







Bypassing CSRF token validation



In this section, we'll explain what CSRF tokens are, how they protect against CSRF attacks, and how you can potentially bypass these defenses.

What is a CSRF token?

A CSRF token is a unique, secret, and unpredictable value that is generated by the server-side application and shared with the client. When issuing a request to perform a sensitive action, such as submitting a form, the client must include the correct CSRF token. Otherwise, the server will refuse to perform the requested action.

A common way to share CSRF tokens with the client is to include them as a hidden parameter in an HTML form, for example:


```
<form name="change-email-form" action="/my-account/change-email" method="POST">
  <label>Email</label>
  <input required type="email" name="email" value="example@normal-website.com">
  <input required type="hidden" name="csrf" value="50FaWgdOhi9M9wyna8taRlk3ODOR8d6u
  <button class='button' type='submit'> Update email </button>
</form>
```

Submitting this form results in the following request:

```
POST /my-account/change-email HTTP/1.1
Host: normal-website.com
Content-Length: 70
Content-Type: application/x-www-form-urlencoded

csrf=50FaWgdOhi9M9wyna8taRlk3ODOR8d6u&email=example@normal-website.com
```

When implemented correctly, CSRF tokens help protect against CSRF attacks by making it difficult for an attacker to construct a valid request on behalf of the victim. As the attacker has no way of predicting the correct value for the CSRF token, they won't be able to include it in the malicious request.

 **Note**

CSRF tokens don't have to be sent as hidden parameters in a POST request. Some applications place CSRF tokens in HTTP headers, for example. The way in which tokens are transmitted has a significant impact on the security of a mechanism as a whole. For more information, see [How to prevent CSRF vulnerabilities](#).

Common flaws in CSRF token validation

CSRF vulnerabilities typically arise due to flawed validation of CSRF tokens. In this section, we'll cover some of the most common issues that enable attackers to bypass these defenses.

Validation of 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.

In this situation, the attacker can switch to the GET method to bypass the validation and deliver a **CSRF attack**:

```
GET /email/change?email=pwned@evil-user.net HTTP/1.1
Host: vulnerable-website.com
Cookie: session=2yQIDcpia41WrATfjPqvm9tOkDvkMvLm
```

Want to track your progress
have a more personalized
learning experience? (It's fr

Sign up

Logi

In this topic

CSRF »

XSS vs CSRF »

Bypassing flawed CSRF to validation »

Bypassing SameSite cooki restrictions »

Bypassing Referer-based defenses »

How to prevent CSRF vulnerabilities »

All topics

SQL injection »

XSS »

CSRF »

Clickjacking »

DOM-based »

CORS »

XXE »

SSRF »

Request smuggling »

Command injection »

Server-side template injecti

Insecure deserialization »

Directory traversal »

Access control »

Authentication »

OAuth authentication »

Business logic vulnerabilitie

Web cache poisoning »

HTTP Host header attacks

WebSockets »

Information disclosure »

File upload vulnerabilities »

JWT attacks »

Essential skills »

Client-side prototype polluti



Validation of 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.

In this situation, the attacker can remove the entire parameter containing the token (not just its value) to bypass the validation and deliver a **CSRF attack**:

```
POST /email/change HTTP/1.1
Host: vulnerable-website.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 25
Cookie: session=2yQIDcpia41WrATfjPqvm9tOkDvkMvLm

email=pwned@evil-user.net
```

[TRY FOR FREE](#)

LAB

PRACTITIONER

[CSRF where token validation depends on token being present >>](#)

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.

LAB

PRACTITIONER

[CSRF where token is not tied to user session >>](#)

CSRF token is tied to a non-session cookie

In a variation on the preceding vulnerability, 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=rZHCnSzEp8dbI6atzagGoSYyqJq

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

This situation is harder to exploit but is still vulnerable. If the web site contains any behavior that allows an attacker to set a cookie in a victim's browser, then an attack is possible. The attacker can log in to the application using their own account, obtain a valid token and associated cookie, 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.

LAB

PRACTITIONER

[CSRF where token is tied to non-session cookie >>](#)

Note

The cookie-setting behavior does not even need to exist within the same web application as the **CSRF vulnerability**. Any other application within the same overall DNS domain can potentially be leveraged to set cookies in the application that is being targeted, if the cookie that is controlled has suitable scope. For example, a cookie-setting function on `staging.demo.normal-website.com` could be leveraged to place a cookie that is submitted to `secure.normal-website.com`.

CSRF token is simply duplicated in a cookie

In a further variation on the preceding vulnerability, 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, and is advocated because it is simple to implement and avoids the need for any server-side state:

```
POST /email/change HTTP/1.1
Host: vulnerable-website.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 68
Cookie: session=1DQGdzYbOJQzLP7460tfyiv3do7MjyPw; csrf=R8ov2YBfTYmzFyjtit8o2hKBuoIjXXV
csrf=R8ov2YBfTYmzFyjtit8o2hKBuoIjXXVpa&email=wiener@normal-user.com
```

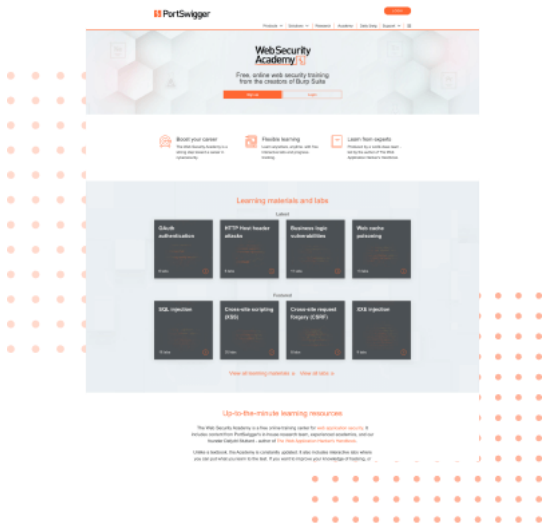
In this situation, the attacker can again perform a CSRF attack if the web site contains any cookie setting functionality. Here, the attacker doesn't need to obtain a valid token of their own. They simply invent a token (perhaps in the required format, if that is being checked), leverage the cookie-setting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

LAB

PRACTITIONER

CSRF where token is duplicated in cookie »

Register for free to track your learning progress



- ✔ Practise exploiting vulnerabilities on realistic targets.
- ✔ Record your progression from Apprentice to Expert.
- ✔ See where you rank in our Hall of Fame.

Enter your email

>

Already got an account? [Login here](#)