obtained from *Cryptology ePrint Archive* at “Fast Collision Attack on MD5”. Retrieved from: https://eprint.iacr.org/2013/170 Accessed 30/09/19.

[9] Obtained from *Auth0* at “Hashing in Action: Understanding bcrypt”. Retrieved from: https://auth0.com/blog/hashing-in-action-understanding-bcrypt/ Accessed 01/10/19.

[10] Obtained from *OWASP* at “Password Storage Cheat Sheet”. Retrieved from: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#ref7 Accessed 01/10/19.

[11] Obtained from *PHP* at “password_hash()”. Retrieved from: https://www.php.net/manual/en/function.password-hash.php Accessed 01/10/19.

[12] Obtained from *OWASP* at “Testing for Local File Inclusion”. Retrieved from: https://www.owasp.org/index.php/Testing_for_Local_File_Inclusion Accessed 01/10/19.

[13] Obtained from *OWASP* at “Top 10-2017 A6-Security Misconfiguration”. Retrieved from: https://www.owasp.org/index.php/Top_10-2017_A6-Security_Misconfiguration Accessed 01/10/19.

[14] Obtained from *CWE* at “CWE-259: Use of Hard-coded Password”. Retrieved from: https://cwe.mitre.org/data/definitions/259.html Accessed 01/10/19.

[15] Obtained from *Sonarsource* at “PHP Rules - RSPEC-2068”. Retrieved from: https://rules.sonarsource.com/php/RSPEC-2068. Accessed 01/10/19.

[16] Obtained from OWASP at “Broken Access Control”. Retrieved from: https://www.owasp.org/index.php/Broken_Access_Control Accessed 11/10/19.

database.php based on db.php by Jonathon Winter in CCSEP Practical 6 – Website Injection. Accessed 27/09/19.

Website design generated using Bootstrap Studio (https://bootstrapstudio.io/) with Student Licence.

Bootstrap v4.3.1 (https://getbootstrap.com/)
Copyright 2011-2019 The Bootstrap Authors
Copyright 2011-2019 Twitter, Inc.
Licensed under MIT (https://github.com/twbs/bootstrap/blob/master/LICENSE)

- **Files:**
  - assets/js/**jquery.min.js**
  - assets/css/**form.css**
- **All Files in:**
  - assets/bootstrap/

Item.php placeholder item image from https://placeholder.com/
User.php placeholder profile image from http://oakclifffilmfestival.com/jury/placeholder-user/
“Ebuy” logo designed by Kevin Le.