replace title

replace author

replace date

Contents

List of Figures

Chapter 1

Introduction

Chapter 2

Executive Summary

2.1 Planning

2.1.1 Approach

2.1.2 Scope

2.1.3 Objectives

2.2 Methodology

2.2.1 Information Gathering

2.2.2 System Attacks

2.3 Summary of Findings

Chapter 3

Key Findings

3.1 Auction Site

This section covers the vulnerabilities found with the user-facing auction site.

3.1.1 Path Traversal

Security Implications / Risk Level

Path traversal allows for arbitrary file read across the system, for any files readable by the apache user (www-data). This is dangerous as it could potentially leak sensitive company data, as well as user data. If combined with other vulnerabilities, such as incorrect permissions on log files, it is possible to achieve Remote Code Execution through malicious log read/write.
Overall, the execution of this exploit is trivial, and the repercussions are potentially serious but not disasterous. Due to this, the risk level of this exploit is evaluated to be **medium**.

Cause of Vulnerability

The vulnerability is caused by the method used to retrieve and display image files on the website. Instead of directly referencing the image file through the 'src' field on an 'img' HTML tag, a PHP script is instead used to include the file.
While using PHP include scripts may not normally be dangerous, the file name to be retrieved can be edited by the user, allowing them to easily select which file should be displayed. A lack of filter/extension whitelist makes this even more potent.

Steps to Replicate

- Firstly the inspect element tool in Mozilla Firefox was used to inspect an image, revealing the image URL (Fig. 3.1).

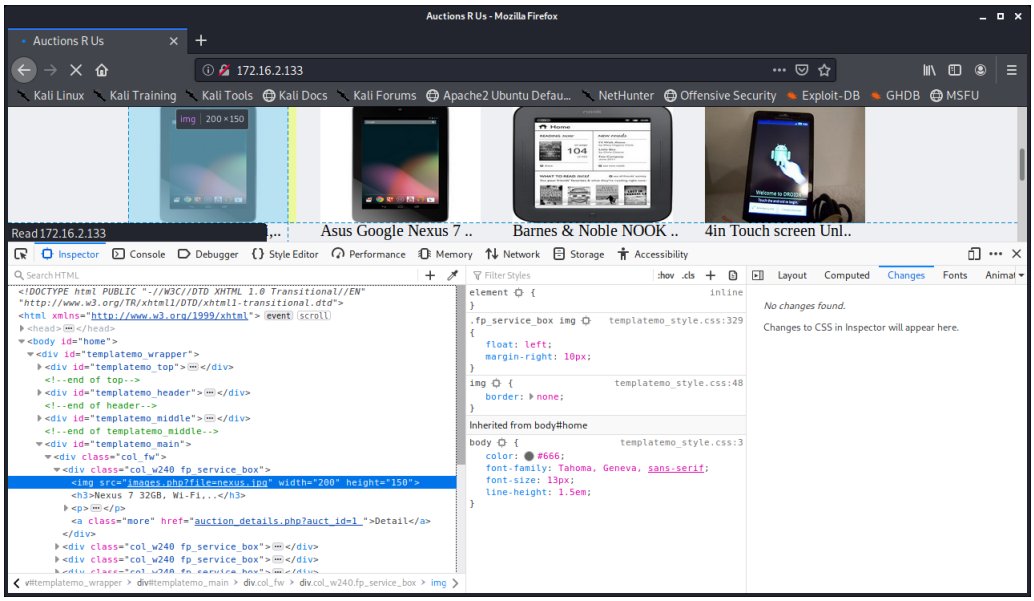

Figure 3.1: Inspecting the image

- The image URL could then be opened, showing the ASCII representation of the binary content for the image file (Fig. 3.2).
- Finally, the URL parameter 'file' could be replaced with a file path, allowing for arbitrary file read. In this example, the ../ operator was used to go up directories until root, and then the /etc/passwd file was navigated to (Fig. 3.3).

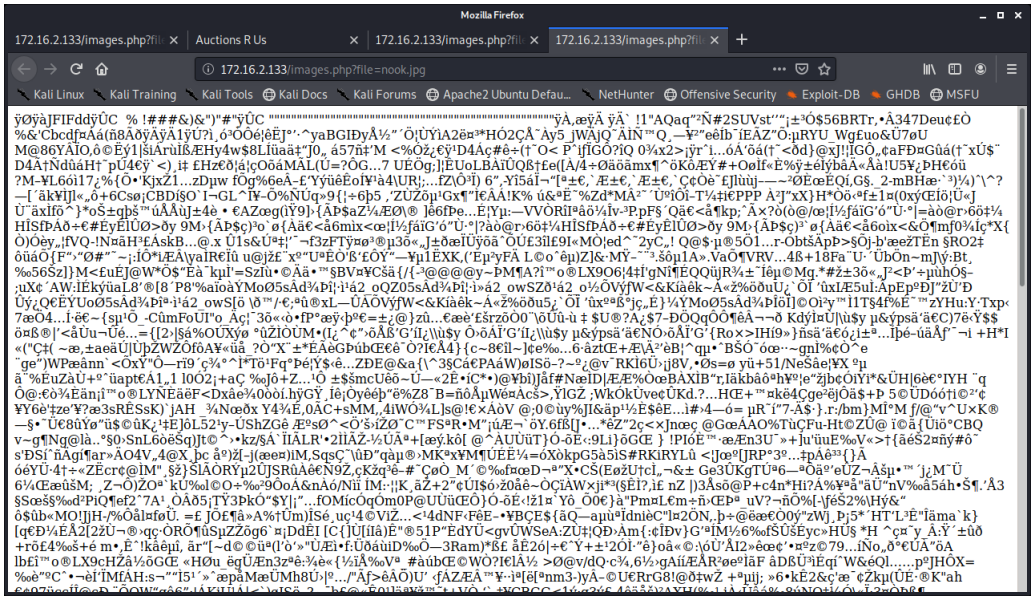

Figure 3.2: Viewing the image contents

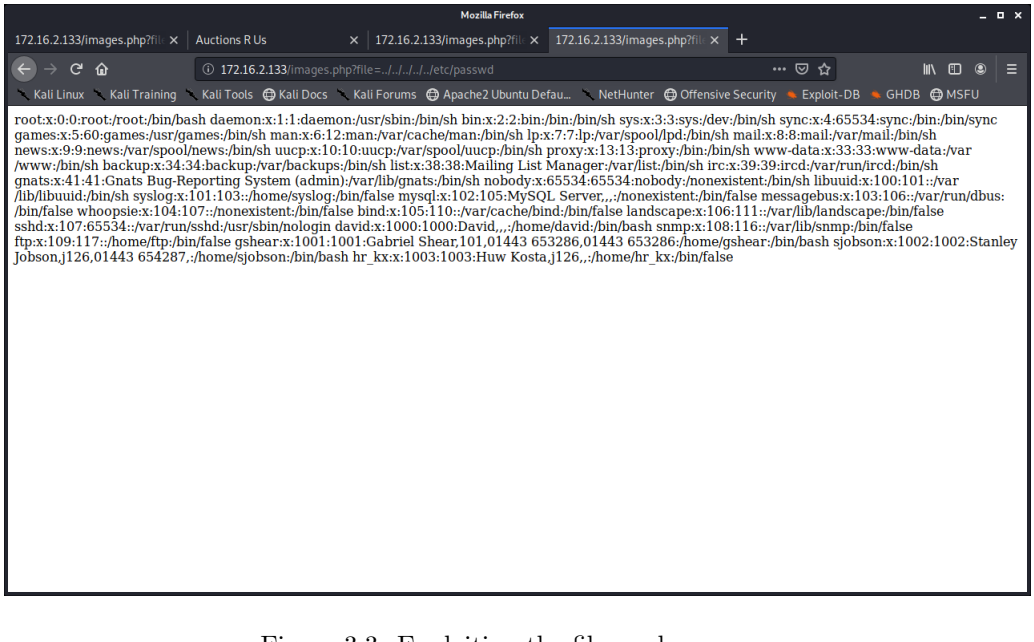

Figure 3.3: Exploiting the file read

3.1.2 SQL Injection

Security Implications / Risk Level

SQL injection is a blanket term covering any kind of unintended user control over the SQL queries interacting with a database. This can manifest in many forms, such as:

- Authentication bypass, where SQL queries can be modified to bypass authentication checks such as login forms.
- Union injection, where the UNION keyword in SQL can be used to access data from other columns, tables, and databases.
- Error based injection, where SQL errors are intentionally created in order to gain information about the database.
- Blind injection, where queries that return TRUE or FALSE can be used to gain information about the database.

From the testing done, the website appears to be vulnerable to both authentication bypass, allowing attackers to log in to accounts, and blind injection, allowing for full read access across the database.
The execution of these exploits are relatively easy with the use of tools like SQLmap, and the repercussions can be very serious, allowing attackers to log in to administrator accounts as well as reading any user/auction data from the database. Due to this, the risk level of this exploit is evaluated to be **high**.

Cause of Vulnerability

SQL injection vulnerabilities are a result of allowing unsanitised user input in to SQL queries. Sanitising SQL queries involves removing any kind of dangerous character from the input, such as quotation marks (single and double) and comment tags. If this is not done, attackers are able to modify queries in specific ways to allow them to perform SQL injection.
One example of this would be performing an authentication bypass injection. A normal query may use a query like:

```
SELECT * FROM users WHERE username = '$inputname' AND password =
'$inputpass';
```

If an attacker enters something like ' OR 1=1# in to the username field, the query becomes:

```
SELECT * FROM users WHERE username = '' OR 1=1#;
```

Which will pick the first username from the table and sign the attacker in.

Steps to Replicate

- For initial testing of SQL injection, a basic authentication bypass was used. The payload ' OR 1=1– - was entered in to the username field (Fig. 3.4), which subsequently allowed access to the 'David' account, giving use of the admin panel as well (Fig. 3.5).

Figure 3.4: Authentication bypass SQLi

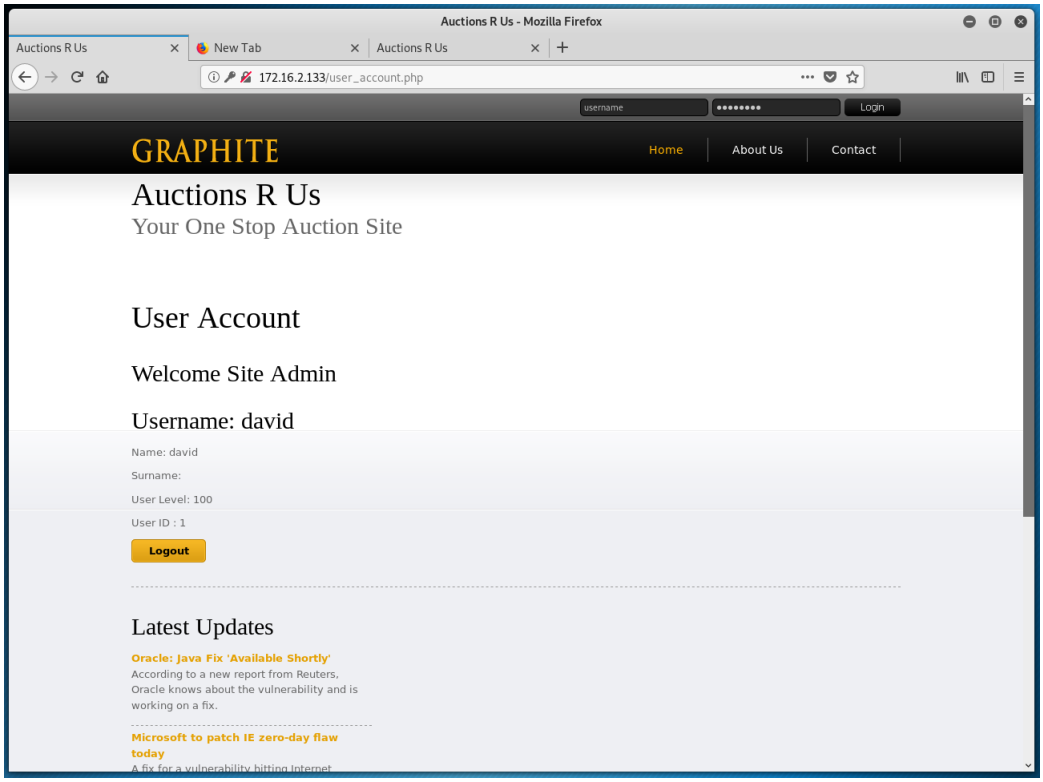

Figure 3.5: Admin panel access gained

3.1.3 Weak Authentication

Security Implications / Risk Level

Cause of Vulnerability

Steps to Replicate

script used to bruteforce the cookies

3.2 SRV1

3.3 SRV2

3.4 Ubuntu Client

Chapter 4

Recommendations

4.1 Auction Site

4.2 SRV1

4.3 SRV2

4.4 Ubuntu Client

Chapter 5

Conclusion

Chapter 6

References