Web Application

Vulnerability Assessment

and

Penetration Testing Report

For

Mutillidae

From Rahul Gupta

Assessment Date:25-12-2021

Table Of Contents

1 INJECTION…………………………………………………………………………………………4

* 1. SQl Injection……………………………………………………………………………………....5
  2. Bypass Authentication …………………………………………………………………………....5

1.2.SQL MAP………………………………………………………………………………………….7

1.3.HTML Injection……………………………………………………………………………………12

1.4. COMMAND Injection…………………………………………………………………………....13

2 BROKEN AUTHENTICATION AND SESSSION MANAGEMENT…………………………….16

2.1 Authentication Bypass (Brute force) ……………………………………………………………..16

3 CROSS SITE SCRIPTING …………………………………………………………………………20

3.1 Cross site scripting in DNS lookup (Reflected)……………………………………………………25

4 INSECURE DIRECT OBJECT REFERENCES ………………………………………………,.,,…23

4.1 IDOR in Text File Viewer …………………………………………………………………………23

4.2 IDNOR in Source Viewer …………………………………………………………………………25

5 SECUIRTY MISCONFIGURATION ………………………………………………………………28

5.1 Unrestricted File Upload…………………………………………………………………………...28

6 SENSITIVE DATA EXPOSURE …………………………………………..………………………33

6.1 Robots.txt…………………………………………………………………………………………..33

6.2 Passwords………………………………………………………………………………………….34

6.3 Click jacking ………………………………………………………………………………………35

7 MISSING FUNCTION LEVEL ACCESS CONTROL ……………………………………………

8 CROSS SITE REQUEST FORGERY ……………………………………………………..……….38

8.1 Register User……………………………………………………………………………………….38

9 USING COMPONENTS WITH KNOWN VULNERABILITY ……………………………….…..41

10 UNVALIDATED REDIRECTS AND FORWARDS ……………………………………………..43

**Technical Details**

**1. 1 SQL Injection**

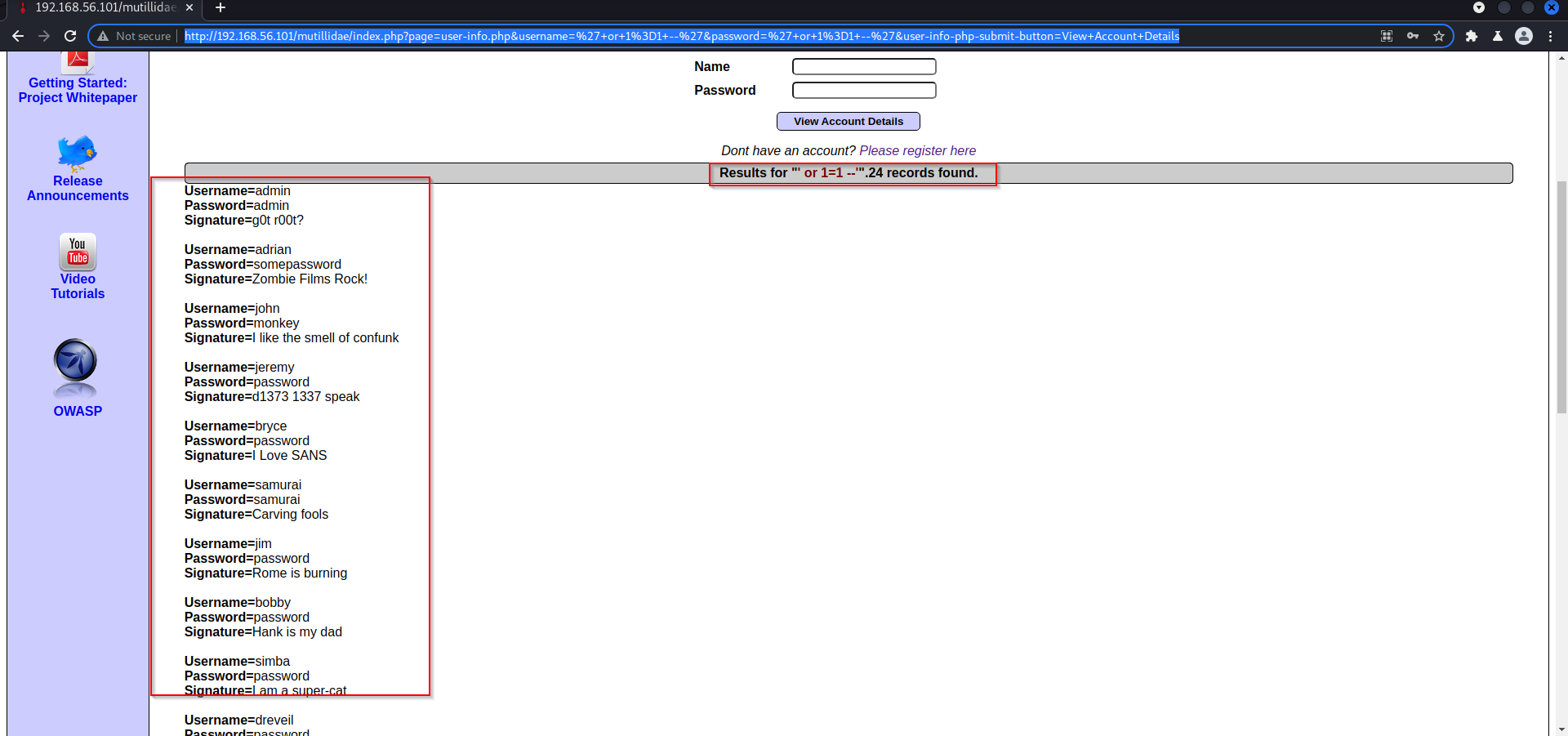
Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization

**Affected URL**: http://192.168.56.101/mutillidae/index.php?page=user-info.php&username=%27&password=&user-info-php-submit-button=View+Account+Details

**Description**: we passed ‘ in Name Textbox can we got SQL error this means this application is vulnerable to SQL INJECTION

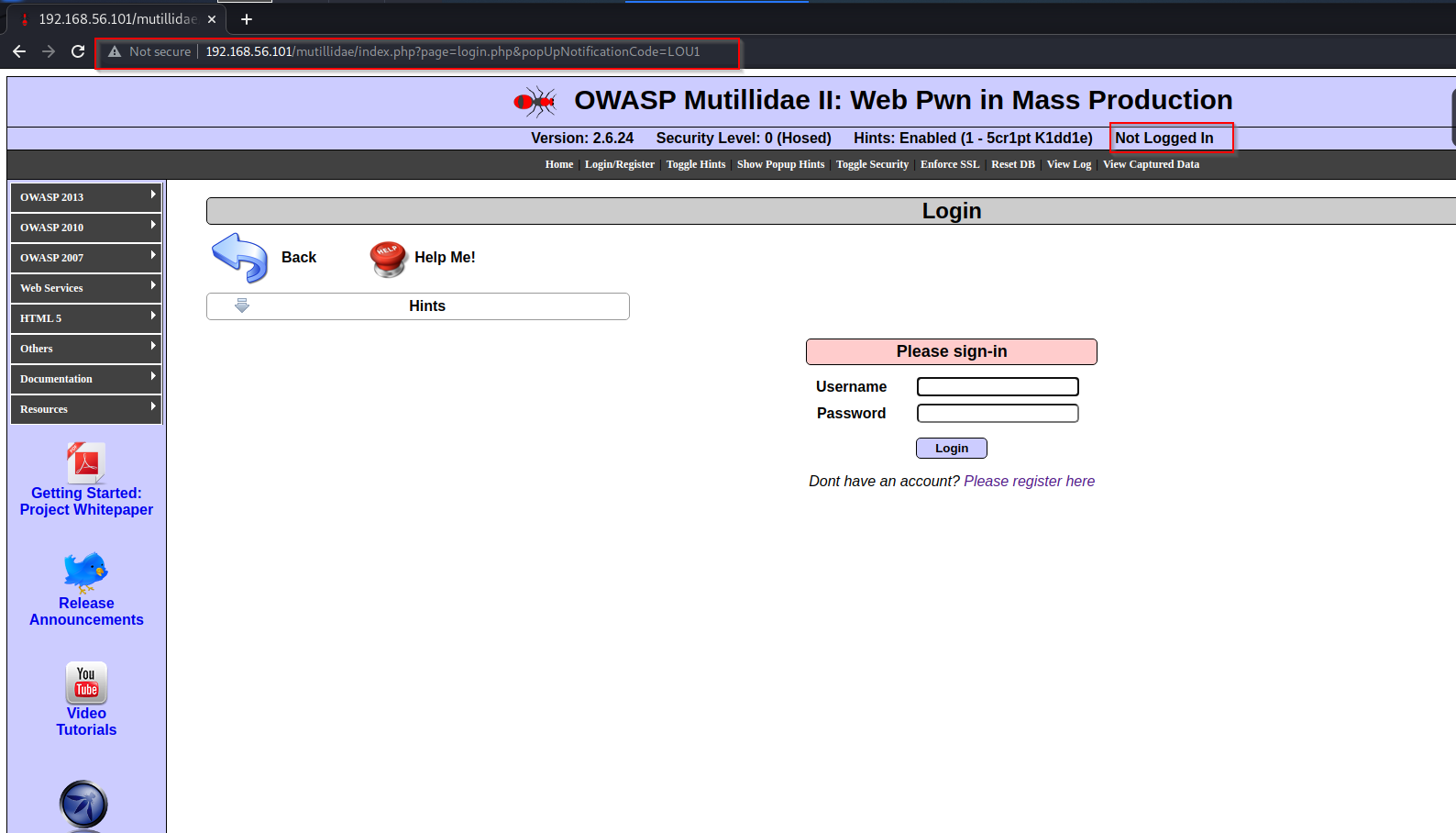
Fig:1

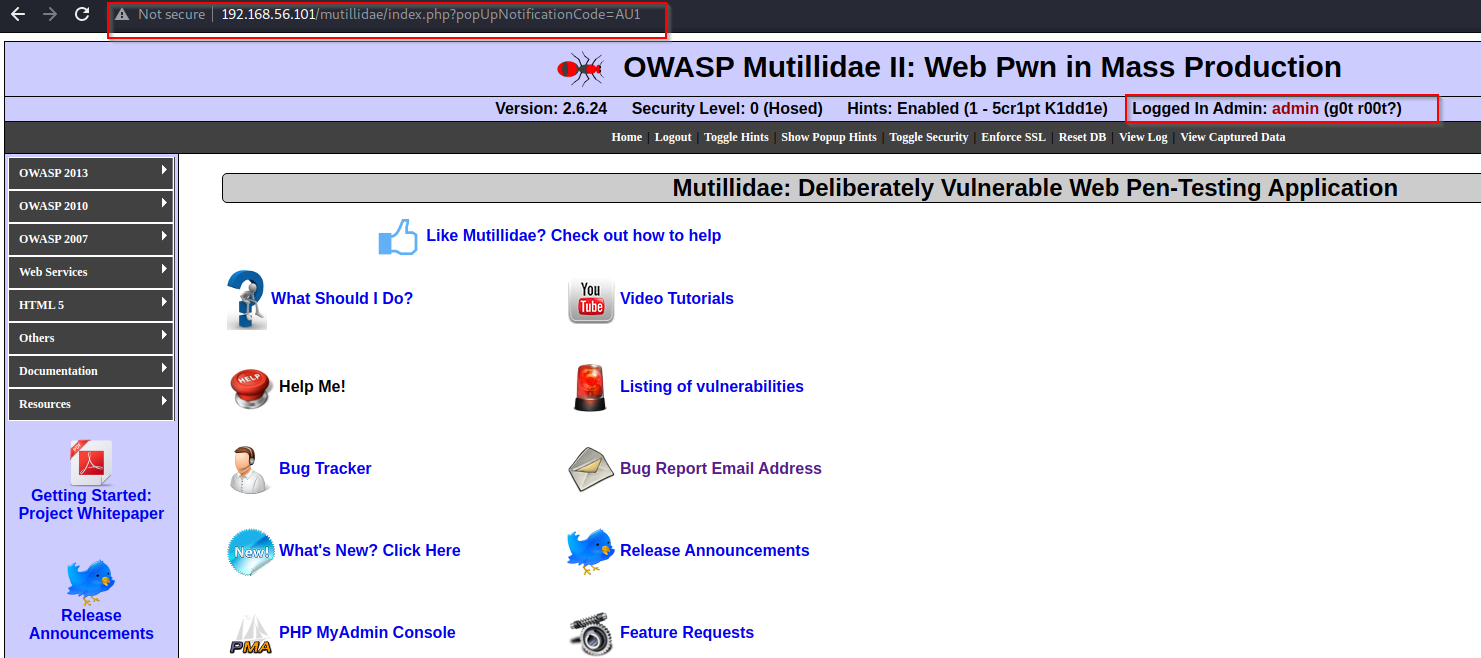

**USER INFO (SQL INJECTION)**

Description: we used **“ ‘ or 1=1—’** “ in Name and Password Form , as a result we got user names and password of all the users

**1.2 BYPASS AUTHENTICATION**

**Affected URL**: http://192.168.56.101/mutillidae/index.php?page=login.php&popUpNotificationCode=LOU1

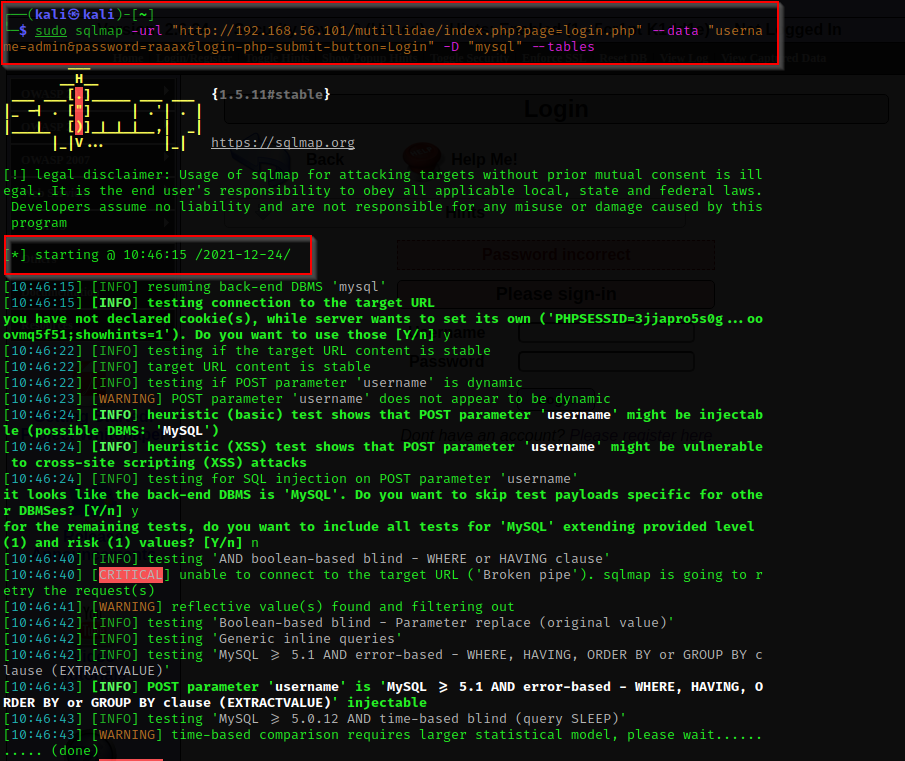
**Description**: we easily got root user account by input of “ ‘ or 1=1—’ “ .

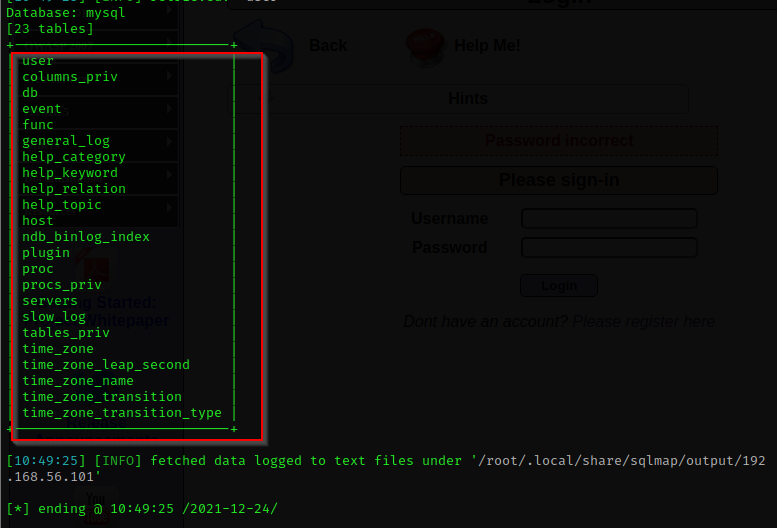


**1.3 SQL Injection using SQLMAP**

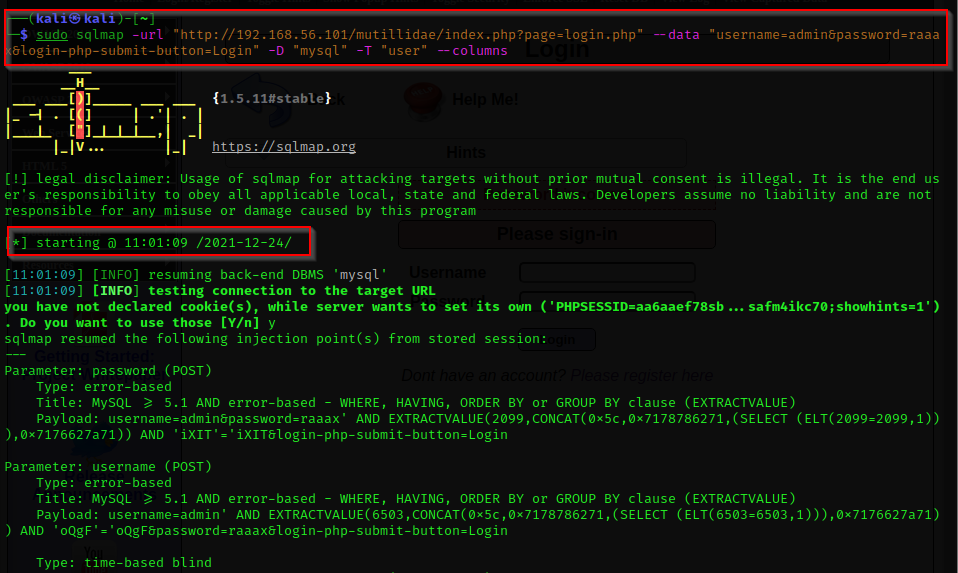
**URL**: <http://192.168.56.101/mutillidae/index.php?page=login.php>

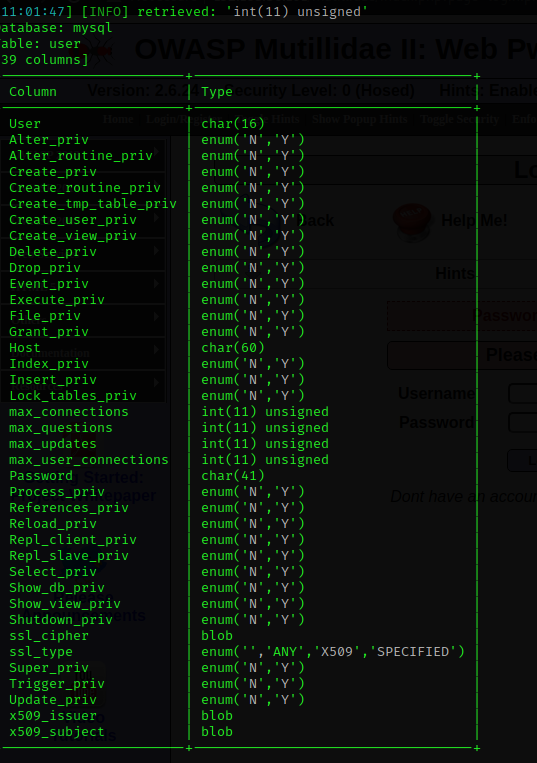
**Description**: server is vulnerable to SQl injection with the help of sqlmap , we successfully retrieve aal the details of database

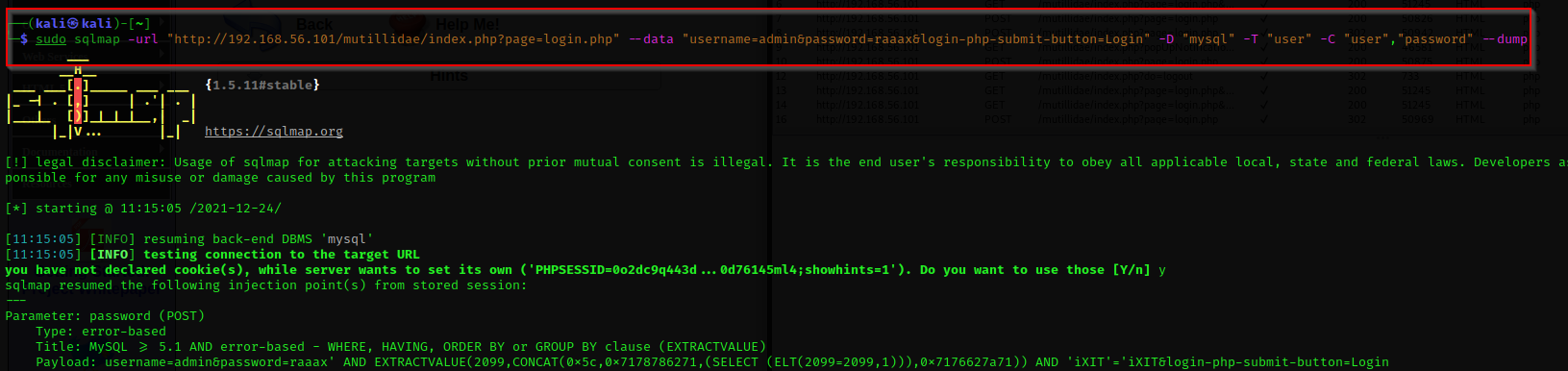
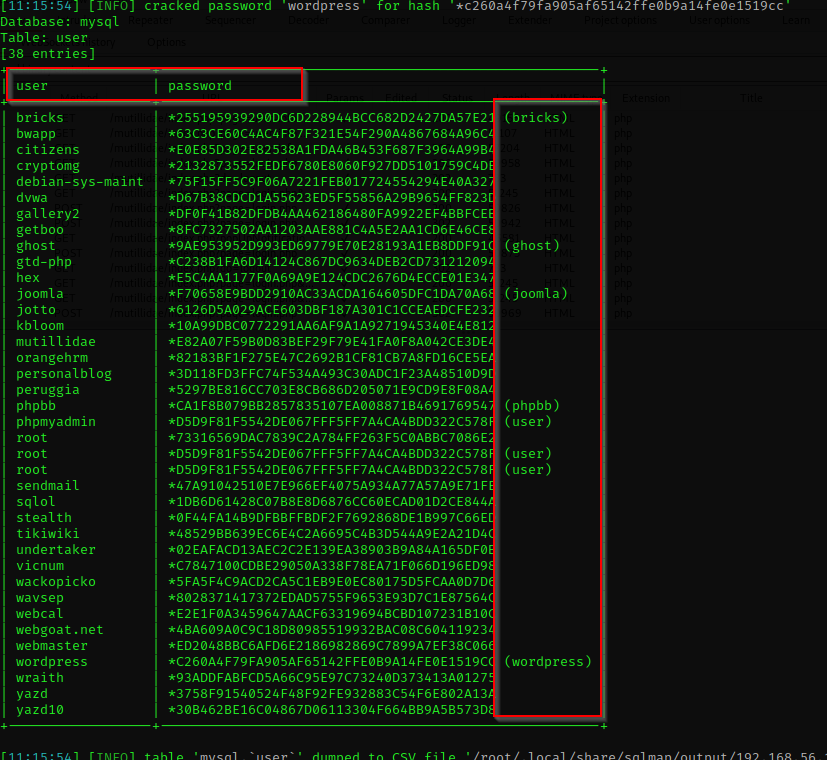




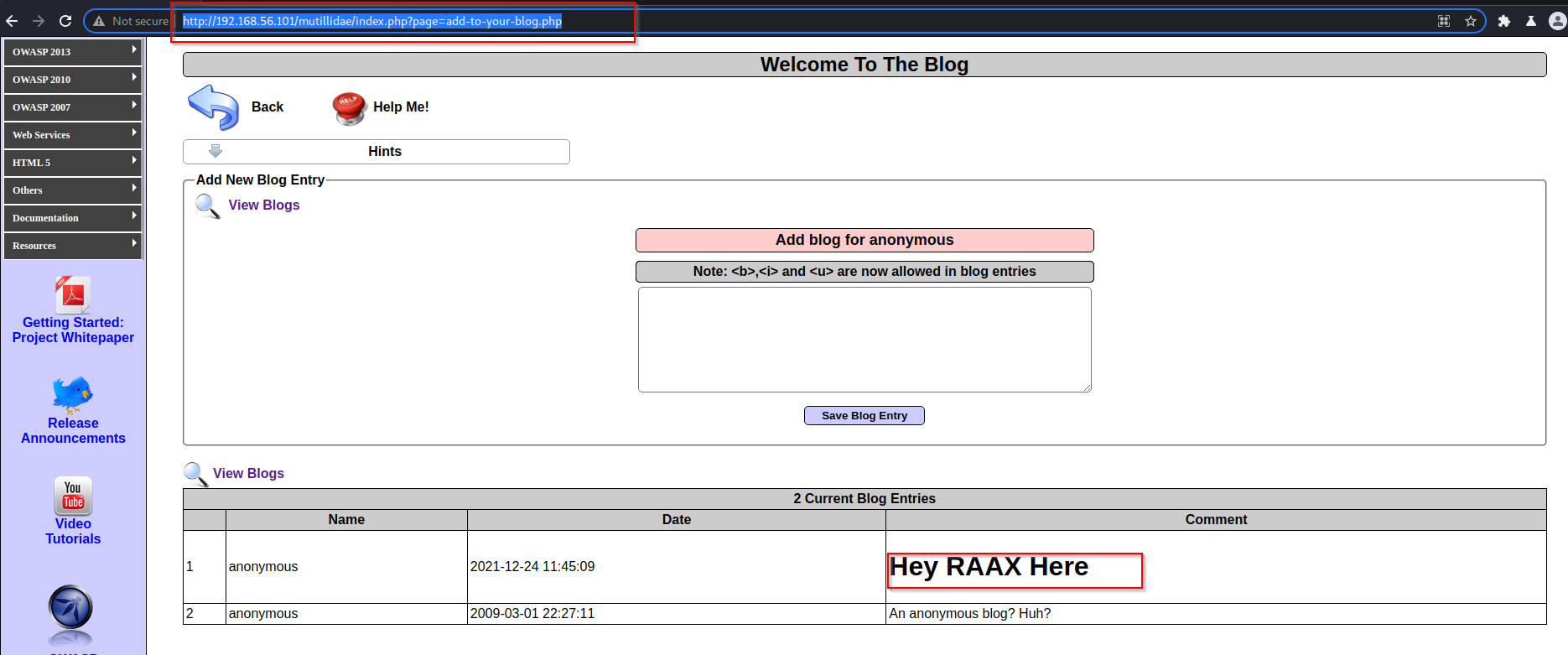
We got all the tables present in Database

We also got all the columns name present in database



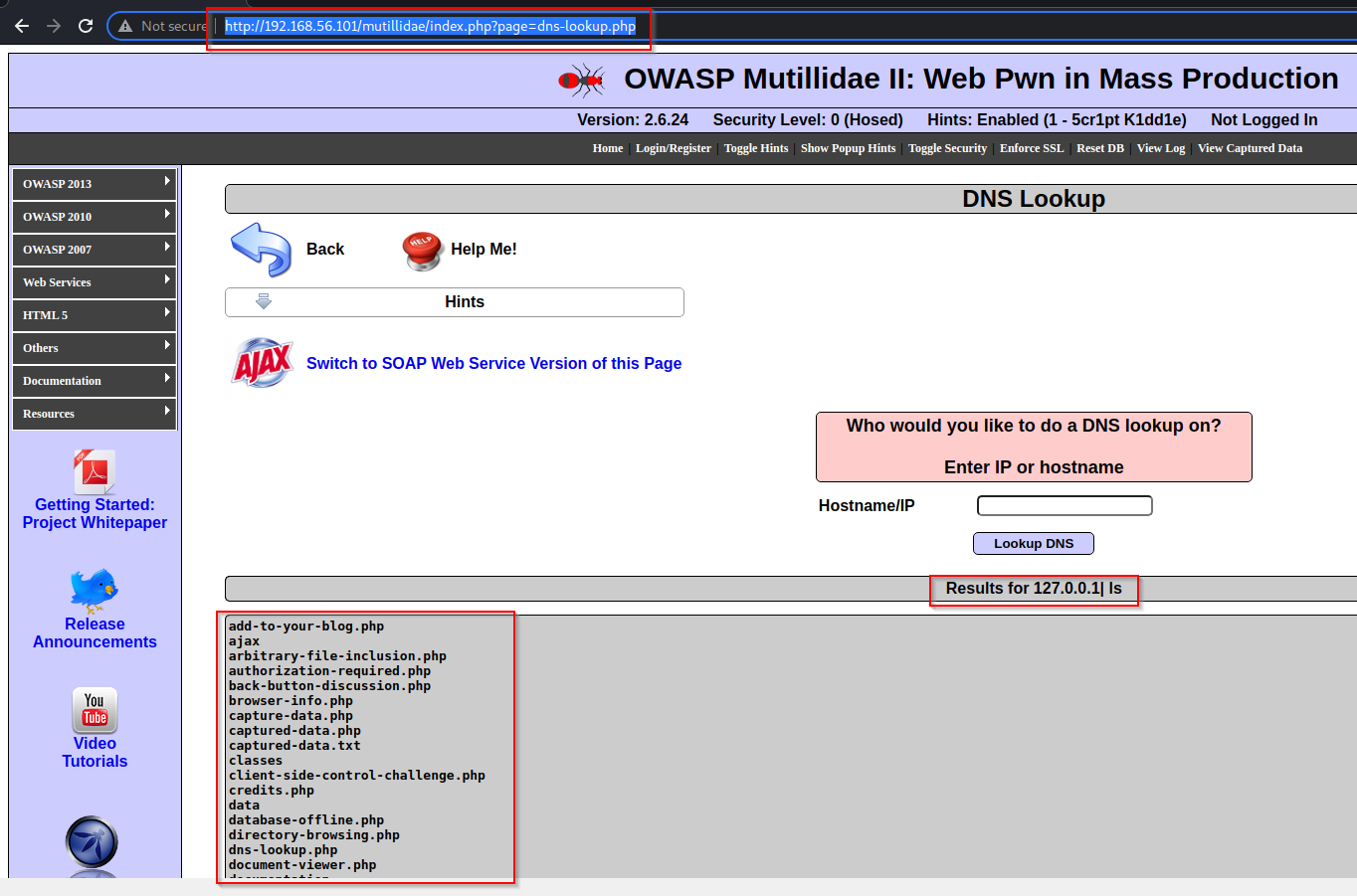
We have got all Usernames and their Password

**1.4 HTML INJECTION**

**Description:** we are able to inject arbitrary HTML code into web page

**1.5 Command Injection**

url: <http://192.168.56.101/mutillidae/index.php?page=dns-lookup.php>

Description: we were able to do execution of arbitrary commands on the host operating system

**Impact:**

1. Injection can result in data loss, corruption, or disclosure to unauthorized parties, loss of accountability, or denial of access. Injection can sometimes lead to complete host takeover.

2. In general, injection vulnerabilities will have a decent impact however and could lead to data loss, remote code execution and authentication bypasses

**Prevention:**

1. For any residual dynamic queries, escape special characters using the specific escape syntax for that interpreter

2. Least privilege: Do not assign DBA or admin type access right to you application accounts

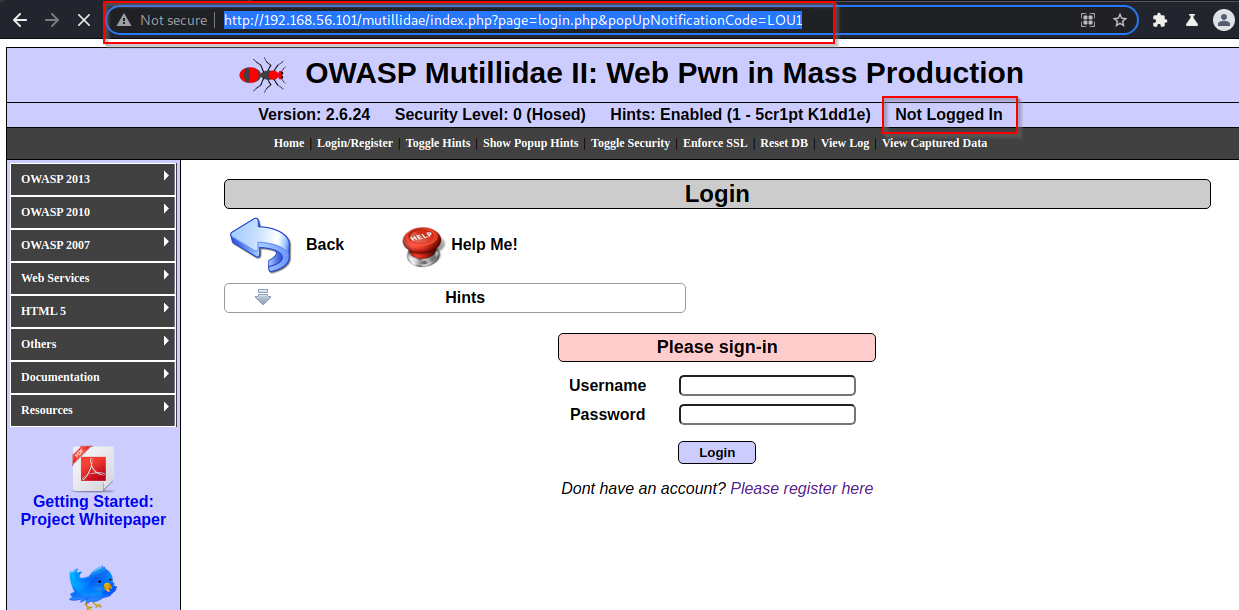
3. use of Parameterized query: parameterized query is a query in which placeholders are used for parameters and the parameter values are supplied at execution time. The most important reason to use parameterized queries is to avoid SQL injection attacks

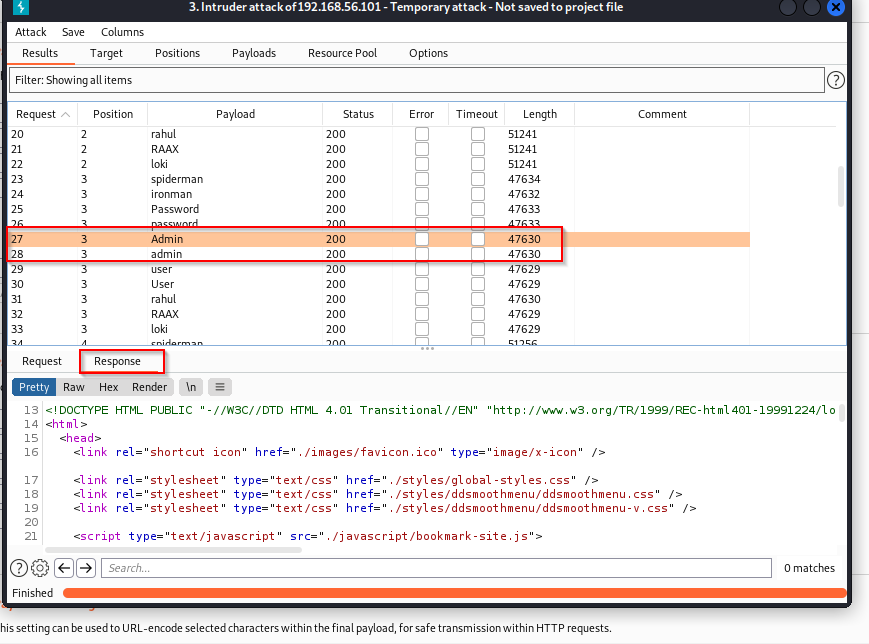
**A2 Broken Authentication**

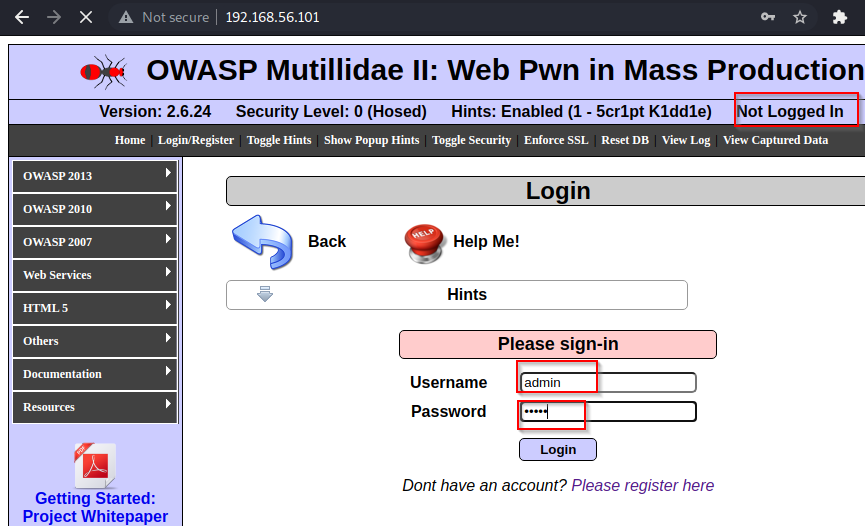
**2 Brute force attack on login page**

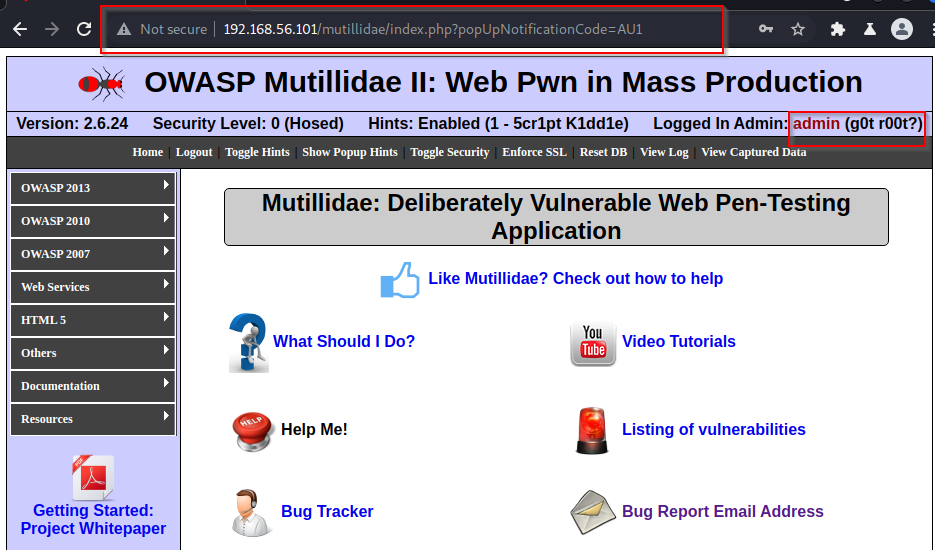
Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users’ identities.

**URL**: <http://192.168.56.101/mutillidae/index.php?page=login.php&popUpNotificationCode=LOU1>

Description: With the use of Brute force Technic we were able to bypass the login form







**Impact:**

1. Attackers have to gain access to only a few accounts, or just one admin account to compromise the system.
2. Depending on the domain of the application, this may allow money laundering, social security fraud, and identity theft, or disclose legally protected highly sensitive information.
3. Getting single admin account is enough for the attacker to comprise web application
4. Broken Authentication can led to data breaches , leakage of sensitive information, identity theft

**Preventions:**

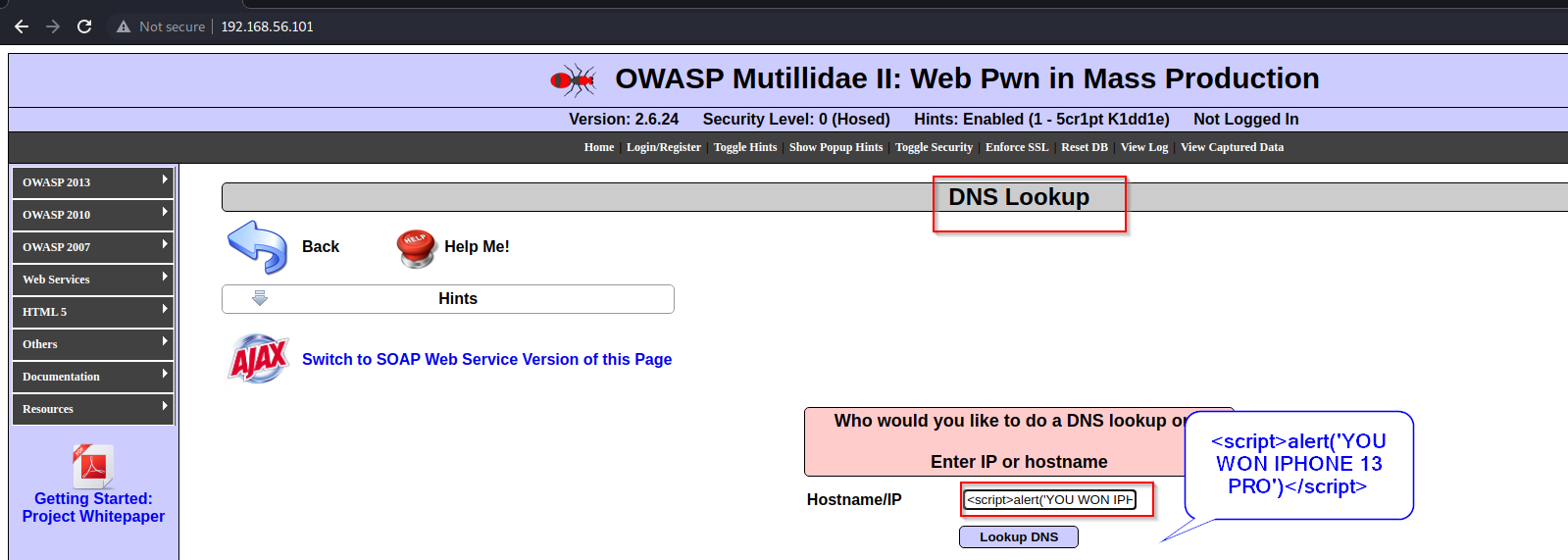
1. Enable password complexity for users
2. Use weak-password checks by forcing users to include a mix of small letters, capital letters, alphanumeric symbols, and special characters while creating passwords.
3. Limit to failed login attempts. log all failures and alert administrators when credential stuffing
4. Enable multi-factor authentication to prevent automated credential stuffing , brute force and stolen credential re-sue attacks
5. Generating new random session IDs with high entropy after login protects against hackers. Remember, those session IDs should not be present in the URL and invalidated after logout

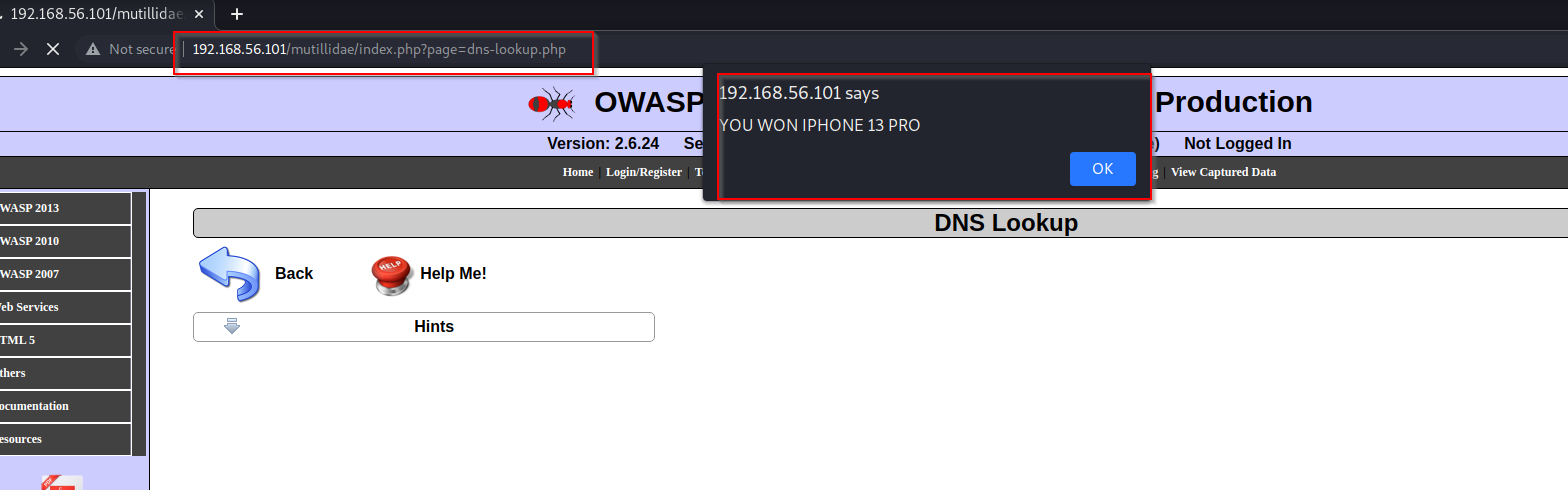
**3 Cross Site Scripting**

XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim’s browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites

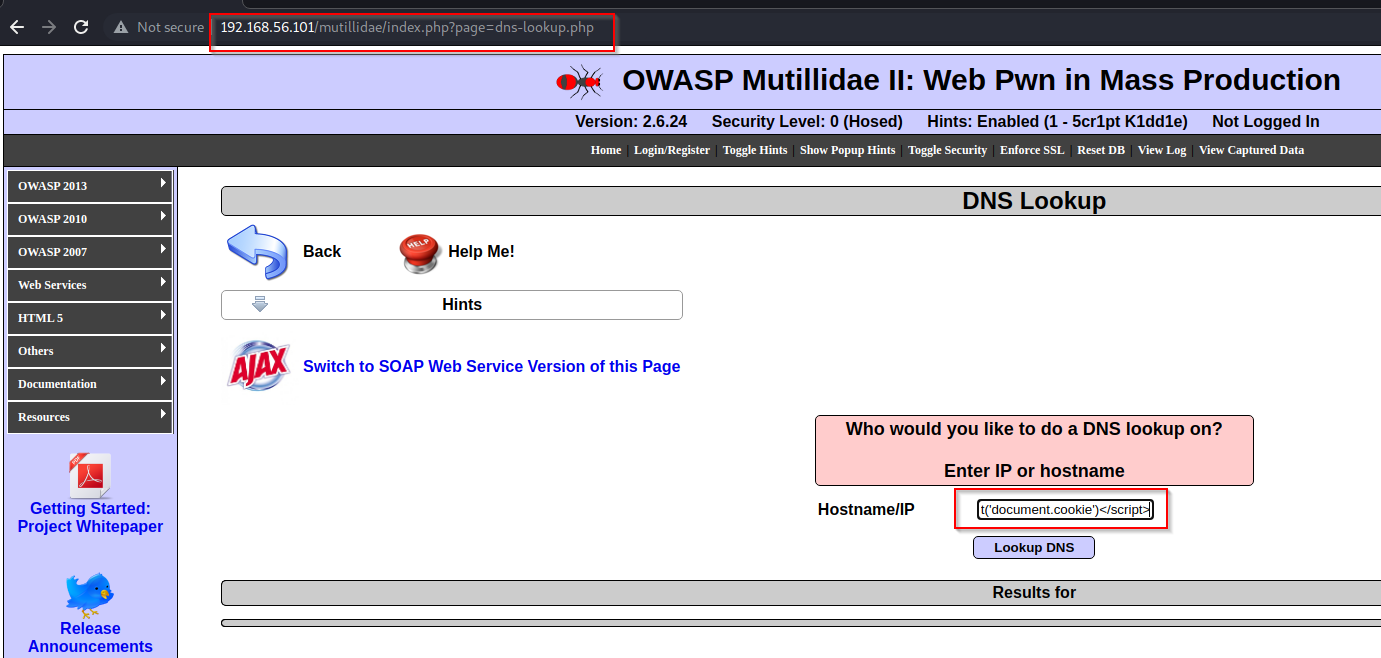
**3.1 Cross site scripting in DNS lookup (Reflected)**

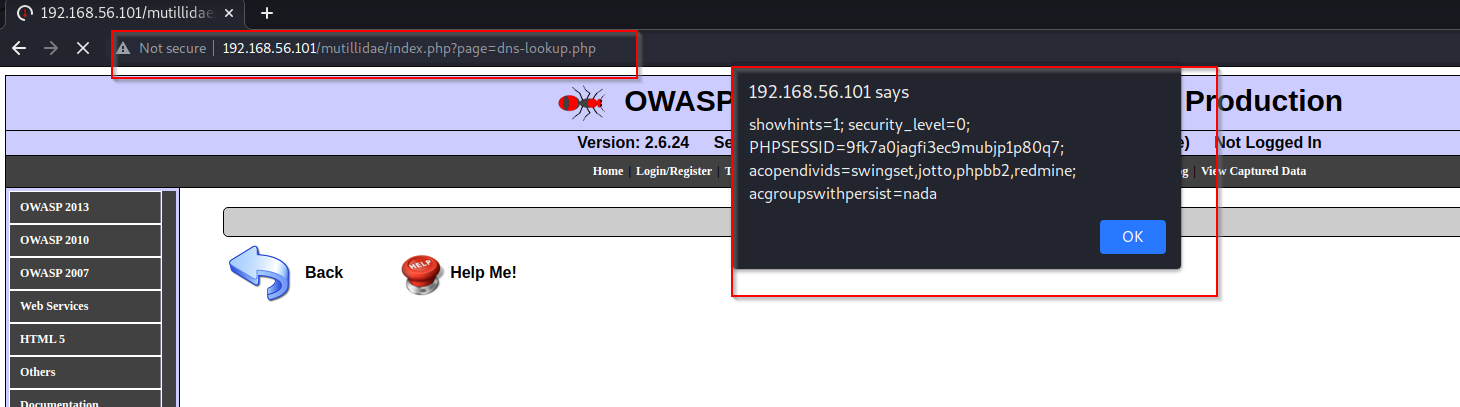
Affected URL: <http://192.168.56.101/mutillidae/index.php?page=dns-lookup.php>

Description: we used script in DNS Lookup box and we got output , this Url is Vulnerable to XSS



**Code**: 192.168.56.102 | <script> alert (‘YOU WON IPHONE 13 PRO’ )</script>

Cross site scripting to get cookies



**Code**: 192.168.56.102 | <script> alert(‘document.cookie’)</script>

**Impact:**

1. Code injected into a vulnerable application can exfiltrate data or install malware on the user’s machine.
2. Attackers can masquerade as authorized users via session cookies, allowing them to perform any action allowed by the user account
3. Attackers seizing online accounts and impersonating users
4. Redirect of web pages to harmful locations

**Prevention:**

1. Escaping untrusted HTTP request data based on the context in the HTML output (body, attribute, JavaScript, CSS, or URL) will resolve Reflected and Stored XSS vulnerabilities
2. **Sanitize user input**: validate to catch potential malicious user-provided input
3. Encode output to prevent potentially malicious user-provided data from triggering automatic load and execute behavior by a browser
4. **User HTTPOnly cookie Flag**: Set the HTTPOnly flag on session cookies, and any custom cookies that are not accessed by any of your JavaScript code
5. **X-xss protection Header**: The HTTP X-XSS-Protection header is a feature available in popular browsers like Google Chrome and Internet Explorer, which filters suspicious content to prevent reflected XSS attacks. If the header detects XSS, it blocks the page from loading, but doesn’t sanitize inputs in the page.

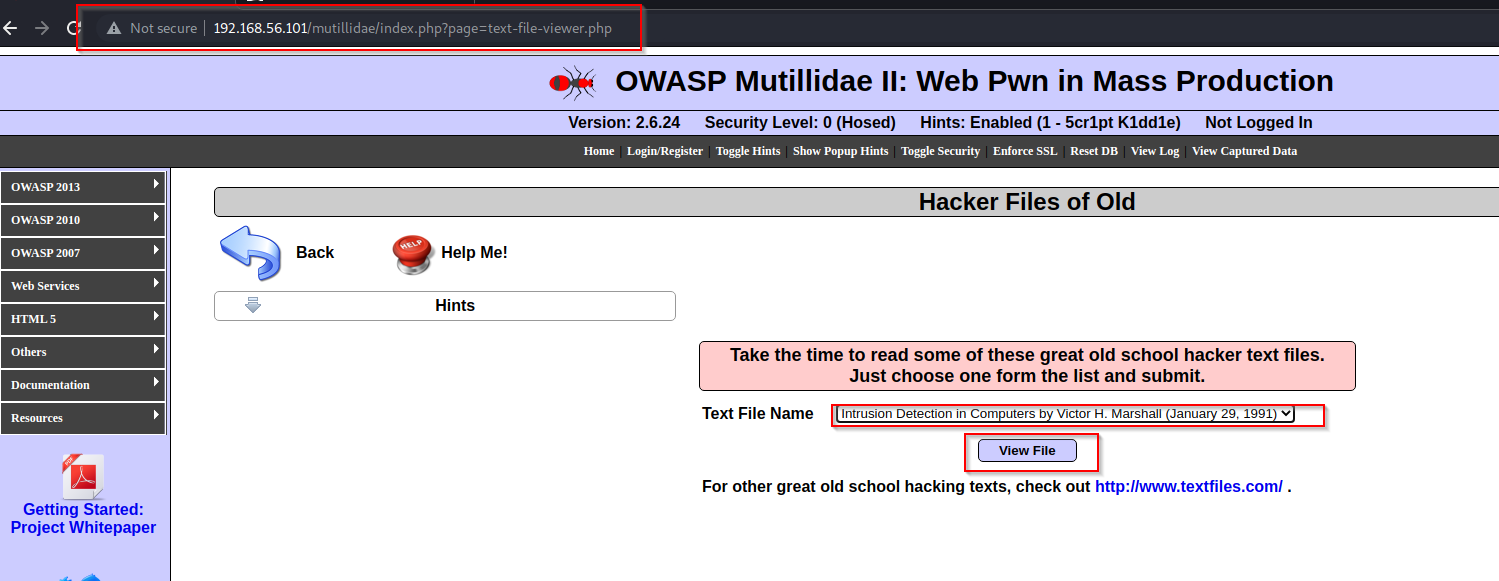
**4 Insecure Direct Object References (IDOR)**

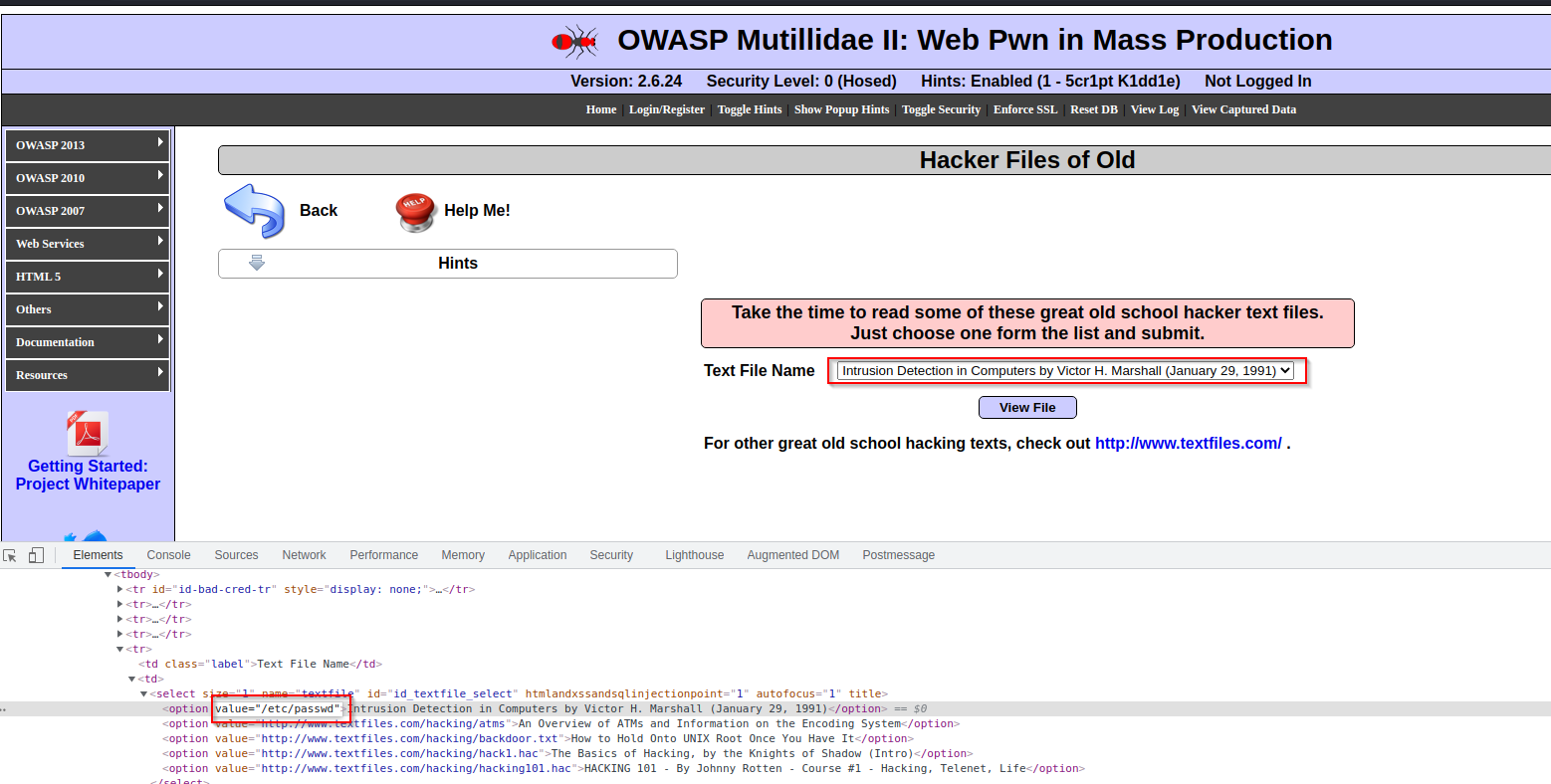
A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

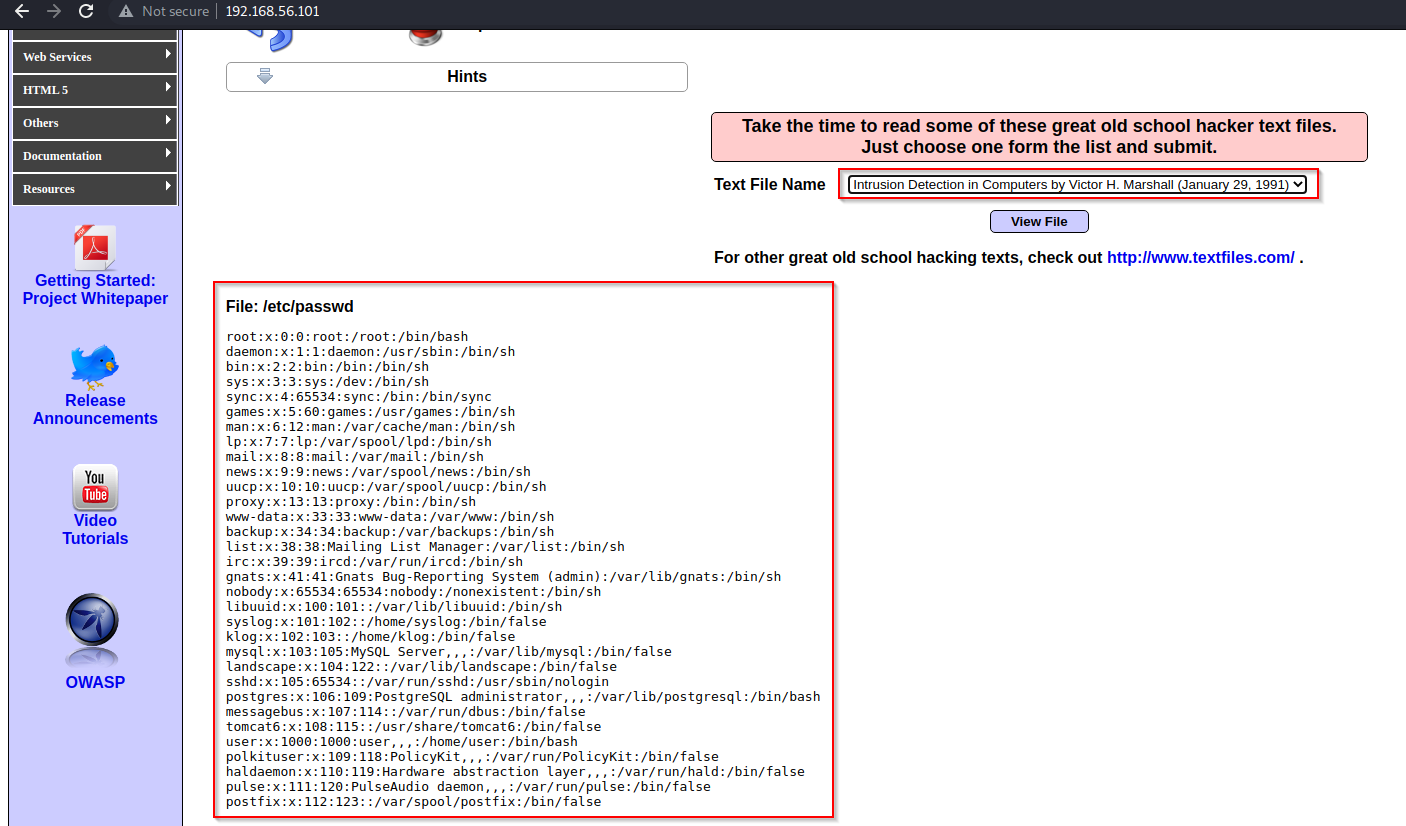
**4.1 IDOR in Text File Viewer**

URL: <http://192.168.56.101/mutillidae/index.php?page=text-file-viewer.php>

Description: we were able to manipulate unauthorized data and gain access to files. directory



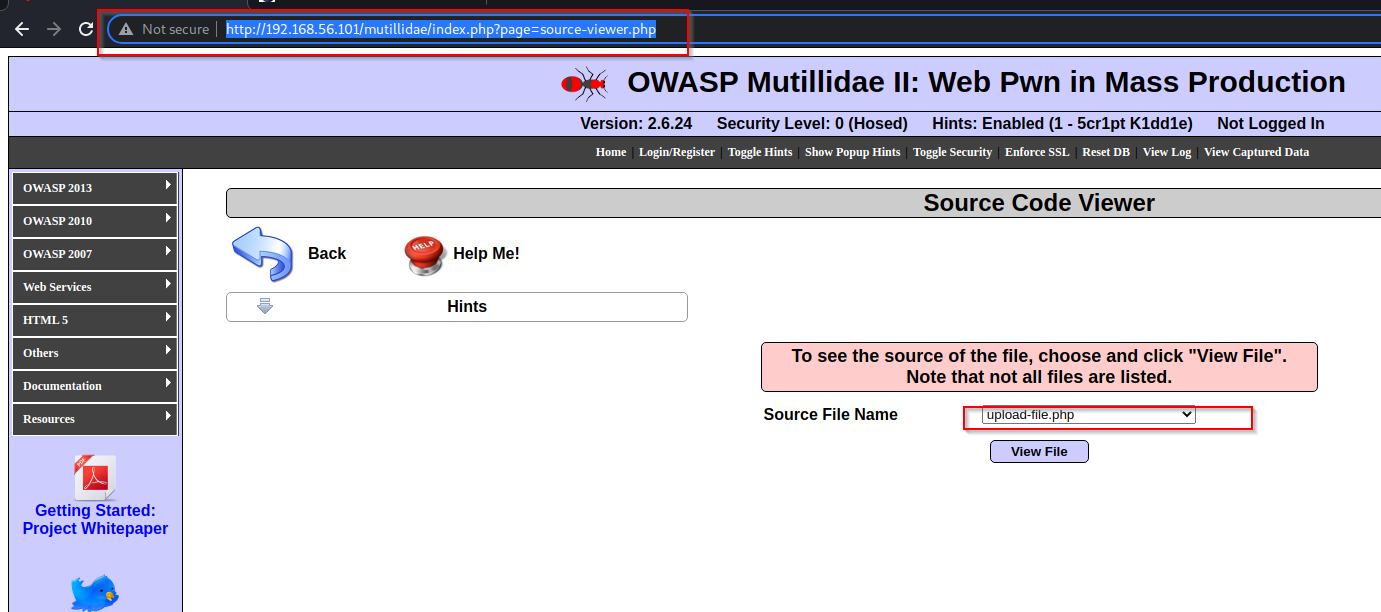


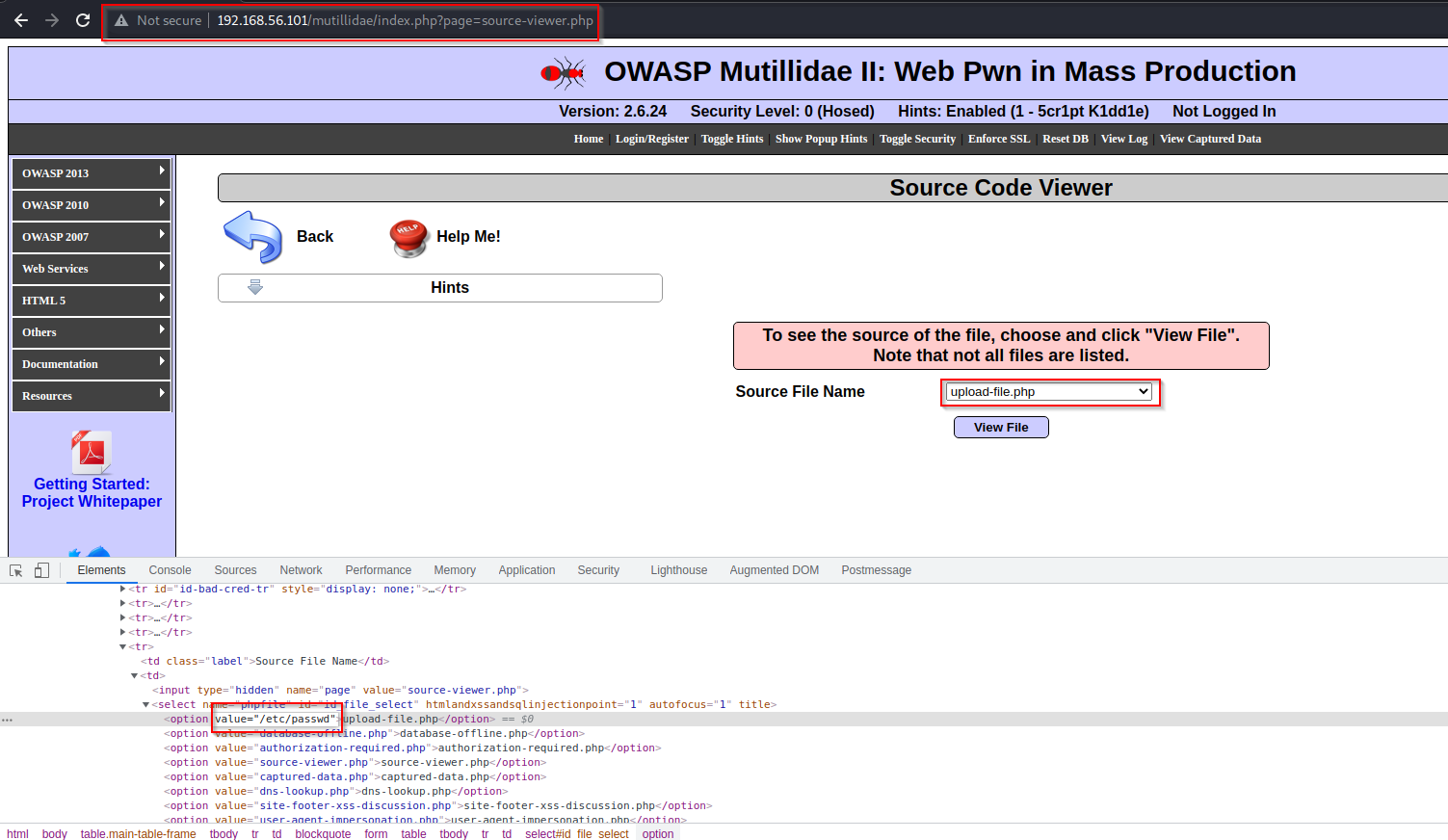


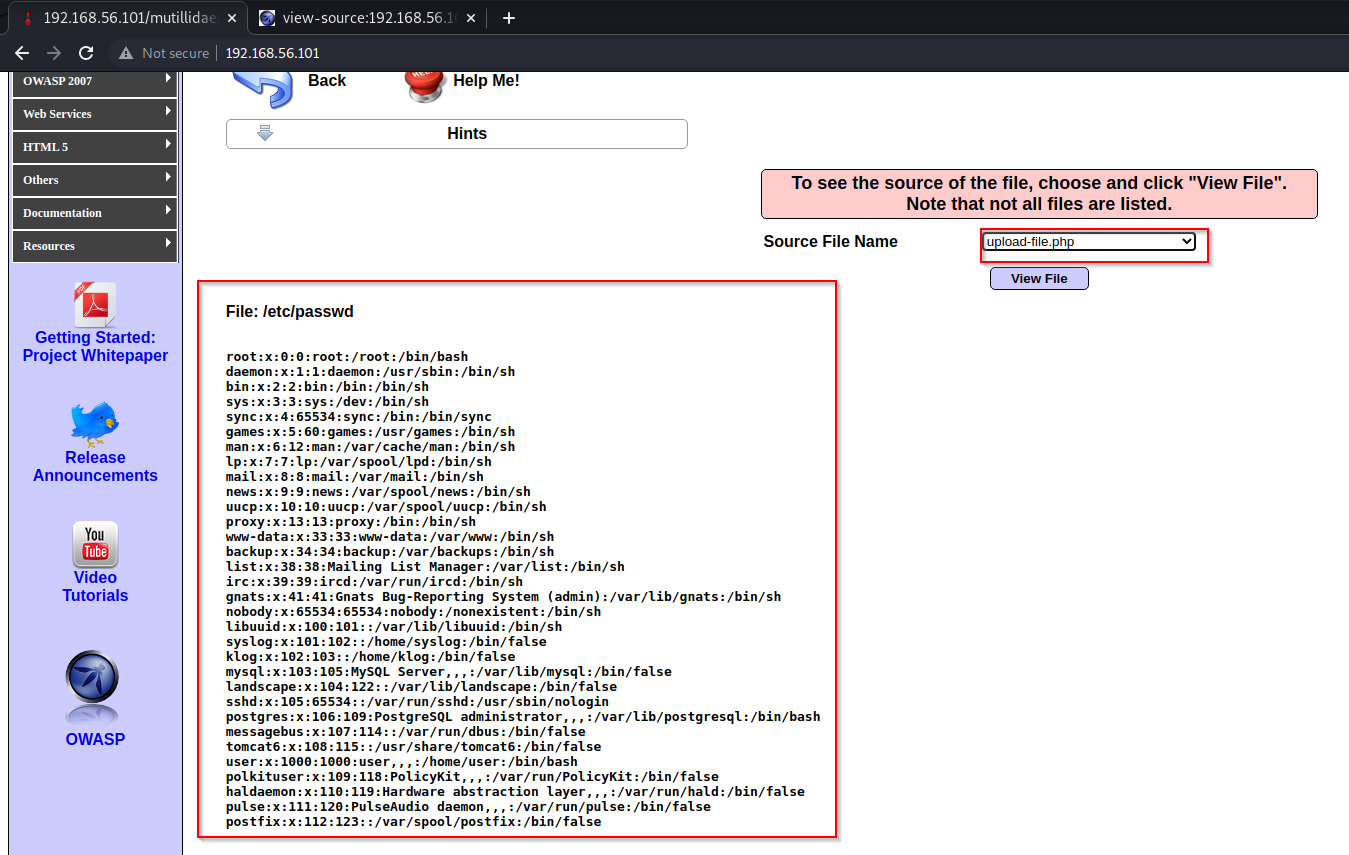
**Code**: /etc/passwd

**4.2 IDOR in Source Viewer**

url: <http://192.168.56.101/mutillidae/index.php?page=source-viewer.php>







Code : /etc/passwd

**Impact:**

1. The application reveals a direct reference to an internal resource or operation.
2. The user is able to manipulate a URL or form parameter to modify the direct reference.
3. The application grants access to the internal object without checking if the user is authorized.
4. Getting direct access to files: Typically combined with path traversal, this type of IDOR lets attackers manipulate file system resources. This could allow them to upload files, manipulate other users’ data, or download paid content for free

**Prevention**

1. Tokens should be generated in such a way that it should only be mapped to the user and should not be public
2. Developers should avoid displaying private object references such as keys or file names.
3. Validation of Parameters should be properly implemented.
4. Verification of all the Referenced objects should be done

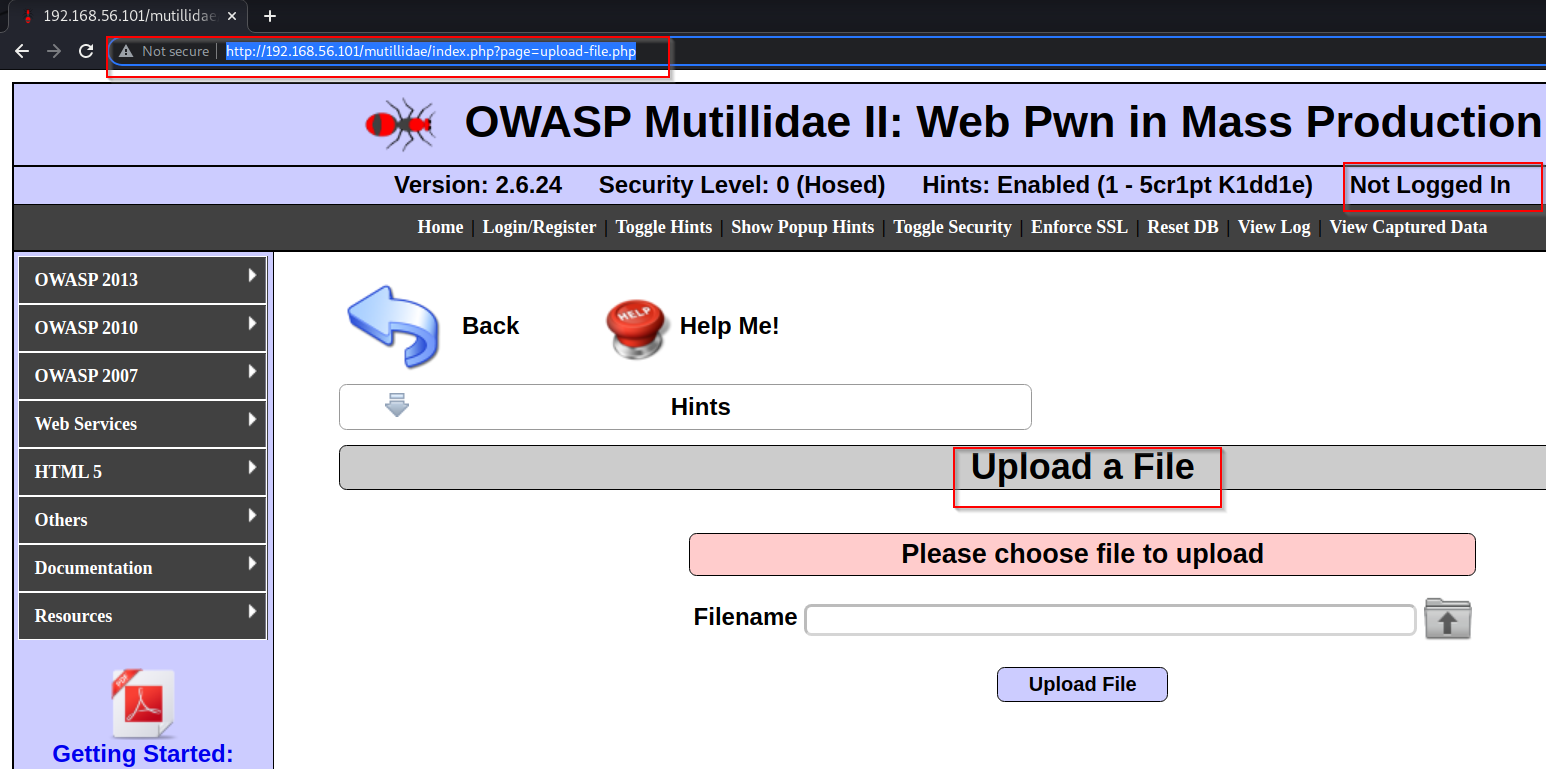
**5 SECURITY MISCONFIGURATION**

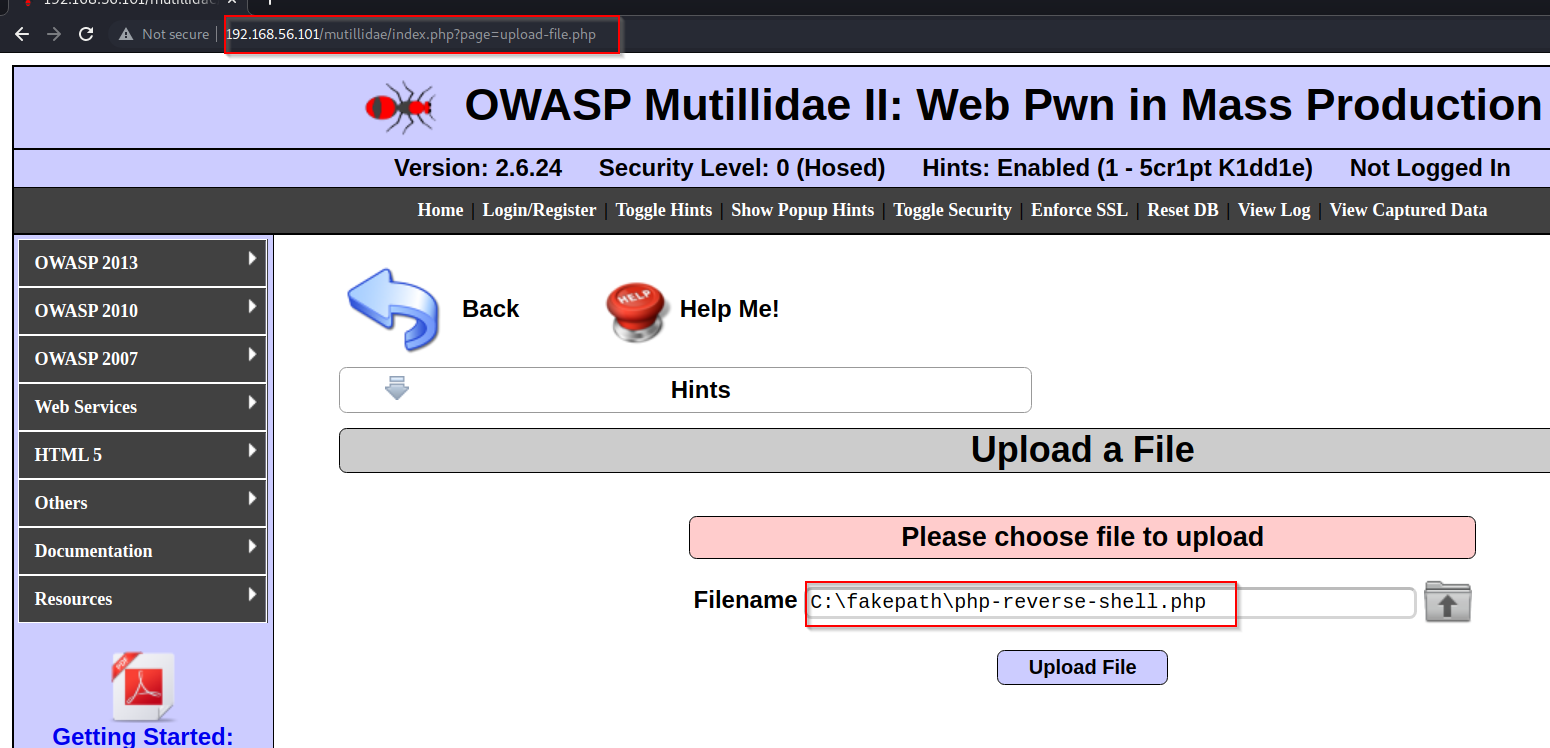
Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date

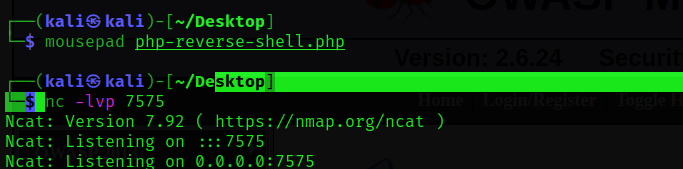
**5.1 Unrestricted File Upload**

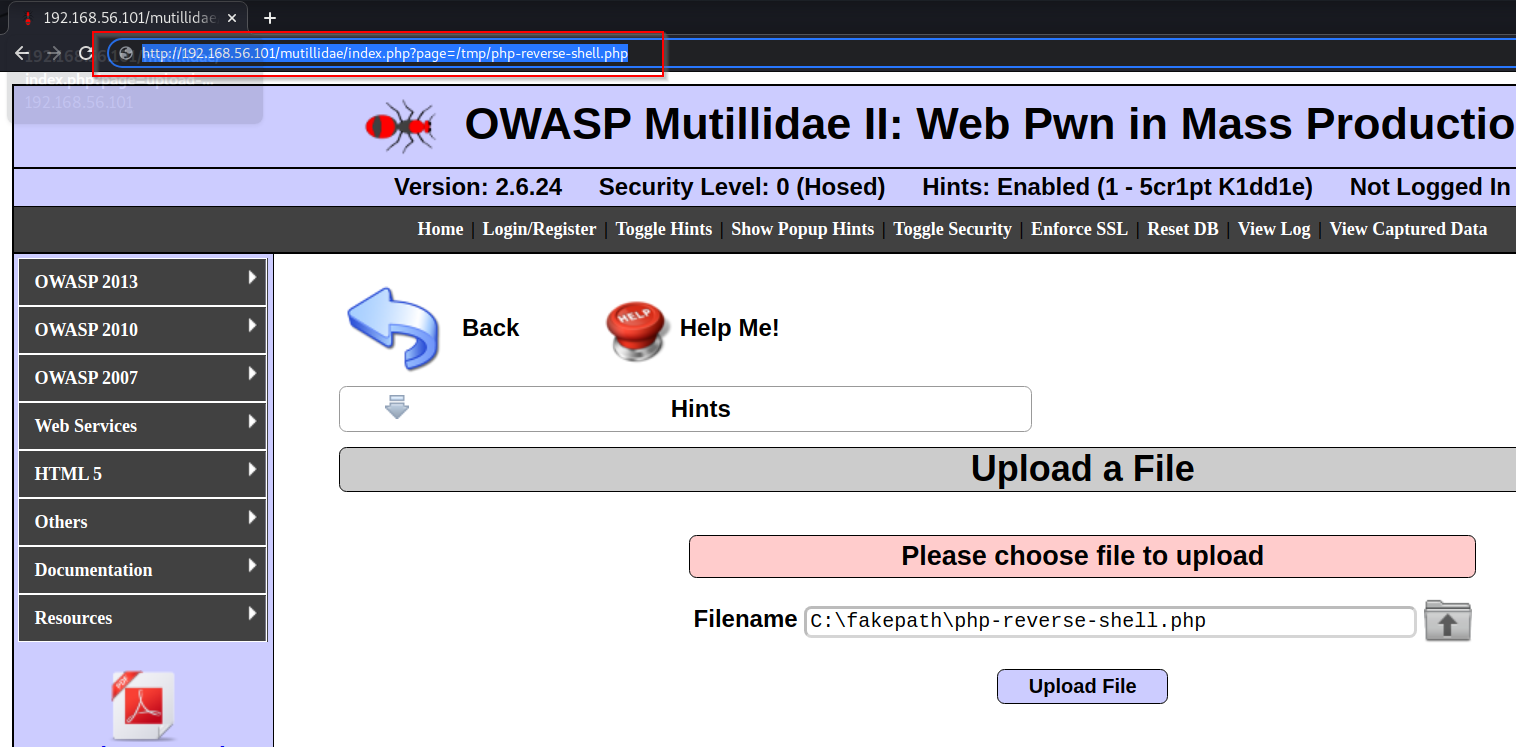
URL: <http://192.168.56.101/mutillidae/index.php?page=upload-file.php>

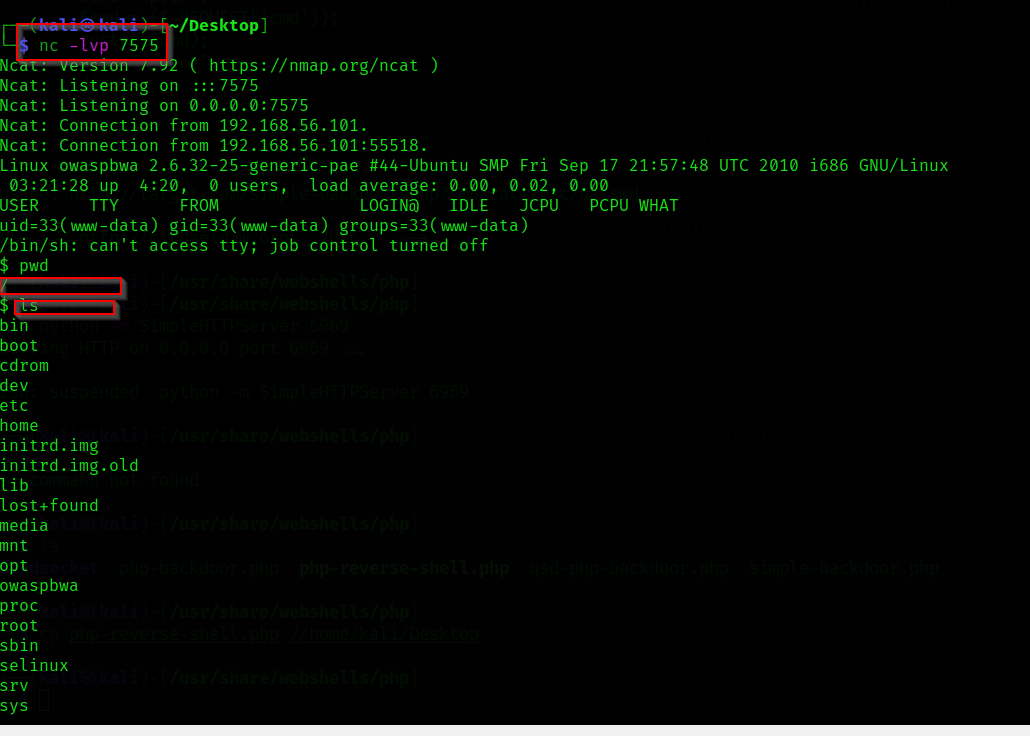
Description: we uploaded malicious file on server and we were able to compromised system











**Impact:**

1. This vulnerability give access to unauthorized access to some system data or functionality.
2. Compromise whole system or network

**Prevention:**

1. Remove Default value like password, username
2. Change default config files permissions
3. Change default error page (error page work as key in reconnaissance).
4. Educate and train your employees on the importance of security configurations and how they can impact the overall security of the organization.
5. Using CIS Benchmark for hardening

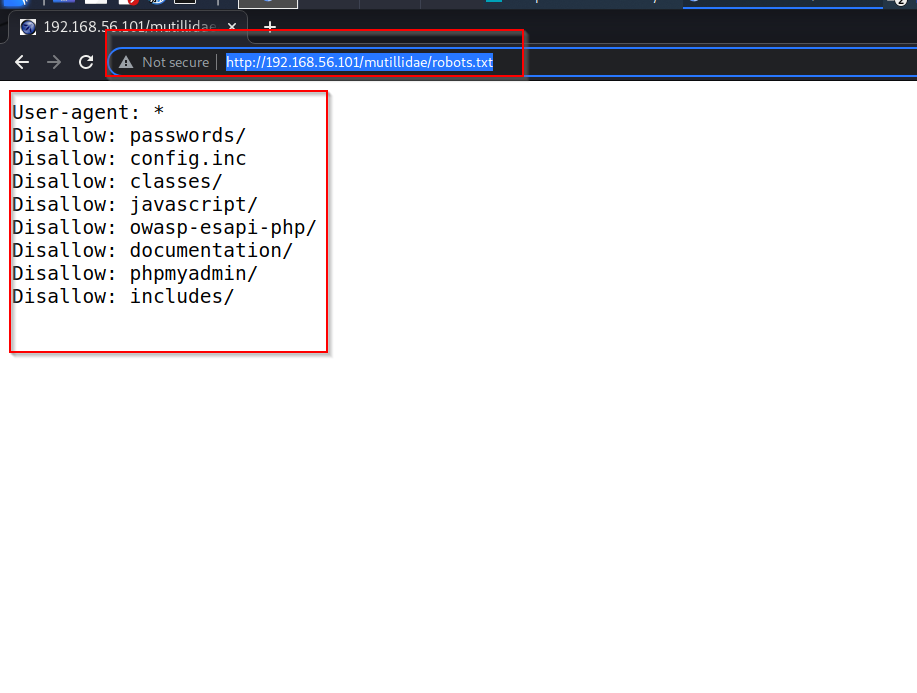
**6. SENSITIVE DATA EXPOSURE**

Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser

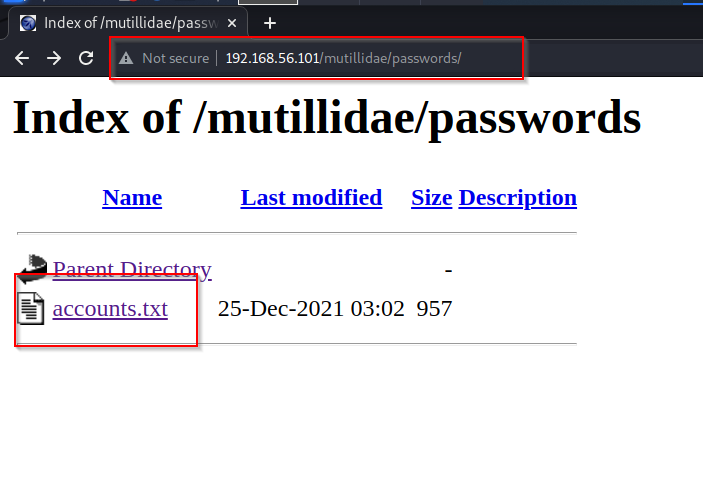
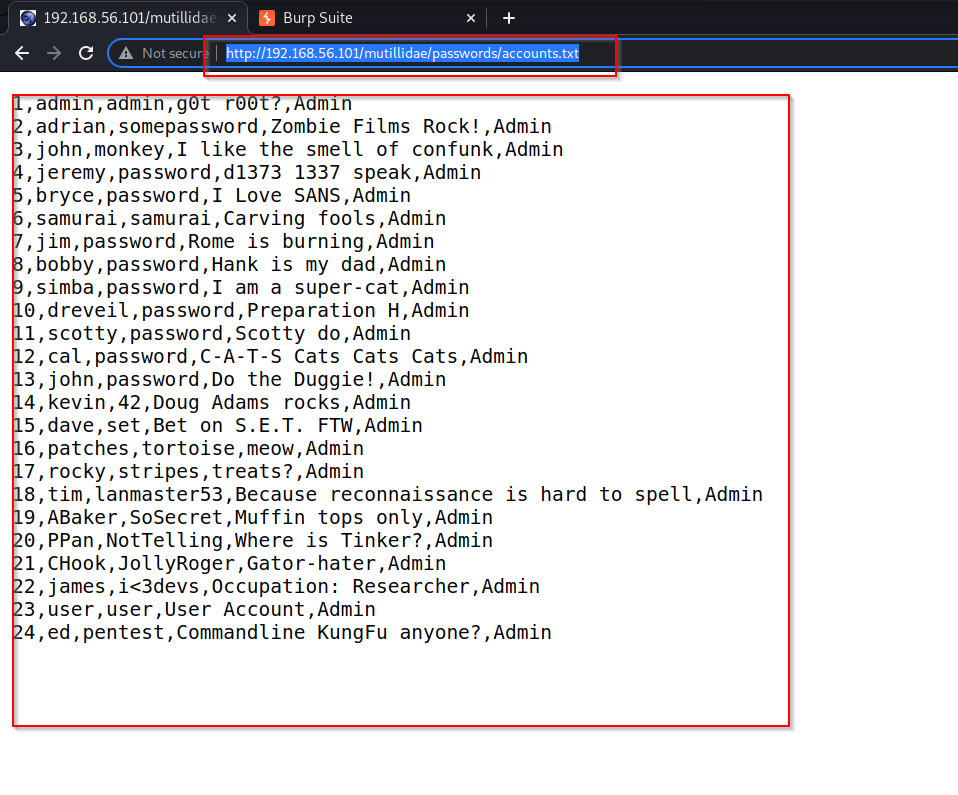
**6.1 Robots.txt**

URL: <http://192.168.56.101/mutillidae/robots.txt>

**Description**: with the help of robots.txt we got all url working on application



**6.2 Passwords**

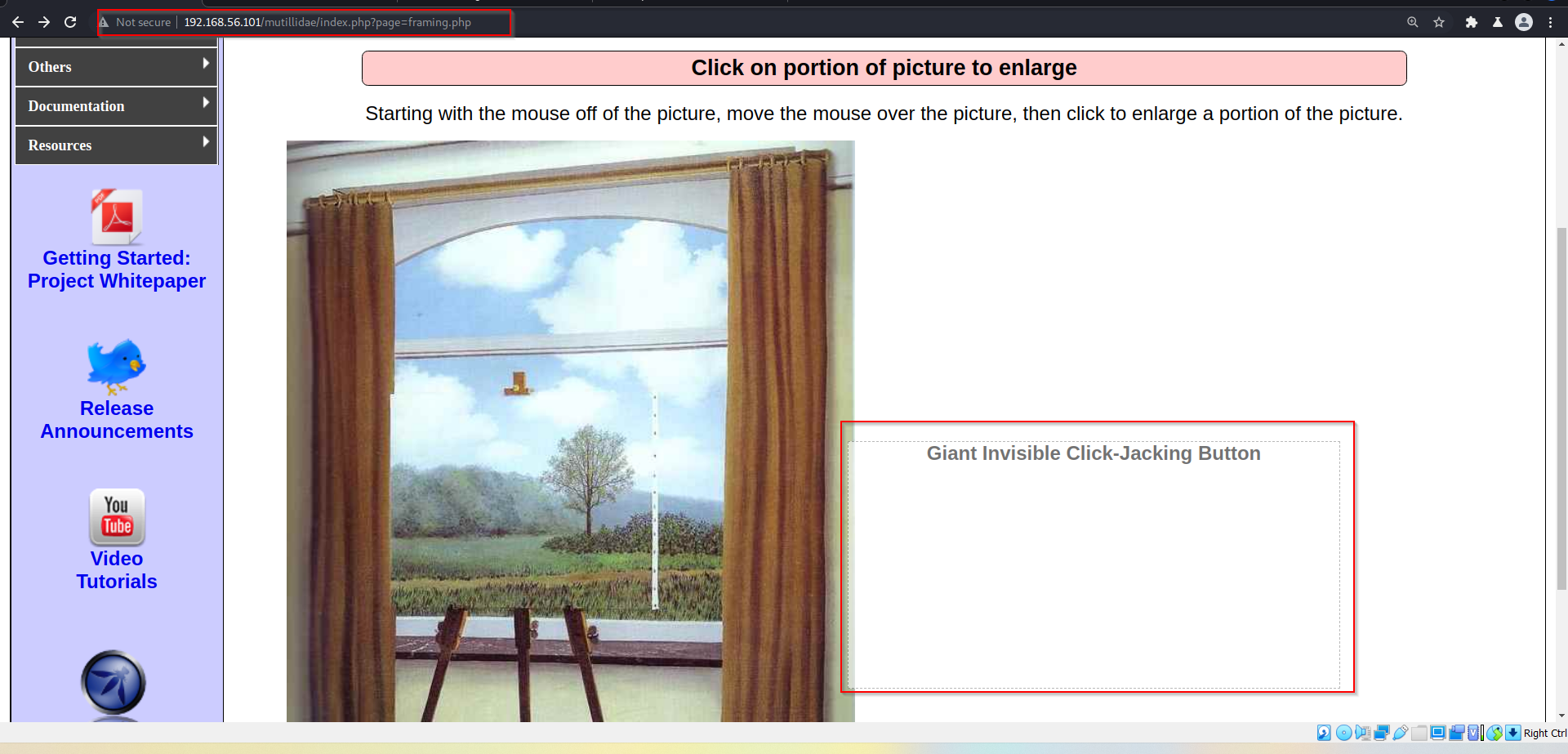
url: <http://192.168.56.101/mutillidae/passwords/accounts.txt>

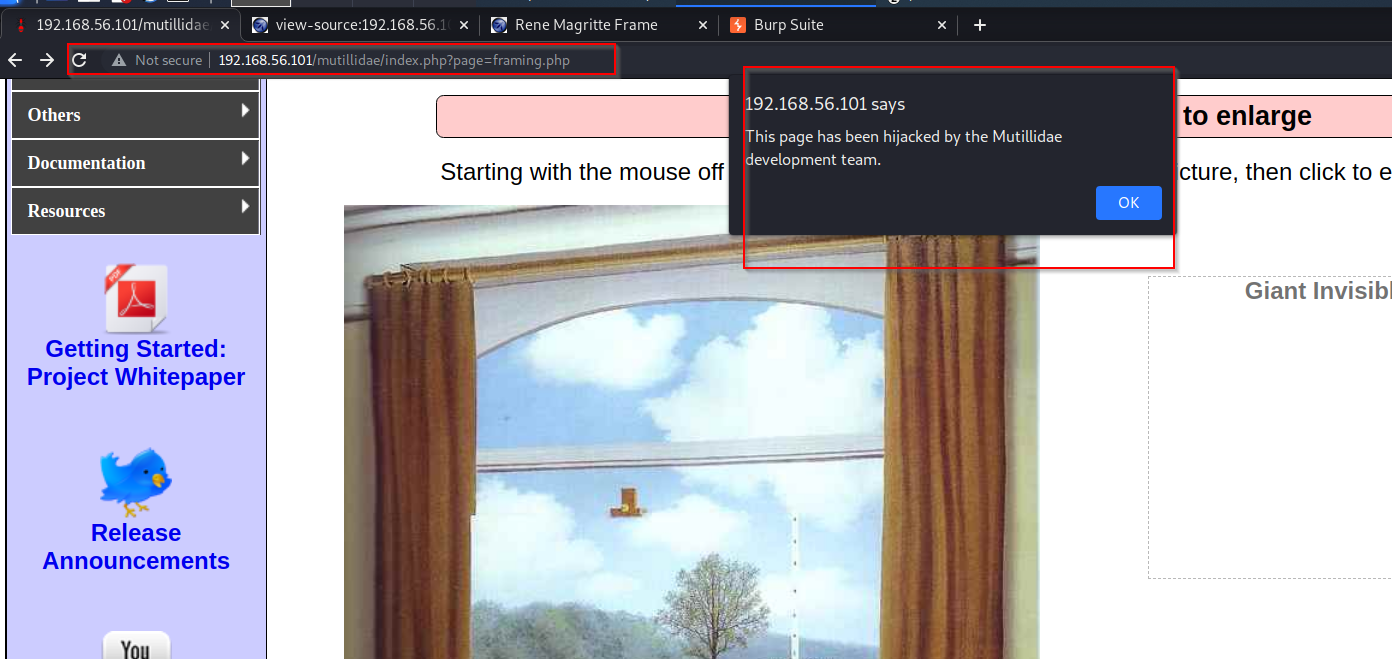
we have all the usernames and Passwords

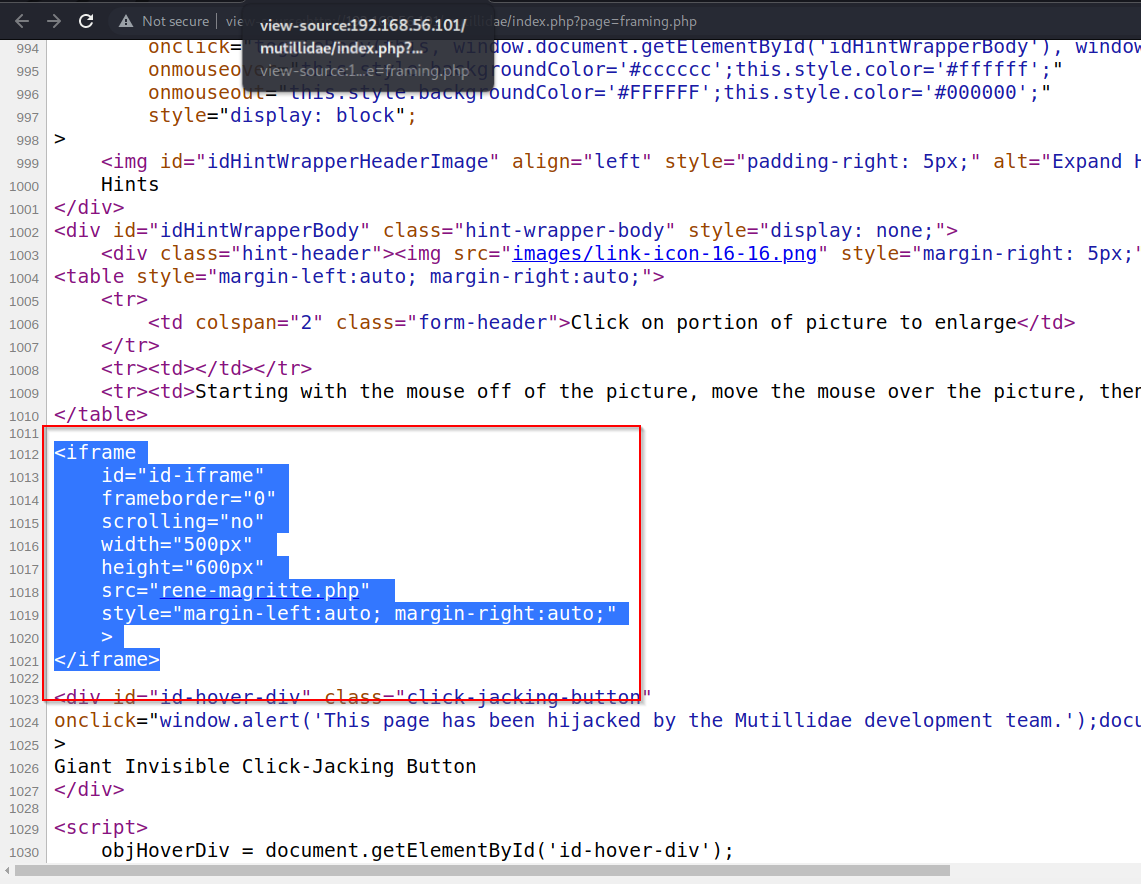
**6.3 Click jacking**

url: <http://192.168.56.101/mutillidae/index.php?page=framing.php>

**Description**: Clickjacking is an attack that tricks a user into clicking a webpage element which is invisible or disguised as another element







**Impact:**

1. Attacker can takeover system
2. Identity theft
3. Data misuse
4. Confidentiality breach

**Prevention:**

1. Use SSL when transmitting credential
2. Disable autocomplete Feature on forms
3. Encrypt all data in rest and transits
4. Use HTTPS protocol
5. Credential should be saved using strong cryptographic methods

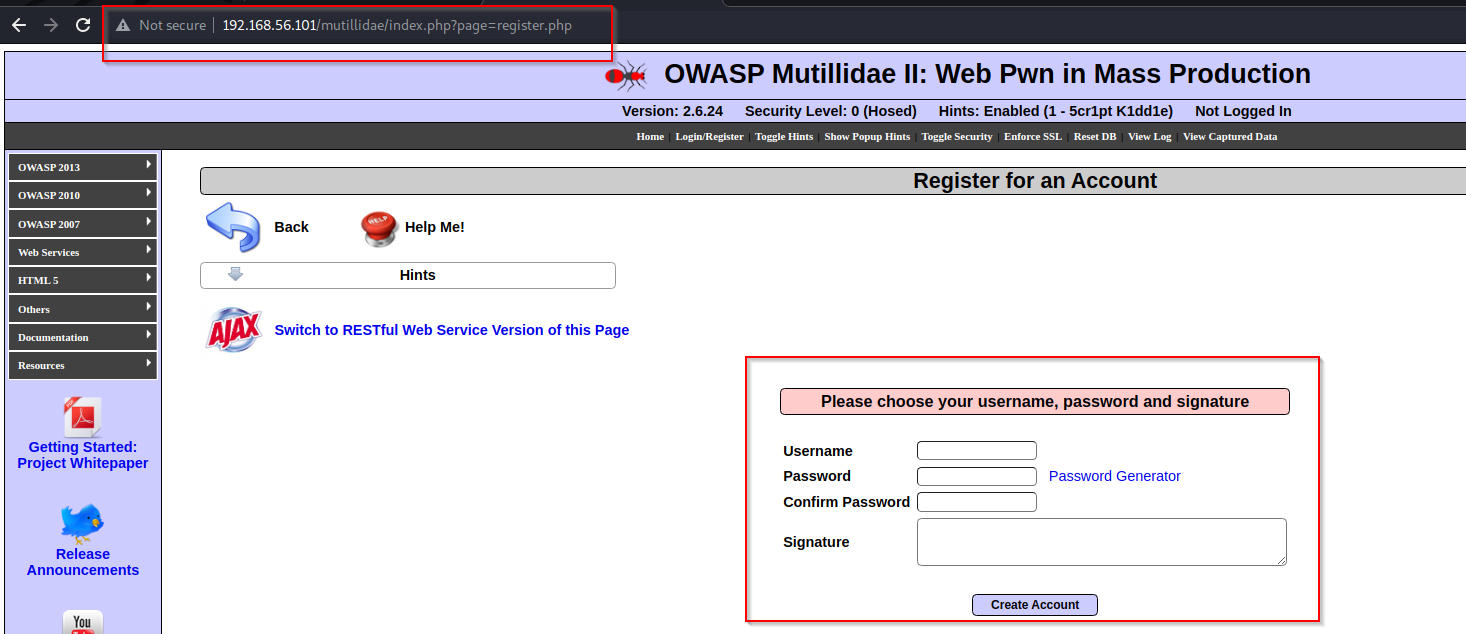
**8. CROSS SITE REQUEST FORGERY**

A CSRF attack forces a logged-on victim’s browser to send a forged HTTP request, including the victim’s session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim’s browser to generate requests the vulnerable application thinks are legitimate requests from the victim

**8.1 Register User**

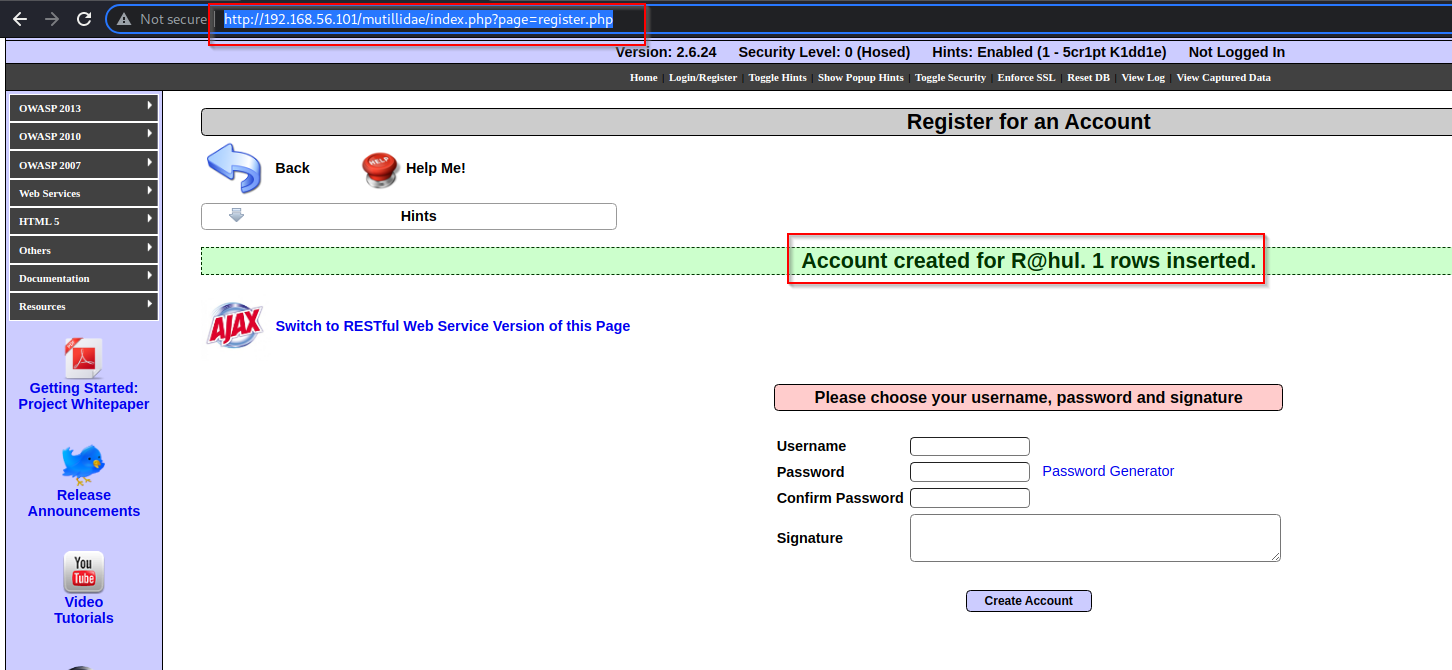
url: <http://192.168.56.101/mutillidae/index.php?page=register.php>

Description:









**Impact:**

1. Unauthorized modification
2. Misuse of privilege account
3. Identify theft
4. Unauthorized Arbitrary code Execution

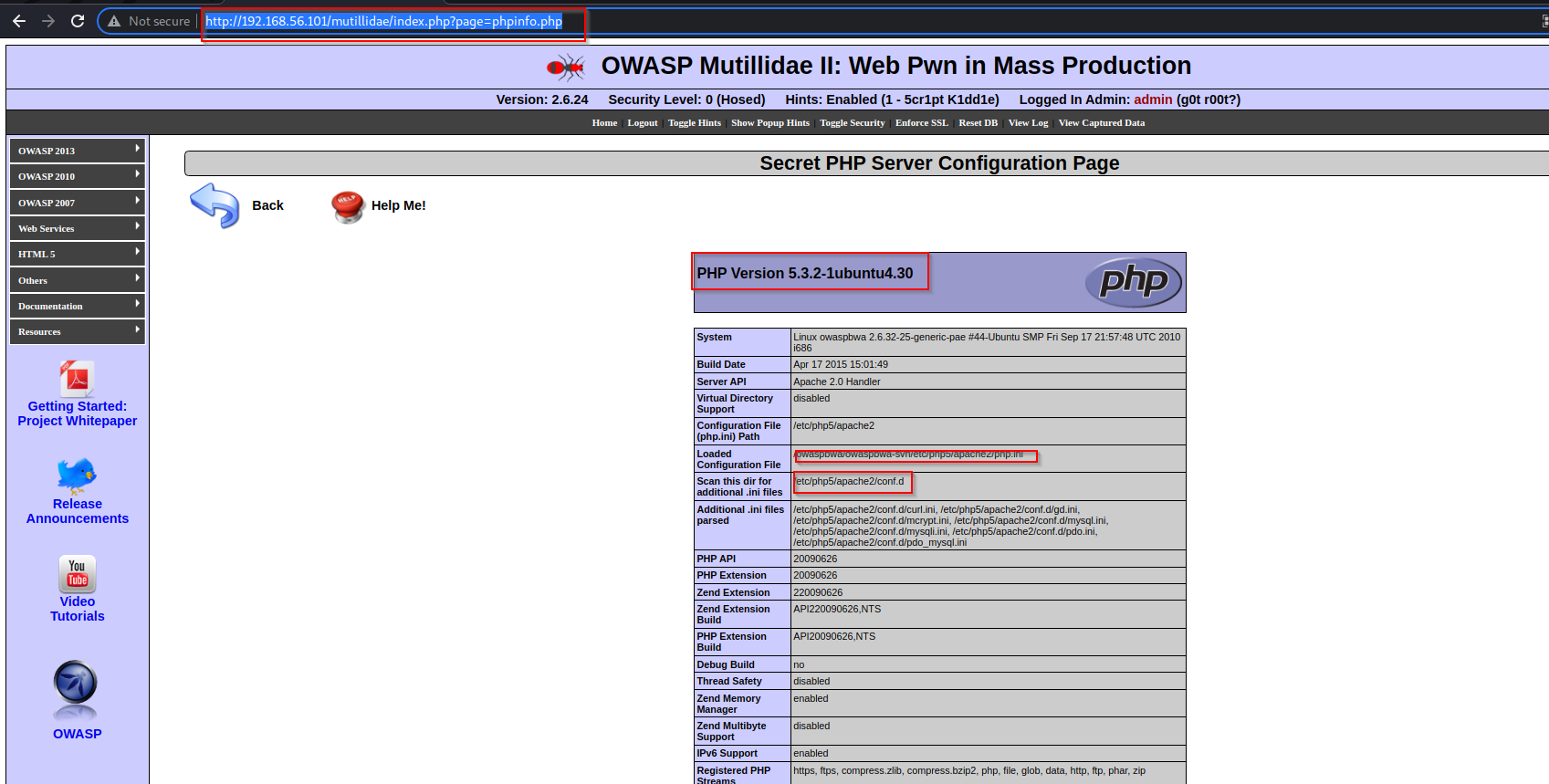
**Prevention:**

1. Input validation
2. HTTP Post method should be used
3. url rewriting
4. use advanced validation Techniques to Reduce CSRF (anti-CSRF token)

**9. USING COMPONENT WITH KNOWN VULNERABILITIES**

The attackers can simply make use of automated scan tools or perform manual analysis of the application to find the flaws and if they are able to find that the particular application uses a component which has previously been found vulnerable, they can easily try to exploit that vulnerability

URL: <http://192.168.56.101/mutillidae/index.php?page=phpinfo.php>



**Impact:**

1. Data Breaches
2. Attacker can get RCE
3. This makes application Exploitable

**Prevention:**

1. Deploy a proper patch management system and make sure the updates and security patching is from trusted vendors only and also remove unneeded or unused components for hardening the application
2. Implement regular monitoring and security assessments testing.
3. Remove unused dependencies, unnecessary features, components, files, and documentation

**10. UNVALIDATED REDIRECTS AND FORWARDS**

Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages

URL:

Description:

**Impact:**

1. Attacks like CSRF can be chained via this bug
2. XSS (Cross Site Scripting) can be exploited in this case
3. Theft of confidential Information
4. User can be tricked on a phishing page and can be exploited in several ways

**Prevention:**

1. Remove redirection functions from the Web Application & replace with direct links
2. A server-side list should be maintained for all the particular URLs that are needed to be redirected
3. Proper Validation in the URL those are redirecting must be integrated at backend
4. Absolute URLs must be used in the Web Applications for all the specific redirects.