CSRF – Cross-Site Request Forgery

# CSRF Attack

A cross-site request forgery (CSRF) attack tricks a user into performing actions on a website that they are already authenticated for. This can lead to unauthorized fund transfers, password changes, and data theft

**Detailed explanation:**

In a successful CSRF attack, the attacker causes the victim user to carry out an action unintentionally. For example, this might be to change the email address on their account, to change their password, or to make a funds transfer. Depending on the nature of the action, the attacker might be able to gain full control over the user's account. If the compromised user has a privileged role within the application, then the attacker might be able to take full control of all the application's data and functionality [[1](https://portswigger.net/web-security/csrf)]

# What are CSRF Tokens?

A [CSRF token](https://brightsec.com/blog/csrf-token/) is a unique, unpredictable secret value generated by a server-side application, and sent to the client for inclusion in subsequent HTTP requests issued by the client. After the token is issued, when the client makes a request, the server checks to see if the request contains the expected token, and rejects it if the token is missing or invalid.

CSRF tokens can prevent CSRF attacks, because they prevent attackers from forming fully valid HTTP requests, which they can feed to a victim. The attacker cannot determine or predict the value of the user’s CSRF token, so any request they generate should not be accepted by the application.

# Conditions of attack

CSRF (Cross-Site Request Forgery) works when three conditions are met:

1. **Relevant Action** – The attacker targets an action, like changing a password or modifying user permissions.
2. **Cookie-Based Session** – The application relies only on session cookies to verify users, without extra validation.
3. **Predictable Requests** – The request has no secret parameters, so the attacker can guess or craft it easily.

If these conditions exist, an attacker can trick a logged-in user into unknowingly performing actions on their behalf. [[1](https://portswigger.net/web-security/csrf)]

# What to include

CSRF tokens should be high-entropy, unpredictable, and securely verified:

* Use a **cryptographically secure PRNG** with a timestamp and a static secret.
* Add **user-specific entropy** and hash the token for extra security.
* Ensure **tokens are unique** and expire after a short time.
* **Verify tokens securely**, e.g., by comparing hashes.
* **Never send tokens in GET requests** to prevent exposure in URLs or referrer headers. [2](https://brightsec.com/blog/csrf-token/#what-is-csrf-token)

# PRNG

A **cryptographically secure pseudorandom number generator** (**CSPRNG**) or **cryptographic pseudorandom number generator** (**CPRNG**) is a [pseudorandom number generator](https://en.wikipedia.org/wiki/Pseudorandom_number_generator) (PRNG) with properties that make it suitable for use in [cryptography](https://en.wikipedia.org/wiki/Cryptography). It is also referred to as a **cryptographic random number generator** (**CRNG**). [[3](https://brightsec.com/blog/csrf-token/#what-is-csrf-token)]

# Website cookie

**Magic cookies** are an old computing term that refers to packets of information that are sent and received without changes to the data. This would commonly be used for a login to computer database systems, such as a business internal network. This concept predates the modern “cookie” we use today. [[2](https://www.kaspersky.com/resource-center/definitions/cookies)]

CSRF Token in Django prevents malicious attacks by generating a unique token for each session. This token is included in user requests and verified by the server to ensure authenticity. While primarily protecting data-altering actions, it's good practice to include CSRF tokens in all forms. [[3](https://brightsec.com/blog/csrf-token/#what-is-csrf-token)]

# Code overview

## Syntax

{% csrf\_token %}

## How CSRF Protection Works

If a user is logged in and clicks a malicious link, an attacker could execute unwanted actions like transferring funds or changing account details. CSRF protection ensures such requests are blocked.

## Token Generation & Inclusion

Django generates a random CSRF token when a session starts. This token is stored on the server and included in forms using {% csrf\_token %}:

<form method="post">

{% csrf\_token %}

<button type="submit">Submit</button>

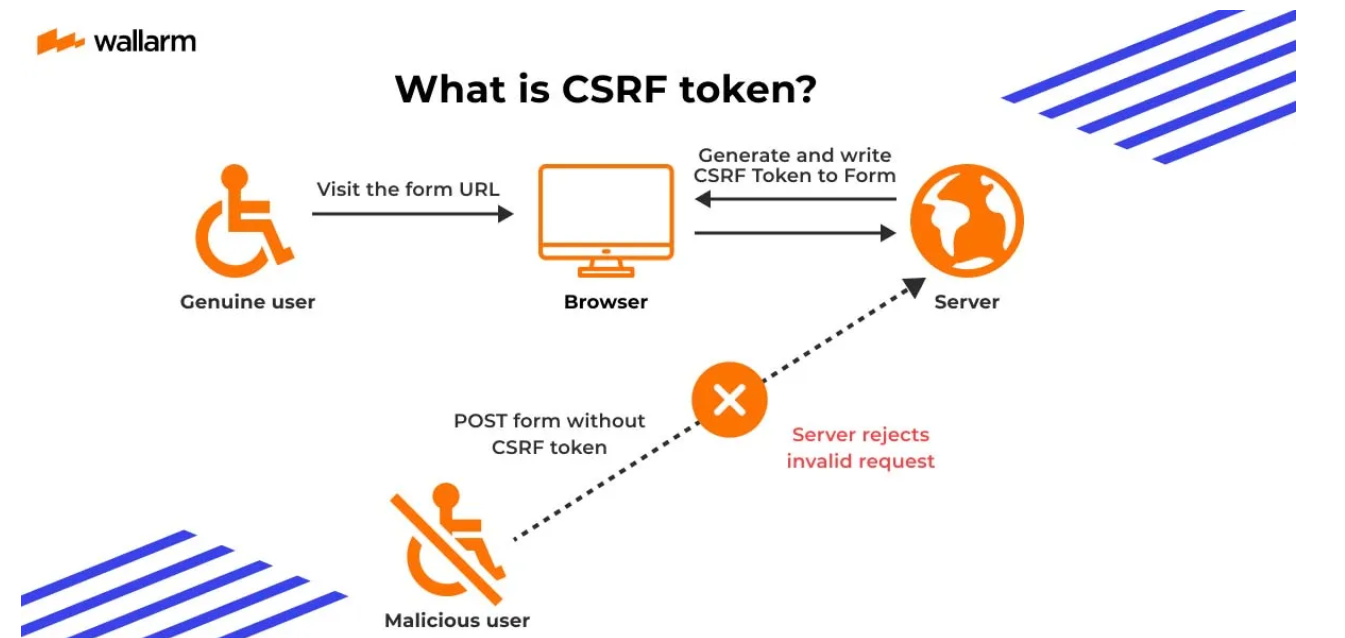
</form>

## Token Validation

When the form is submitted, the token is sent with the request. The server verifies it against the stored session token. If they match, the request proceeds; otherwise, it's rejected as a potential attack.

## Token Validation on Submission

Diagram: How the CSRF token is validated step by step



# Code-wise explaination

## How the attack happens

**Vulnerable Django View (No CSRF Protection)**

from django.http import HttpResponse

from django.shortcuts import render

def update\_email(request):

if request.method == "POST":

new\_email = request.POST.get("email")

request.user.email = new\_email

request.user.save()

return HttpResponse("Email updated successfully!")

return render(request, "update\_email.html")

**Corresponding HTML Form (No CSRF Token)**

<form method="POST" action="/update-email/">

<input type="email" name="email" placeholder="Enter new email">

<button type="submit">Change Email</button>

</form>

**Why this is malicious?**

* No CSRF protection, so any website can trigger this request.
* If the user is logged in, the request will go through automatically without their knowledge.

## How a Hacker Exploits This?

**The attacker creates a malicious website with the following hidden code:**

<img src="https://victim-website.com/update-email/?email=hacker@gmail.com" style="display:none;">

## How to Disable (or Exempt) a View from CSRF Protection (Fully remove the crsf protection)

**Now -> website fully vulnerable to CSRF attack**

from django.views.decorators.csrf import csrf\_exempt

**@csrf\_exempt # ❌ CSRF protection removed (NOT SAFE)**

def update\_email(request):

if request.method == "POST":

new\_email = request.POST.get("email")

request.user.email = new\_email

request.user.save()

return HttpResponse("Email updated successfully!")

return render(request, "update\_email.html")

## How to Prevent CSRF Attacks (safe)

**Secure Django View (With CSRF Protection)**

from django.shortcuts import render

from django.http import HttpResponse

from django.middleware.csrf import get\_token

def update\_email(request):

if request.method == "POST":

new\_email = request.POST.get("email")

request.user.email = new\_email

request.user.save()

return HttpResponse("Email updated securely!")

return render(request, "update\_email.html", {"csrf\_token": get\_token(request)})

**Secure HTML Form (With CSRF Token)**

<form method="POST" action="/update-email/">

{% csrf\_token %}

<input type="email" name="email" placeholder="Enter new email">

<button type="submit">Change Email</button>

</form>

**Why is this safe?**

* **Django automatically generates a CSRF token** and verifies it before processing requests.
* The hacker **cannot steal the token** because it is embedded in the real website’s form.

# Why replying on cookie, authentication is risky

**Why Relying Only on Cookies for Authentication is Risky**

Websites often use **cookies** to remember that you're logged in. Every time you visit a page or submit a form, your browser automatically includes these cookies in the request—**even if the request comes from a malicious site!**

**How CSRF Exploits This Weakness**

Let's say you're logged into your **social media account**, and the website uses only cookies for authentication. Now, you visit a malicious website that contains this hidden code:

**<img src="https://socialmedia.com/update-profile?email=hacker@gmail.com" style="display:none;">**

Here's what happens step by step:

1. **You're logged into your social media account**, and your browser has an authentication cookie.
2. You visit the malicious website.
3. That site loads a hidden image, but instead of a real image, it's a request to **change your email to the hacker's email.**
4. Since you're already logged in, **your browser automatically includes the authentication cookie** in the request.
5. **The social media site thinks YOU made this request** and updates your email.
6. **The hacker can now reset your password and take over your account!**

**Why This Works**

* The website **trusts cookies alone** and doesn’t check where the request is coming from.
* Your browser **automatically includes cookies** in every request, even if it was triggered from a different site.
* The website doesn’t require **extra verification** (like CSRF tokens) before making sensitive changes.

**How CSRF Tokens Stop This**

If the website had used a **CSRF token**, the hacker’s request would fail because:

* The **malicious request wouldn’t have the correct CSRF token** (since it's only available in the real user’s session).
* The website checks for a valid **CSRF token before making changes**.

**Bottom Line:** Cookies alone are **not enough** for authentication. Websites must use extra security like **CSRF tokens, re-authentication, or 2FA** to prevent unauthorized actions.

# References

1. <https://portswigger.net/web-security/csrf>
2. <https://www.kaspersky.com/resource-center/definitions/cookies>
3. <https://brightsec.com/blog/csrf-token/#what-is-csrf-token>