# Task-1 — Web Application Security (DVWA)

## 1. Objective

Identify and exploit common web vulnerabilities in DVWA and demonstrate mitigations. Produce evidence (screenshots/video) and remediation code snippets.

## 2. Environment

**- Kali Linux VM (version)**

- DVWA (local) on Apache

- Tools: Burp Suite, Firefox, curl, phpinfo, python (for notes)

## 3. Tests performed & Findings

### 3.1 SQL Injection

Method: Used DVWA SQL Injection module at Low security. Payloads: ' OR '1'='1' --, UNION SELECT ... to extract users.

Finding: Database returned multiple user records and password hashes, proving SQL injection.

### 

### SQL Injection — Prepared Statements (PDO)

**PHP PDO prepared statement example:**  
$pdo = new PDO("mysql:host=127.0.0.1;dbname=dvwa","dvwa","p@ssw0rd");  
$stmt = $pdo->prepare("SELECT first\_name, last\_name FROM users WHERE user\_id = ?");  
$stmt->execute([$\_GET['id']]);  
while ($row = $stmt->fetch()) {  
 echo htmlspecialchars($row['first\_name'], ENT\_QUOTES, 'UTF-8') . ' ' . htmlspecialchars($row['last\_name'], ENT\_QUOTES, 'UTF-8');  
}

### 3.2 Cross-Site Scripting (XSS)

Types tested: Stored (High), Reflected (Medium), DOM (Low).

Payloads used (examples):

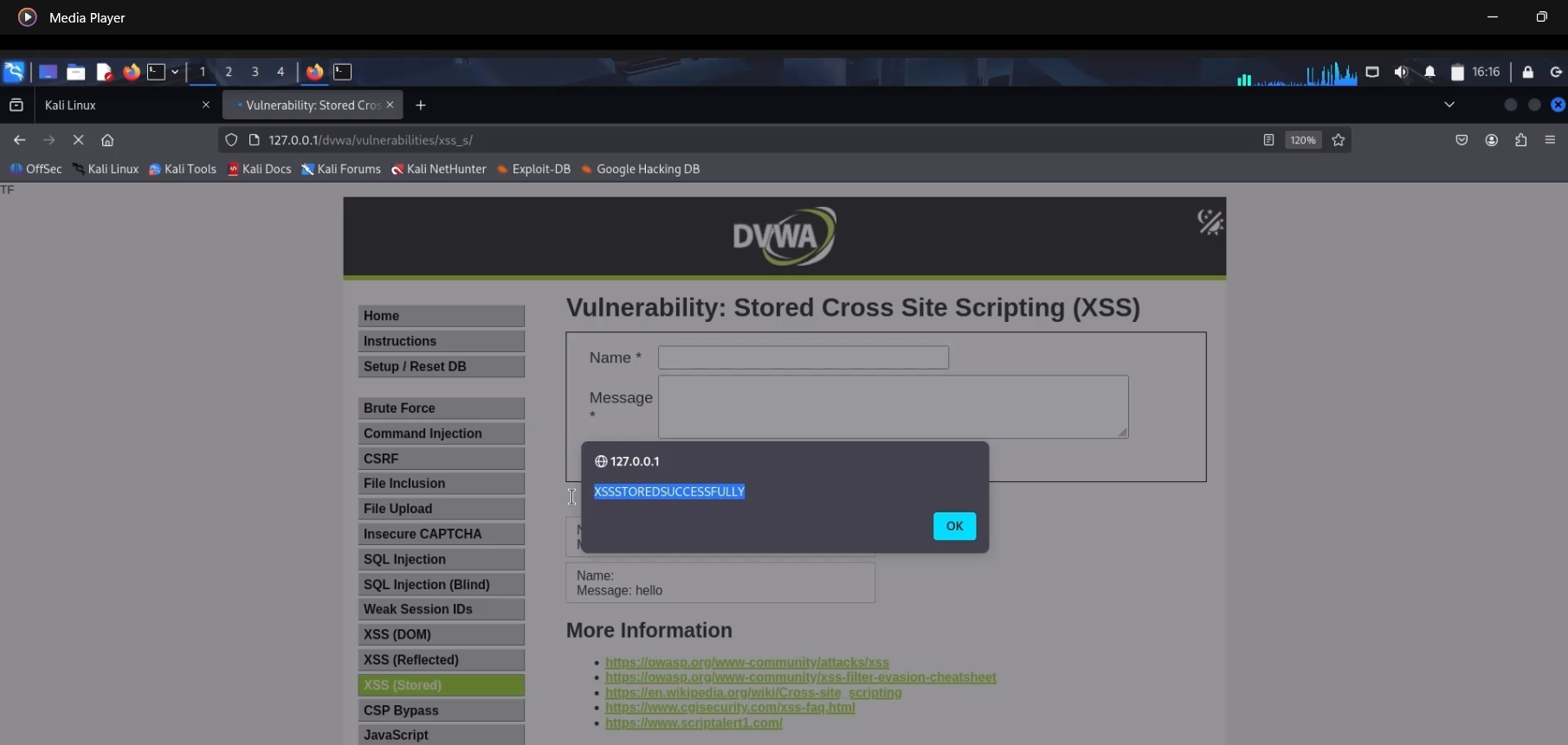
<script>alert("Stored XSS")</script>

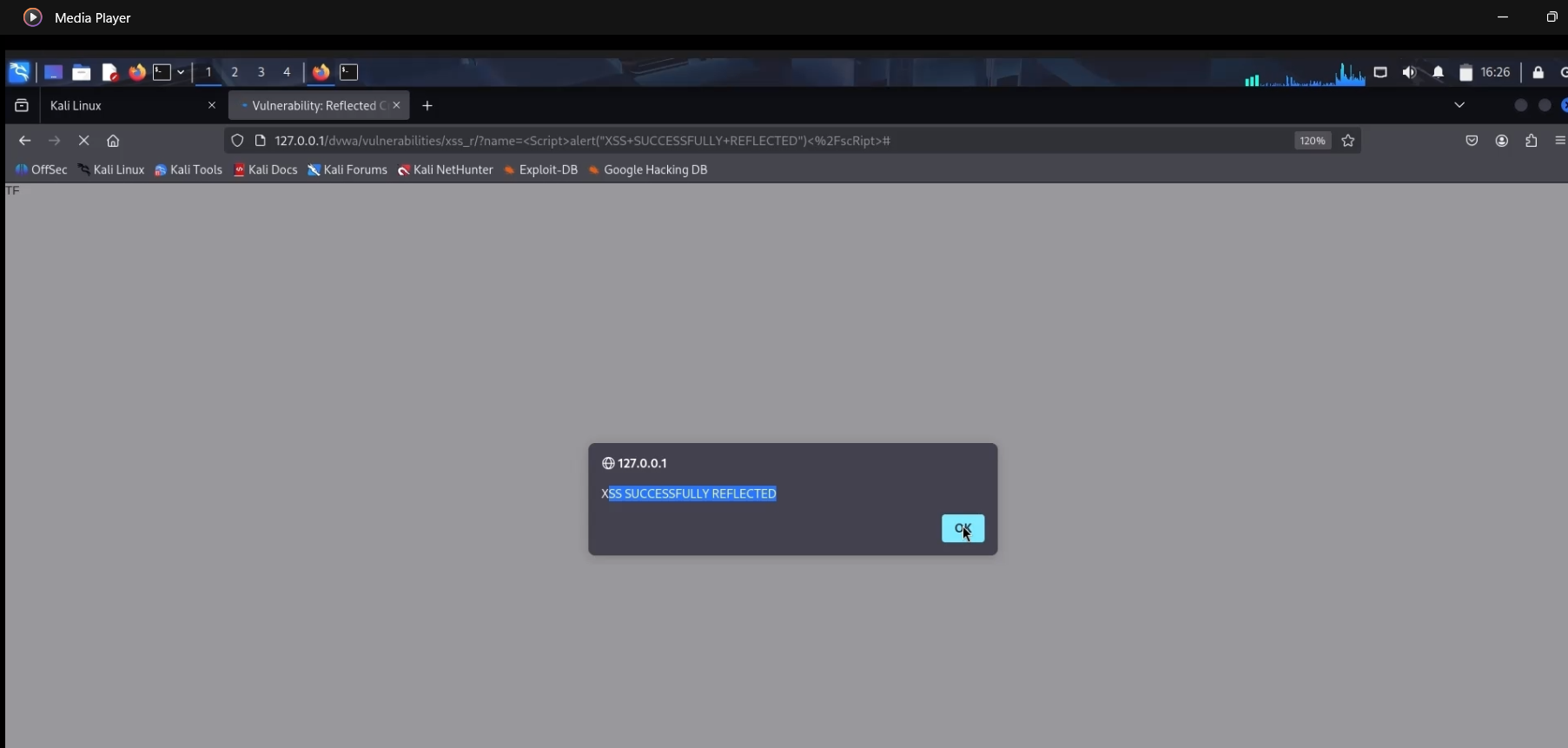
Findings: Stored payload persisted and executed for visitors; reflected payload executed on submission; DOM payload executed client-side.

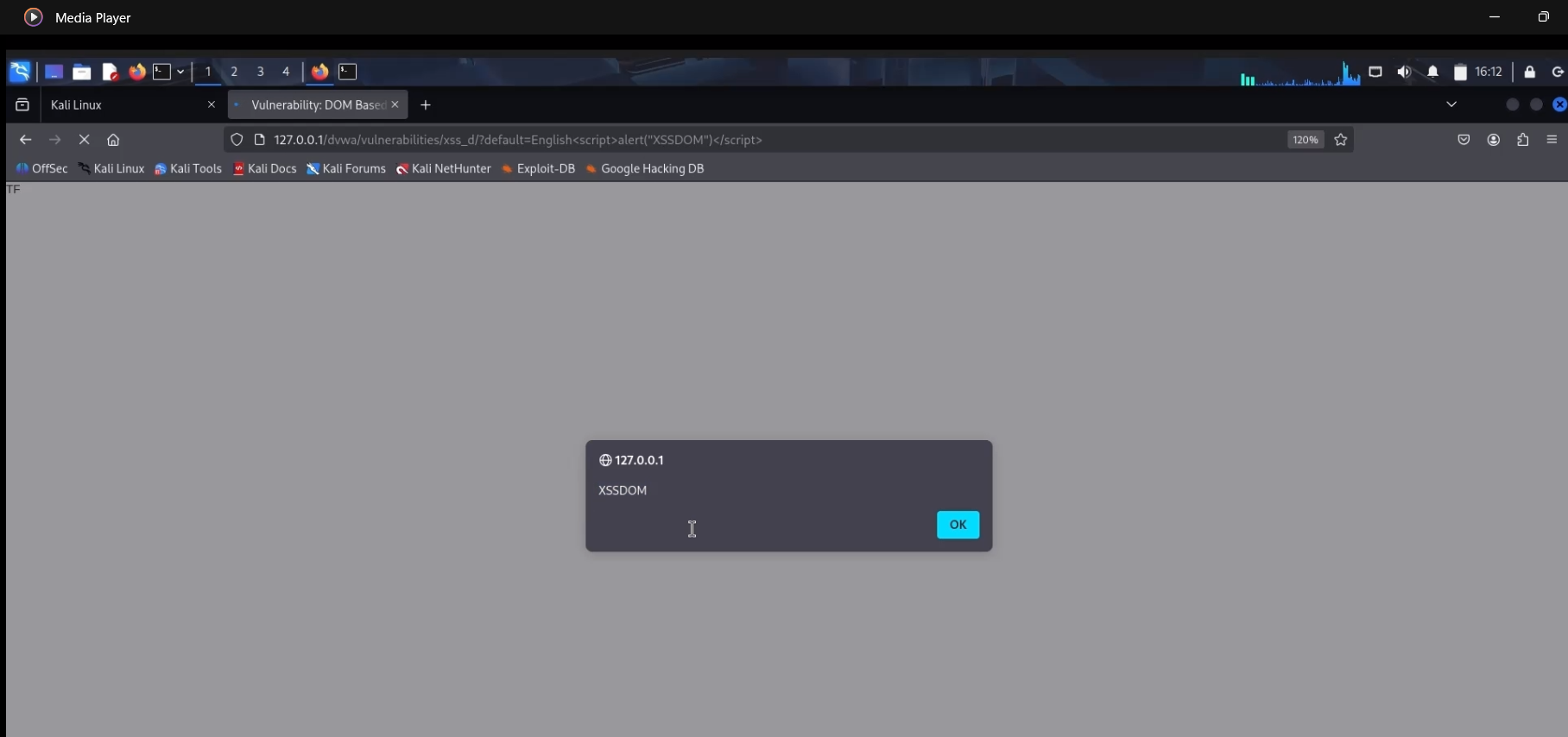
### XSS — Output Encoding

**PHP output encoding example:**  
echo htmlspecialchars($user\_input, ENT\_QUOTES, 'UTF-8');

Also implement Content-Security-Policy (CSP) headers.



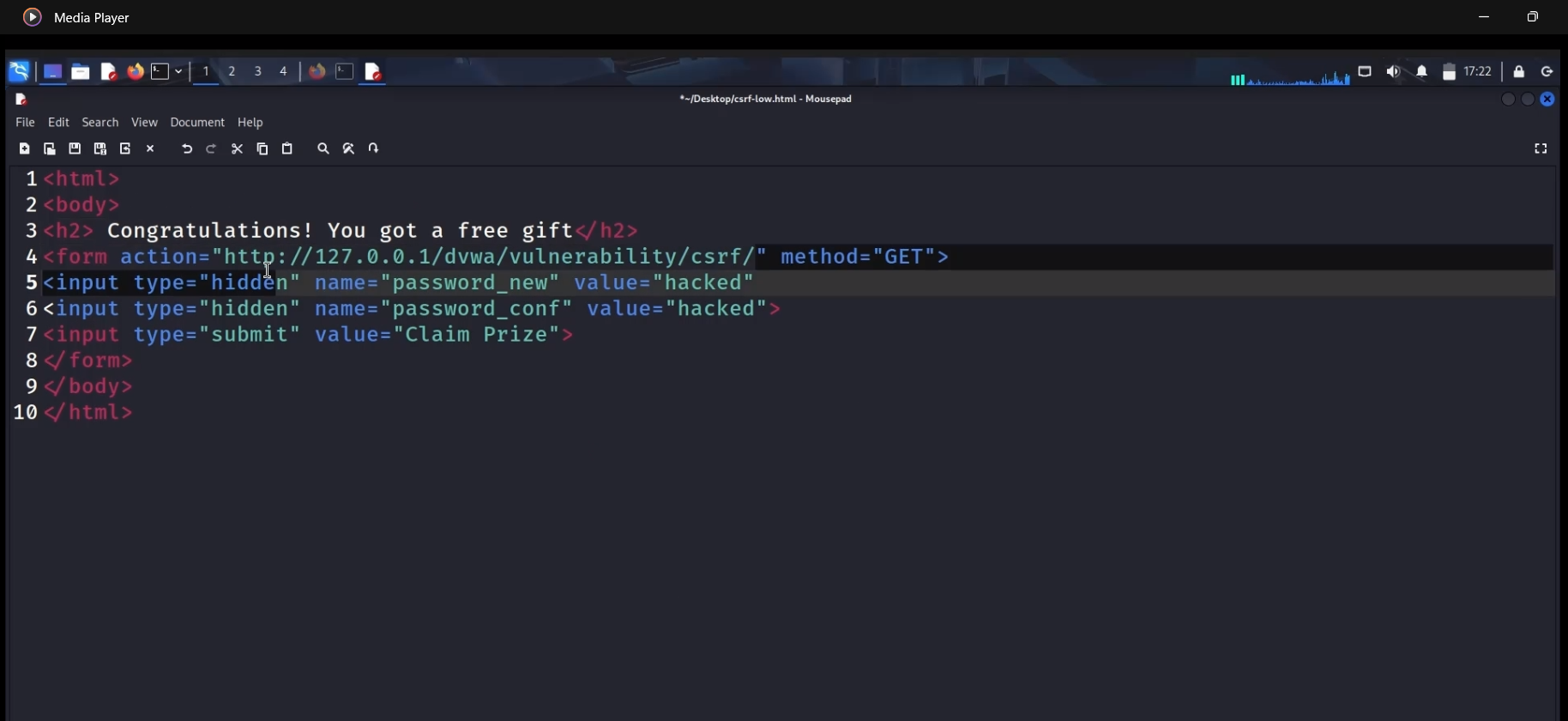


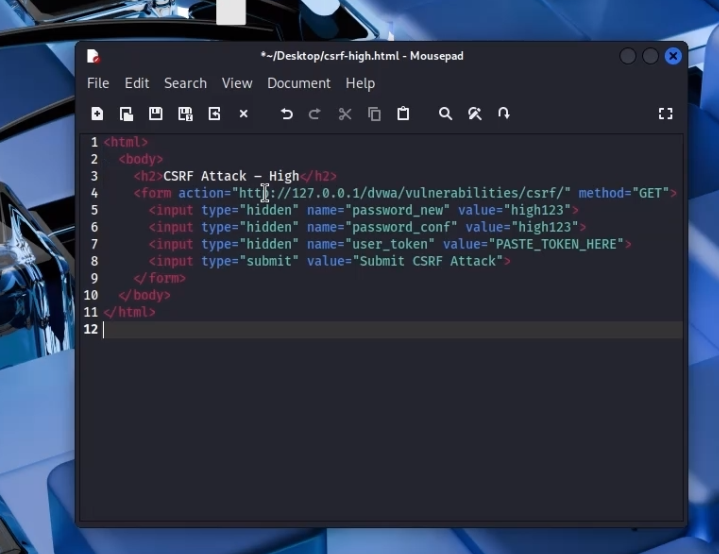


### 3.3 Cross-Site Request Forgery (CSRF)

Tested at Low, Medium, High security levels.

Low: No CSRF token; external form changed user password silently.

High: Token rotates and validated; attack blocked.



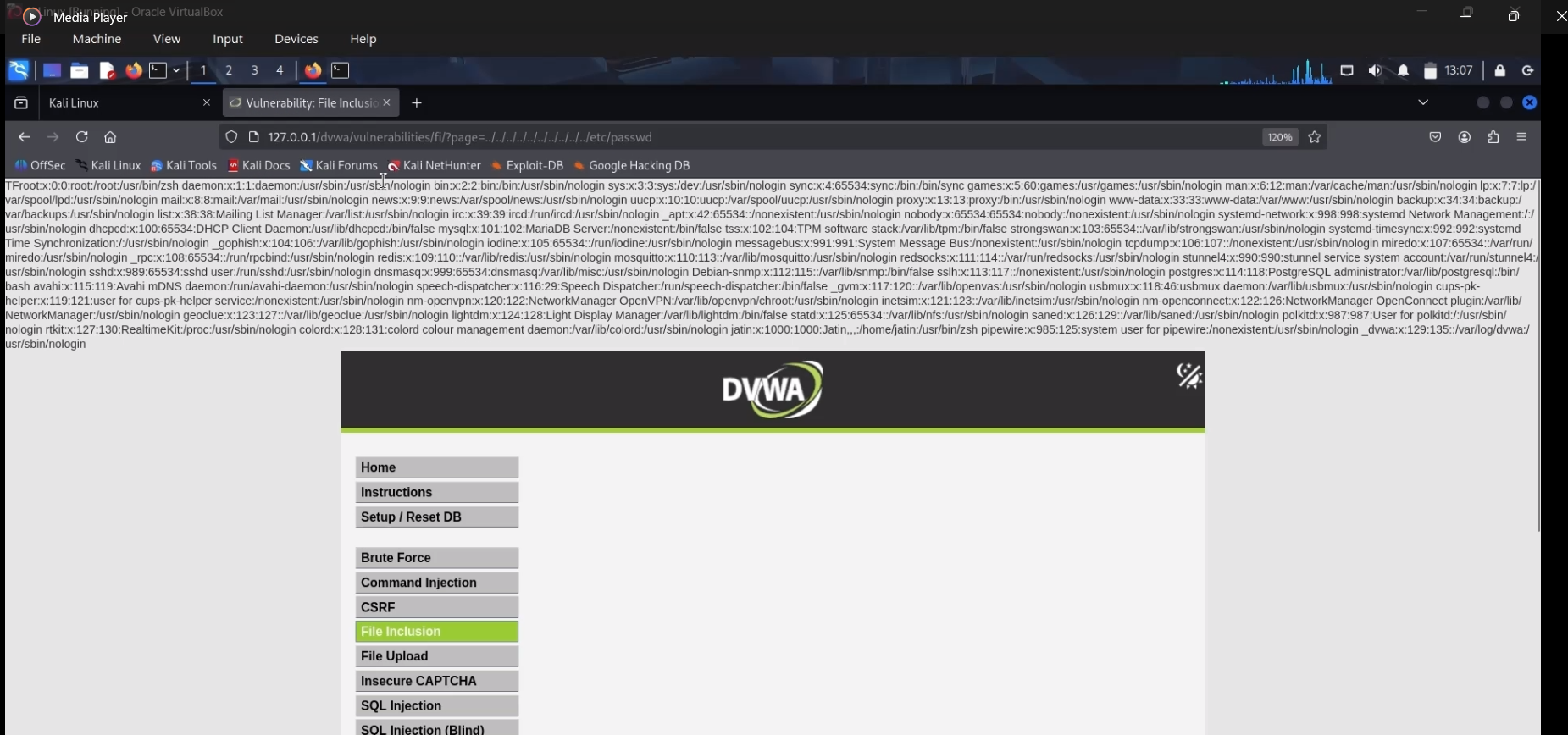
### CSRF — Token-based protection

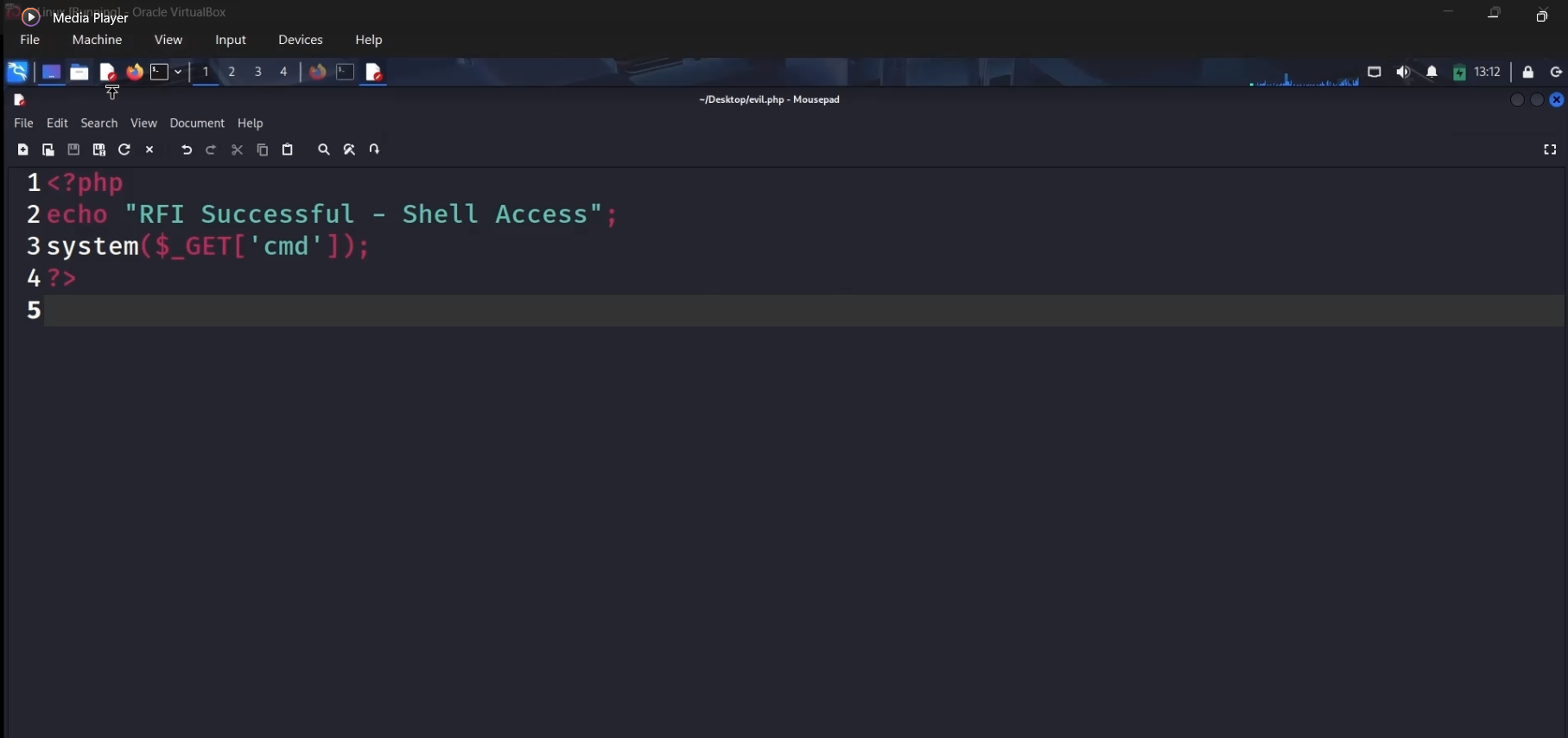
**CSRF token generation and validation (PHP):**  
// generate  
if (empty($\_SESSION['token'])) {  
 $\_SESSION['token'] = bin2hex(random\_bytes(32));  
}  
// form  
<input type="hidden" name="user\_token" value="<?php echo $\_SESSION['token']; ?>">  
// validation  
if (!hash\_equals($\_SESSION['token'], $\_REQUEST['user\_token'] ?? '')) {  
 die('Invalid CSRF token');  
}

3.4 File Inclusion (LFI / RFI)

LFI test payload: ?page=../../../../etc/passwd

Finding: /etc/passwd contents were shown (LFI confirmed).

RFI: Enabled allow\_url\_include temporarily in lab and included attacker evil.php which executed (whoami output). 



### File Inclusion — Whitelist & disable remote includes

Disable remote includes: set allow\_url\_include = Off in php.ini (recommended).

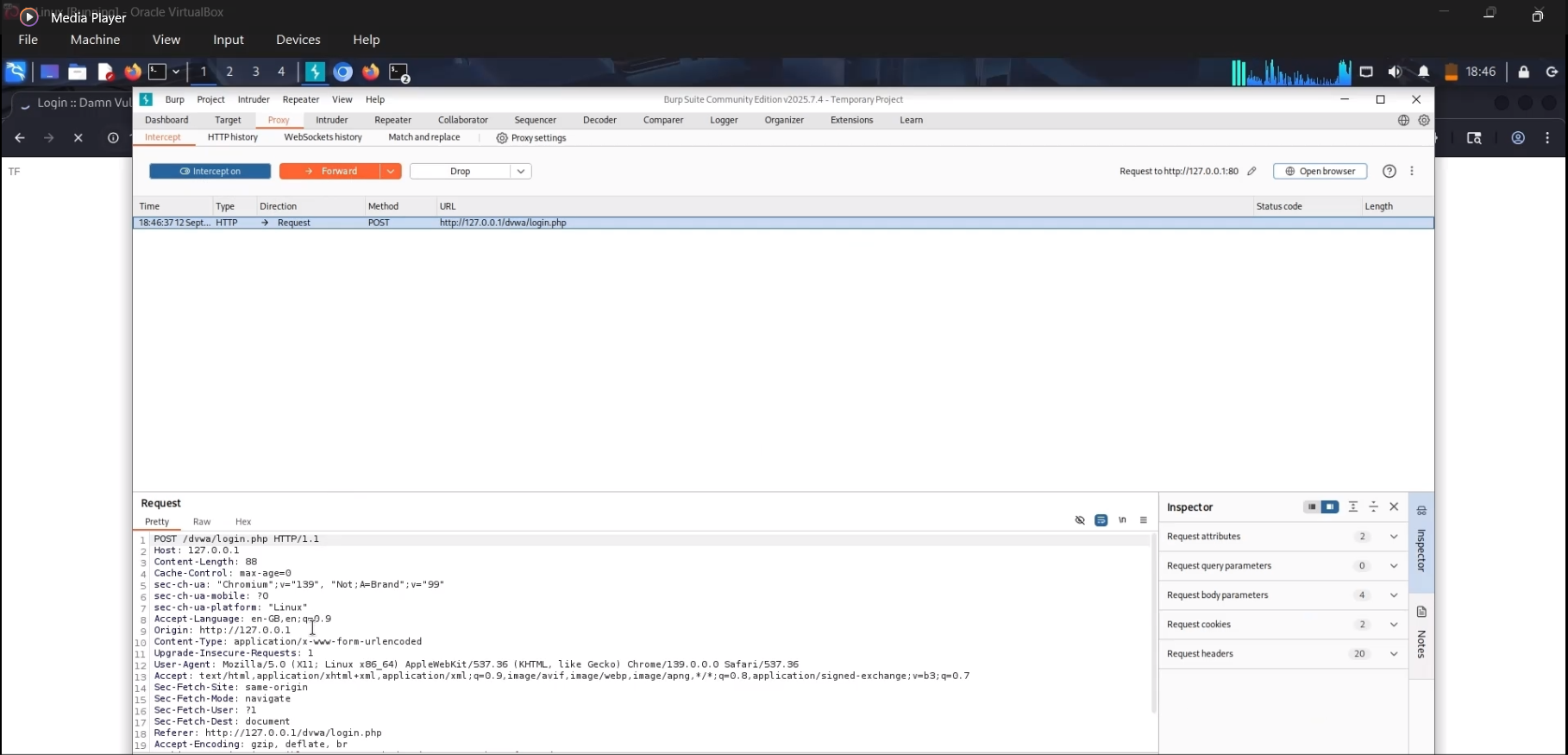
**Whitelist include example:**  
$allowed = ['home.php','about.php'];  
$page = $\_GET['page'] ?? 'home.php';  
if (in\_array($page, $allowed, true)) {  
 include \_\_DIR\_\_ . '/pages/' . $page;  
} else {  
 echo "Access denied";  
}

### 3.6 Burp Suite (Intercept & Intruder)

Intercept: Captured DVWA login POST with username & password in the body.

Modify: Edited password in transit and observed browser result (tampering proof).

Intruder: Fuzzed password field with a small wordlist and observed responses for anomalies.



### 

### 3.7 Web Security Headers

Checked headers with curl/devtools. Added recommended headers in Apache config.

## 

### HTTP Security Headers — Apache config

Add the following Apache config to enable security headers (example):

sudo a2enmod headers  
sudo tee /etc/apache2/conf-available/security-headers.conf > /dev/null <<'EOF'  
Header always set X-Content-Type-Options "nosniff"  
Header always set X-Frame-Options "DENY"  
Header always set Referrer-Policy "no-referrer"  
Header always set X-XSS-Protection "1; mode=block"  
Header always set Permissions-Policy "geolocation=(), microphone=()"  
Header always set Content-Security-Policy "default-src 'self' 'unsafe-inline' 'unsafe-eval'; script-src 'self' 'unsafe-inline' 'unsafe-eval';"  
EOF  
sudo a2enconf security-headers  
sudo systemctl restart apache2

## 5. Conclusion

Summary of risks observed and recommended next steps: revert any temporary insecure lab settings, apply fixes above, enforce HTTPS and strong authentication, and perform regular security testing.