INDIVIDUAL ASSIGNMENT TECHNOLOGY

PARK MALAYSIA

CT123-3-3-ASC ADVANCED SOFTWARE SECURITY

APU3F2305CS(CYB)

Lee Woon Xun

TP067738

HAND OUT DATE : 12 OCT 2023

HAND IN DATE : 31 DEC 2023

WEIGHTAGE : 50%

Table of Contents

Table of Figure

Abstract

The assignment's main focus is on creating safe systems, and available options include command monitoring, web-based apps, and vulnerability detection. The system's development team focuses on confidentiality, integrity, and availability (CIA) and uses HTML, PHP, and Python. Common vulnerabilities are examined and three secure coding practises will be implemented into practise, including SQL injection, missing function-level access control (MFAC), and session hijacking. The role that secure coding plays to achieve security objectives is explained, and examples showing the consequences of not using it are provided. The evaluation criteria include the following: thoroughness of study, lucid system design, specific vulnerability analysis, pertinent secure code, and lucid discussion. The web-based system vulnerability detection and command monitoring system's objectives, goals, and procedures are outlined in the system design section, which also highlights the security precautions used at each step.This assignment provides a thorough guide to developing secure systems overall.

1. Introduction

For system creation to begin, more information about the chosen system's level of difficulty must be gathered. No matter if you're working with a web app, a vulnerability retrieval and warning system, or a command monitoring system, you need to know each step of the process. This means laying out the system's goals, limitations, and necessary functions.

The design step is based on the research results. It involves coming up with a way to build a system that includes security features without making them stand out. This is shown by strong processes for checking data, encrypted channels for talking, and safe ways to prove who you are. HTML, PHP, and Python will be used for the study because they are flexible, widely used, and have a lot of tool and library support for security.

2. System Design

2.1 Scope and objectives

It's important to think about the system's goals and reach when you're building it. Building safety web applications is our primary objective as we work on web applications. The major objective is to meet the CIA's security goals of Privacy, Availability, and Integrity. This means setting up a system that keeps data and system resources safe from people who aren't supposed to see or change them and makes sure they're available when they're needed.

2.2 Processes Involved

Three systems will be built as part of this project to fix three security holes: SQL attack, missing function level access control (MFAC), and session control.

Front page for SQL Injection and Session Hijacking:
HTML is used to make the user interface, which is what allows users to connect with it. HTML was chosen because it is easy to use and is widely used for building websites.

SQL Injection and Session Hijacking backend:

PHP will be used as the backend programming language for server-side processing in the backend of SQL Injection. It makes it easier to create dynamic material and connect to databases. MySQL database control tool PHPMyAdmin is used to take care of databases.

Missing Function Level Access Control (MFAC):

Python is used for study and demonstration in this system. Python is the best language for demonstrating how to use this weakness because it is flexible and has a lot of tools that can be used with it. Additionally, it's easier to understand how MFAC works.

3. Common Vulnerability

If you want to make interconnected systems safer, you should look for and understand the weaknesses that a majority of them share. It is very important to pay attention to these weaknesses when building a system that will safeguard itself from threats of their own. As a security issue, weaknesses are one of a kind because they make it hard to meet the goals of confidence, integrity, and availability.

3.1 SQL Injection

SQL injection is a vulnerability that allows people modify databases by entering in data that hasn't been checked. Someone could change info or get to it without permission if this is completed.

3.2 Missing Function Level Access Control (MFAC)

Another weakness is that there's missing function-level access control. This is when someone can use features or tools that they shouldn't be able to. This way, people who shouldn't be able to could get to or change information.

3.3 Session Hijacking

Lastly, there is session hijacking. If the session control isn't executed right, hackers could get into a user's account and steal their data.

4. Possibility of Exploitation

To really do well on this test, you need to understand how your weaknesses can be used against you. With SQL injection, they could add bad SQL queries to get around protection or get back private data. Using prepared statements, tailored queries, and input checks can help keep this from happening.

When users have unauthorised access to certain functions, this is called exploitation if it includes Missing Function Level Access Control. By setting up the right entry controls and role-based permissions, this risk can be lowered.

Attacks on sessions take advantage of holes in the system that manages sessions. An attacker might use methods like session sniffing or predicting session tokens. To stop this, secure coding methods like encrypting session tokens and renewing them often are very important.

5. Secure Code Implementation

Implementing secure coding techniques is the main focus of the assignment.

5.1 SQL Injection Secure Code

```php
$input_username = filter_input(INPUT_POST, 'username', FILTER_SANITIZE_STRING);

// Sanitize and filter the 'password' POST input
$input_password = filter_input(INPUT_POST, 'password', FILTER_SANITIZE_STRING);
```

Figure 1 filter input PHP.

```php
if ($result->num_rows > 0) {
    $row = $result->fetch_assoc();
    $hashed_password = $row['User_password'];


    if (password_verify($input_password, $hashed_password)) {
```

Figure 2 apply hash password to protect.

The code should use parameterized queries and input validation to prevent SQL injection. This guarantees that user-inputted data has been cleaned and does not represent a threat to the database.

5.2 Missing Function Level Access Control (MFAC) Secure Code

```
#class function
class AthenticationUser:
    def __init__(self):
        self.users = {
            'Bob' : {'password':'123', 'role':'user'},
            'Ali' : {'password':'pass', 'role':'user'},
            'admin' : {'password': 'admin', 'role':'admin'}
        }
```

Figure 3 add role for each user.

```
    #compare username and password in above users list
    def verify(self, username, password):

        #found the user in the user list
        if username in self.users and self.users[username]['password'] == password:
            if self.users[username]['role']:
                return self.users[username]['role']
        else:
            return None
```

Figure 4 verify user condition.

```
    #main thing in MFAC tp verify role
    def printResullt(self, user_role):
        if user_role == 'admin':
            return {self.data['Admin_page']}
        elif user_role == 'user':
            return {self.data['User_page']}
        else:
            return {self.data['Not_Found']}
```

Figure 5 verify role.

```
    check_User = auth_functuin.verify(username, password)
    #print(check_User)

    if check_User:

        print(f"Usersname: {username}")
        print(f"Password: {password}")

        #check result
        data = getMessage.printResullt(check_User)

        print(f"User Role: {check_User}")
        print(f"Data: {data}")
    else:
        print("Authentication unsuccessful")
```

Figure 6 main function identity and print result.

It takes appropriate access control measures to address missing function level access control. Users can only access features that are appropriate for their positions by implementing role-based access control, or RBAC. From figures 3 to 6, RBAC is mainly used to correct the system code and logic to identify each user in real time.

5.3 Session Hijacking Secure Code

```
// Set session cookie parameters with HttpOnly and Secure attributes
session_set_cookie_params([
    'httponly' => true,
    'secure' => true,
    'samesite' => 'Strict', // or 'Lax' depending on your requirements
]);
```

Figure 7 setup cookie.

```
// session timeout in 1 hour
ini_set('session.gc_maxlifetime', 3600);
```

Figure 8 set the session time only 1 hour after relogging.

```
//check user ip if change then invalid the session
if (isset($_SESSION['user_ip']) && $_SESSION['user_ip'] !== $_SERVER['REMOTE_ADDR']) {
    // Invalid session, possibly hijacked
    session_unset();
    session_destroy();
    header('Location: login.html'); // Redirect to login
    exit();
}
```

Figure 9 check exit ip before more for different device.

```
$_SESSION['user_ip'] = $_SERVER['REMOTE_ADDR'];
```

Figure 10 check exit IP and apply.

```
// Check if the session already exists
if (isset($_SESSION['user_agent']) && $_SESSION['user_agent'] !== $_SERVER['HTTP_USER_AGENT']) {
   // Invalid session, possibly hijacked
   session_unset();
   session_destroy();
   header('Location: login.html'); // Redirect to login
   exit();
}
```

Figure 11 check exit user agent before more for localhost testing.

```
$_SESSION['user_agent'] = $_SERVER['HTTP_USER_AGENT'];
```

Figure 12 if no exit user agent before after verify success user login then keep the user agent.

```
session_regenerate_id(true);
$sessionID = session_id();
```

Figure 13 set the generation session id related cookie id and store the id.

It is critical for secure session management in order to prevent session hijacking. This include applying HTTPS, identifying IP addresses, identifying user-agent control, employing secure session tokens, and on a regular basis renewing session identifiers. In figures 7 to 13, all methods are used to prevent Session Hijacking from threatening the web-based system. Including the session setting duration in Figure 8, once the time is exhausted, all session data sets will be logged out. Figure9 and Figure11 are the user-agent that identifies the user's IP address and browser. Once the data exists before logging in, it will automatically log out and use the session data to allow the user to log in again. In Figure 13, a new session id will be generated for the user each time.

6. Purpose and Important of Secure Coding

The primary purpose of secure coding is to enhance a system's protection against any security risks. Developers strengthen the system's general availability, confidentiality, and integrity through implementing secure coding behaviours into action.

The capacity of secure code to prevent system manipulations, unauthorised access, and data breaches makes it important. Reducing the potential of security issues and guaranteeing the system's resilience against changing cyber threats are two benefits of using secure coding practises.

Additionally, secure coding increases user and stakeholder trust while also improving regulatory compliance. It covers the entire security software development life cycle and is a key element that transforms the entire software development into an effective development and prevents the consideration of the security part to prevent secondary development.

7. Scenarios and Impact

7.1 SQL injection

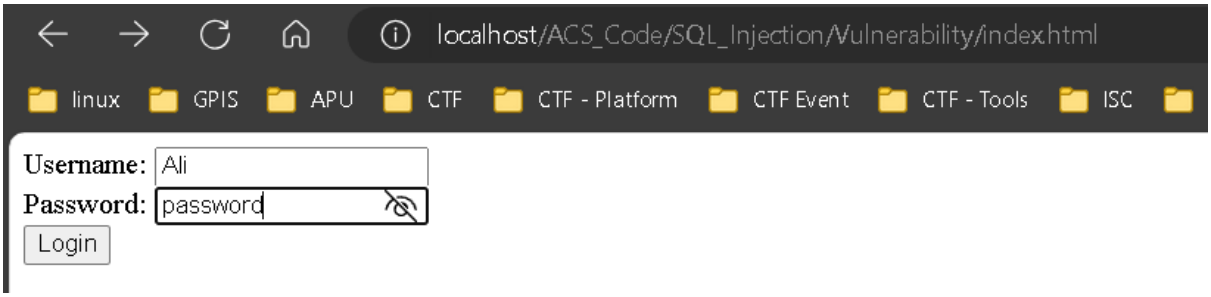

Figure 14 normal login page.

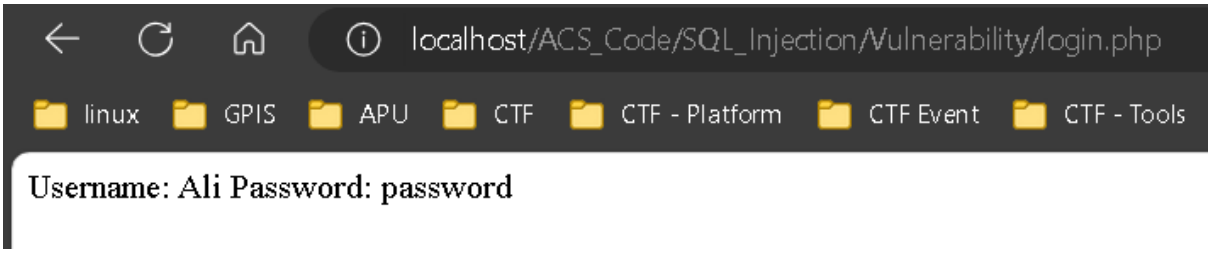

Figure 15 if login successful.

In the ordinary login interface, users can enter their registered account and password to log in to the system. Figure 15 shows the reflection after the user "Ali" logs in normally.

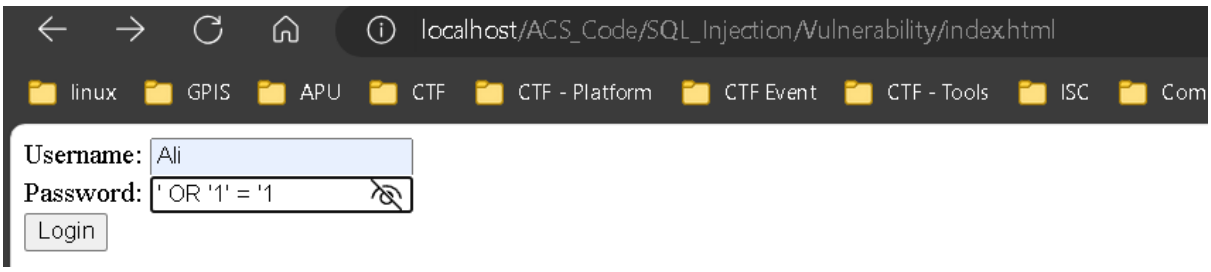

Figure 16 implement SQL injection.

After detection, the attacker found that the login entrance was not protected in any way and launched a SQL Injection attack. The chart in Figure 16 shows that the attacker only needs one line of SQL code to crack the vulnerability and log into the system.

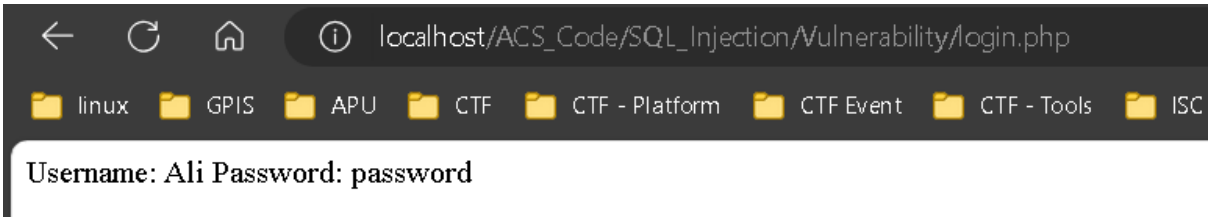

Figure 17 get the user password.

The attacker's attack method using SQL injection is undoubtedly to access the database in the simplest way, and the SQL injection attack method is unauthorized access or even permissions. Therefore, private data such as usernames and passwords may be queried, added, modified and deleted by attackers. Inadequate secure coding processes may result in major data breaches, compromised user privacy, and possibly legal consequences.

7.2 Missing Function Level Access Control (MFAC)

Missing function level access control is a system development language that simulates MFAC attacks by Python.

```python
#simulation MFAC function
def main_mfac():
    auth_functuin = AthenticationUser()
    getMessage = Result()

    #as user/admin login
    username = 'Bob'
    password = '123'

    check_User = auth_functuin.verify(username, password)
    #print(check_User)

    if check_User:

        #check result
        data = getMessage.printResullt(check_User)

        print(f"Users: {check_User}")
        print(f"Data: {data}")
    else:
        print("Authentication unsuccessful")

#run systemn
main_mfac()
```

Figure 18 main function to run MFAC simulation.

What is shown in the above diagram is that the main function named 'main_mfac' mainly allows the key part of the system, in which the call class method is used to command its two class functions. A print result that verifies the user's identity.

```
#class function
class AthenticationUser:
    def __init__(self):
        self.users = {
            'Bob' : {'password':'123'},
            'Ali' : {'password':'pass'},
            'admin' : {'password': 'admin'}
        }

```

Figure 19 default users' data list.

From figure 19 shows how user data is always stored when using the Python development language.

```
#compare username and password in above users list
def verify(self, username, password):

    #found the user in the user list
    if username in self.users and self.users[username]['password'] == password:
        if self.users[username] == 'admin':
            return username
        else:
            return username
    else:
        return None
```

Figure 20 condition compare with username.

In this part, the user account is found by entering the username and password for data comparison, and then the username is returned to represent the find.

```
class Result:
    def __init__(self):
        self.data = {
            'User_page':'Welcome this is USER  dashboard',
            'Admin_page' : 'Welcome this is ADMIN dasboard',
            'Not_Found' : 'Access Denied: Not Found the username and password'
        }

    #check username and print result
    def printResullt(self, username):
        if username == 'admin':
            return {self.data['Admin_page']}
        elif username == 'Bob' or username =='Ali':
            return {self.data['User_page']}
        else:
            return {self.data['Not_Found']}
```

Figure 21 condition verify.

In another class method called 'Result', the identity is mainly identified based on the username returned after the search. Since the system is mainly for the purpose of security knowledge, the richness and complexity of the system are reduced, and the fields are displayed directly. In the ‘printResult’ function, compare the above field data and return to the main function.

Exploit inadequate access controls to obtain administrative rights. This attacker may interfere with system operations or alter data without authorization by manipulating crucial system components. To avoid circumstances like this, secure coding techniques like RBAC become essential.

7.3 Session Hijacking

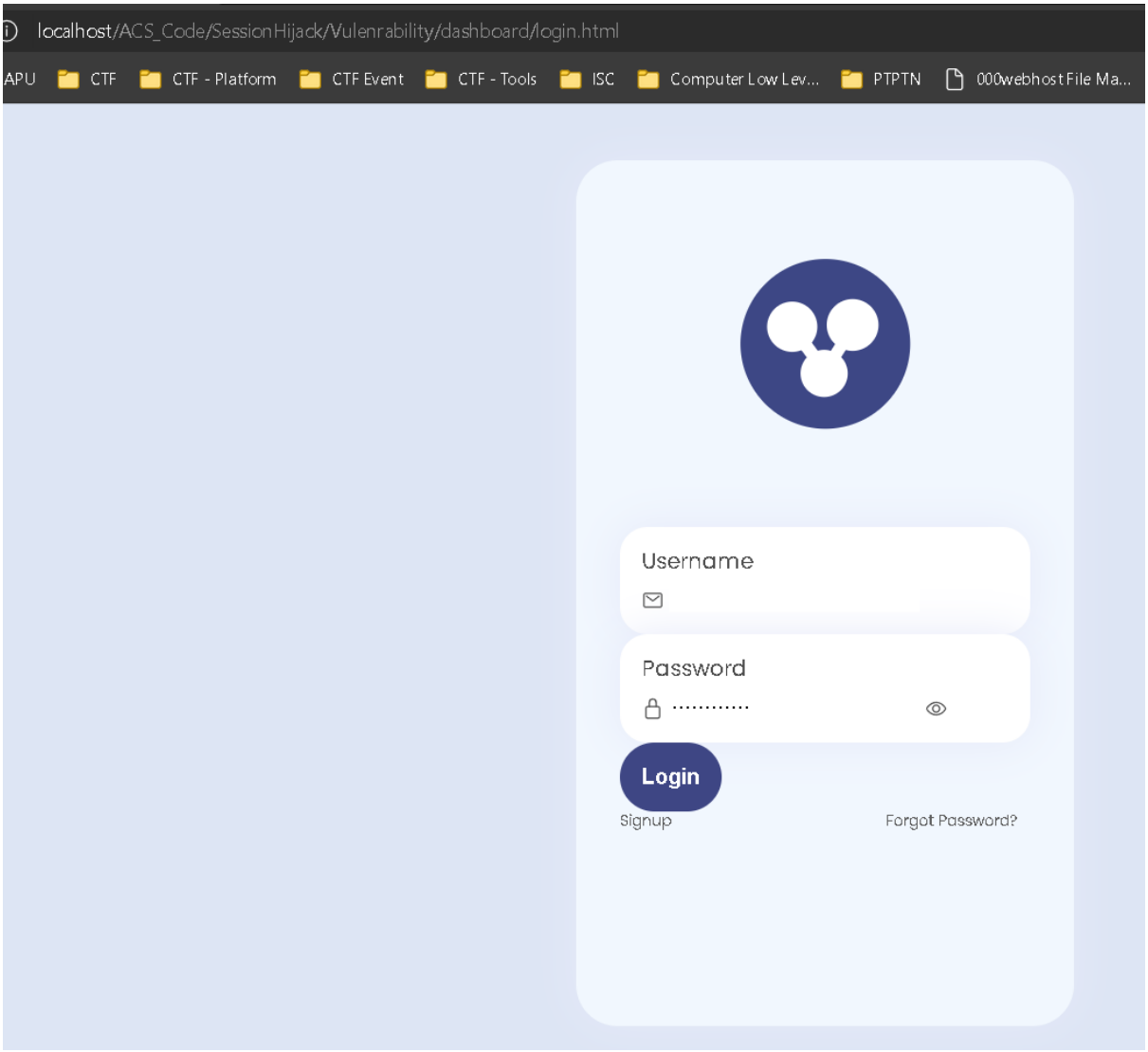

Figure 22 Session Hijacking Login page.

For example, to demonstrate the Session Hijacking vulnerability, the victim needs to log in to the system normally.

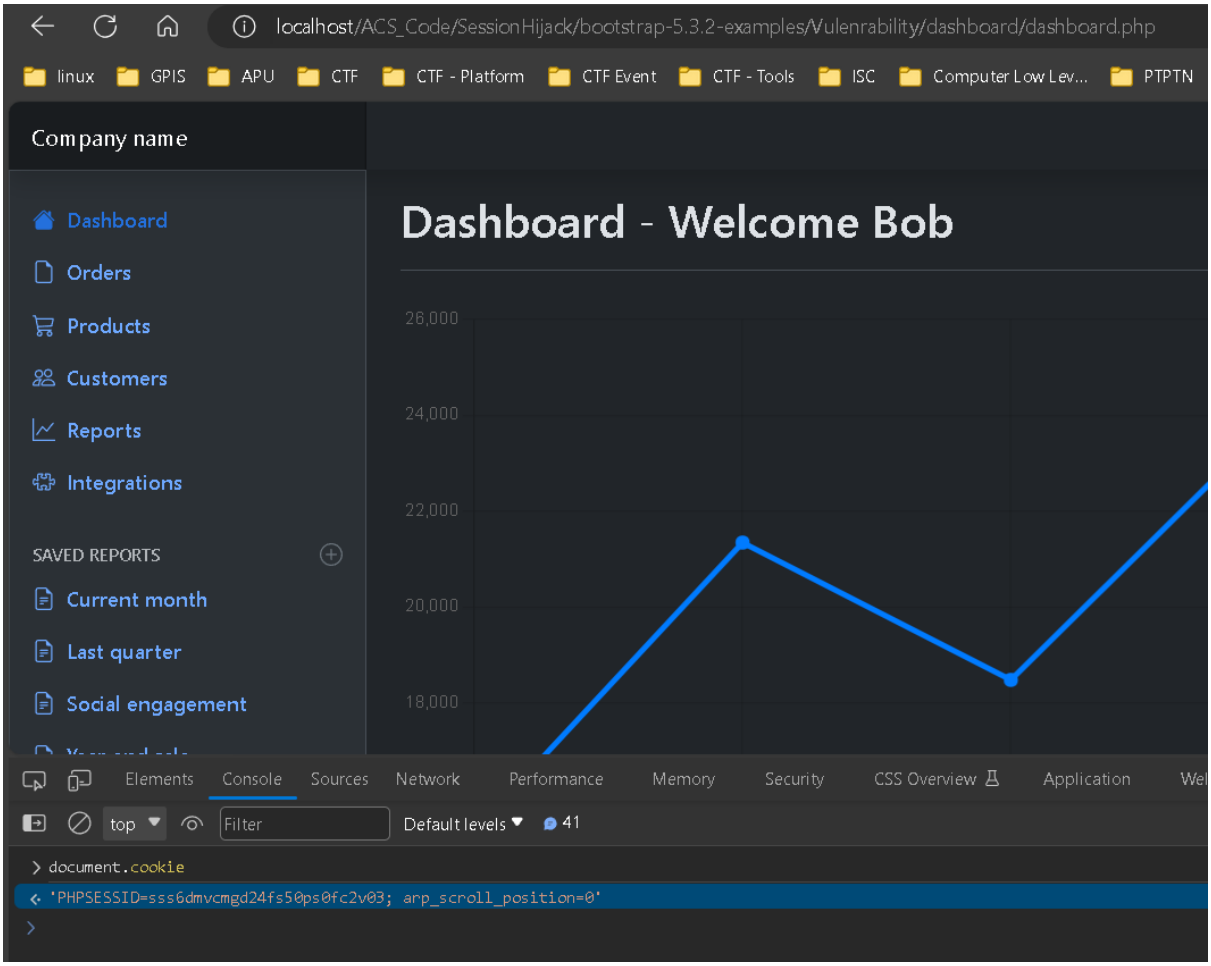

Figure 23 after user login got session / cookie save it details.

After normal login, the server will automatically return a session to the user's browser and store it in a 'cookie'. We can get the cookie value through the 'Inspect' function that comes with the browser. For example, the above chart demonstrates the cookie value from the 'console' display.

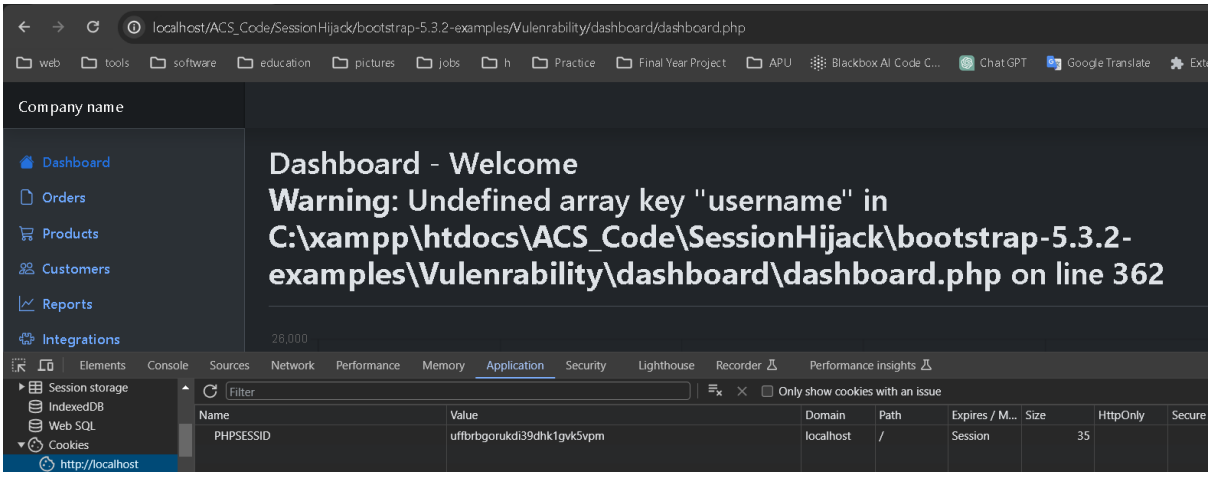

Figure 24 In attacker browser can access page but not info and cookie different.

Assuming that the attacker logs in to the connection on his own browser, an error or '404' page will be displayed, but in this demonstration system we present the vulnerable system in the simplest way.

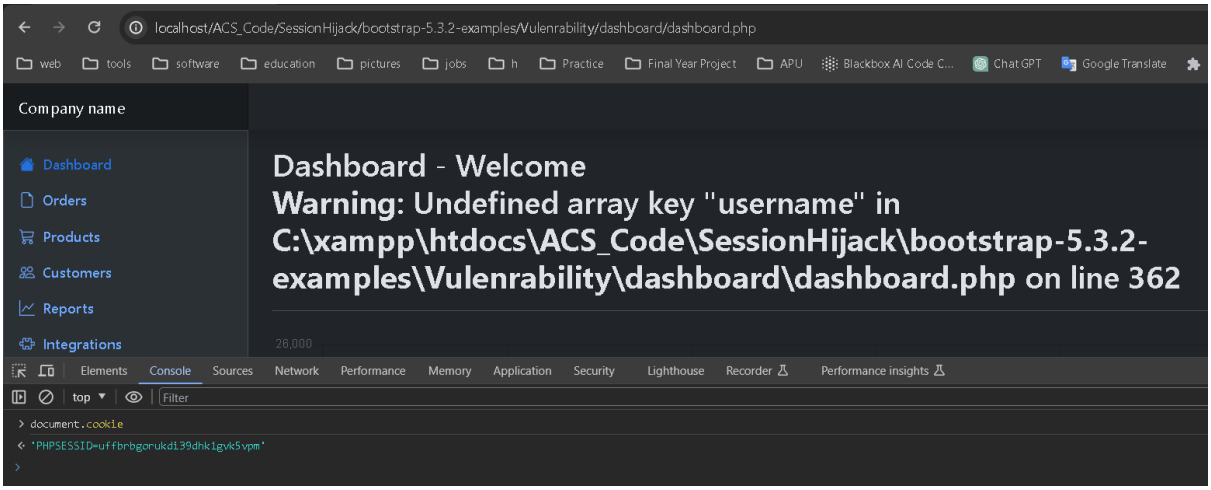

Figure 25 attack browser cookie.

Assuming that the attacker logs in to the connection on his own browser, an error or '404' page will be displayed, but in this demonstration system we present the vulnerable system in the simplest way.

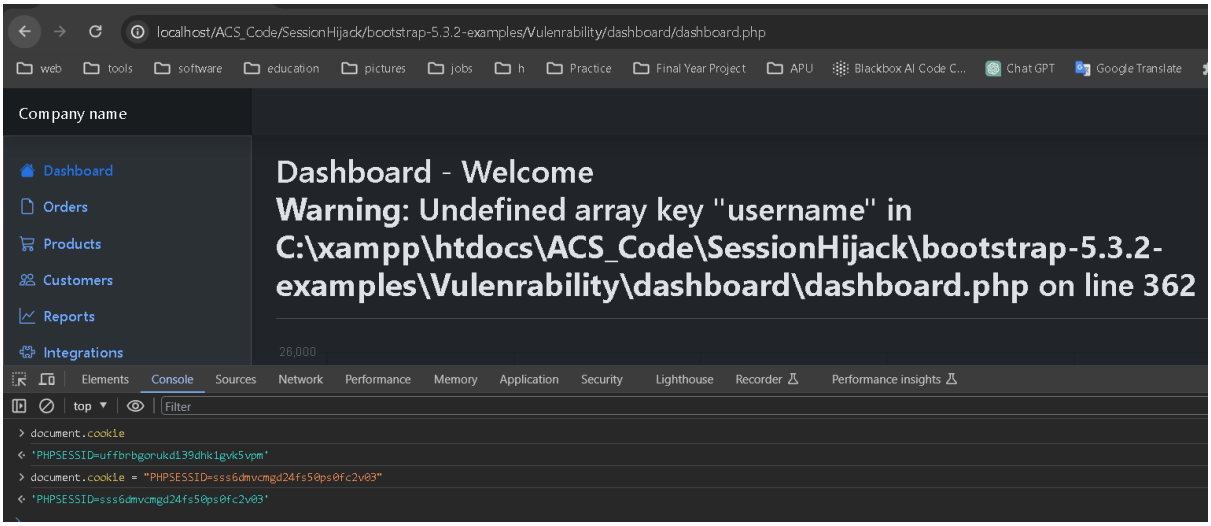

Figure 26 then attack hijack the victim session/ cookie replace itself cookie.

Therefore, the attacker may use various techniques to obtain the session ID logged in by the victim, and this session ID allows the user to log in or access the same account and website by changing the cookie value in another browser or another device.

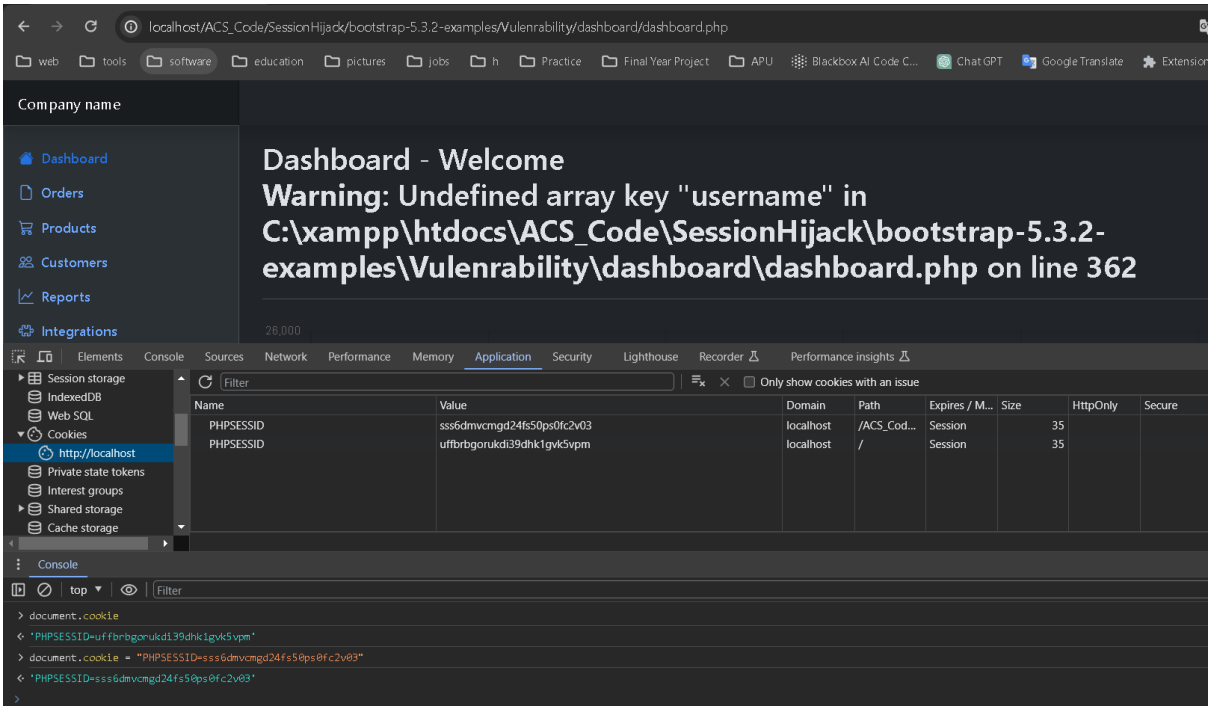

Figure 27 already store inside it.

Therefore, according to the above picture, the attacker obtained the victim's session ID and replaced it with his own browser.

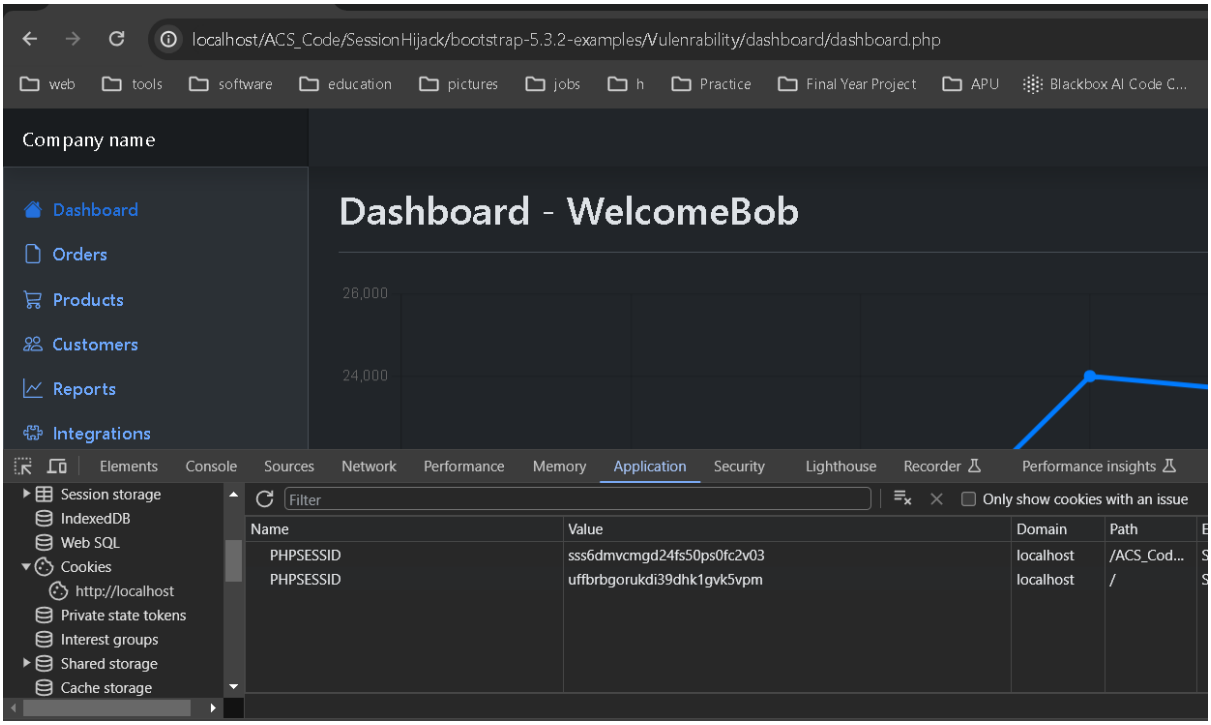

Figure 28 then attacker refresh the browser can access yet.

Once changed, the attacker can access the system or website with victim's account simply by following the new page.

Attackers can easily obtain unauthorised access to user accounts without the necessary security coding defences in place, which could result in identity theft, unauthorised transactions, or manipulation of user data.

8. Conclusion

The assignment will be graded according to the following criteria: thoroughness of the research; precision in vulnerability assessments; clarity of the system design; applicability of secure coding practises; and lucidity of the explanations of the goals and implications of secure coding.

In summary, the task includes a thorough system development process, vulnerability analysis, the deployment of secure code, and an explanation of the consequences of not implementing it. The adoption of a holistic strategy ensures an in-depth

understanding of the importance of security in system development and the part that secure coding performs in achieving strong cybersecurity.

References

cloudflare. (n.d.). *What is SQL injection?* Retrieved from cloudflare: https://www.cloudflare.com/learning/security/threats/sql-injection/

detectify. (2016, Jul 13). *OWASP TOP 10: Missing Function Level Access Control*. Retrieved from detectify: https://blog.detectify.com/best-practices/owasp-top-10-missing-function-level-access-control-7/

dominfosec. (n.d.). *Missing Function Level Access Control: Assessing and Implementing Access Controls*. Retrieved from dominfosec: https://dominfosec.com/our-cases/missing-function-level-access-control-assessing-and-implementing-access-controls/

eccouncil. (2022, March 29). *What Is Session Hijacking, and How Can It Be Prevented?* Retrieved from eccouncil: https://www.eccouncil.org/cybersecurity-exchange/ethical-hacking/how-to-prevent-session-hijacking-attacks/

Johnson, A. (2021, May 5). *Session hijacking: What is a session hijacking and how does it work?* Retrieved from norton: https://us.norton.com/blog/id-theft/session-hijacking

kingthorin. (n.d.). *SQL Injection*. Retrieved from owasp: https://owasp.org/www-community/attacks/SQL_Injection

Madou, M. (2020, May 11). *Coders Conquer Security Infrastructure as Code Series: Missing Function Level Access Control*. Retrieved from securecodewarrior: https://www.securecodewarrior.com/article/coders-conquer-security-infrastructure-as-code-missing-function-level-access-control

owasp. (n.d.). *Session hijacking attack*. Retrieved from owasp: https://owasp.org/www-community/attacks/Session_hijacking_attack

portswigger. (n.d.). *SQL injection*. Retrieved from portswigger: https://portswigger.net/web-security/sql-injection