Everything you didn't want to know about Cross-site Request Forgery in Django

— DjangoCon EU 2022 —

Andreas Schmitz

2022-09-23

$ whoami

Andreas Schmitz

- Lead Software Development Engineer at …
  - ✉️ me@andreas.earth
  - 🌍 www.andreas.earth
- … wirbauen.digital
  - 🏗️ Construction-Tech Startup
  - 📍 Cologne, Germany
  - 🚀 Trying to digitize the Construction Industry

wirbauen
.digital

Cross-site… what?

Cross-site… what?!

Do these questions sound familiar?

771 votes — What is a CSRF token? What is its importance and how does it work?

5 answers

I am writing an application (**Django**, it so happens) and I just want an idea of what actually a

What is a CSRF token? What is its importance and how does it work?

Asked 11 years, 4 months ago Modified 11 months ago Viewed 484k times

771

I am writing an application (Django, it so happens) and I just want an idea of what actually a "CSRF token" is and how it protects the data.

Is the post data not safe if you do not use CSRF tokens?

python django

Salvatorelab **11.2k** answered May 9, 2013 at 9:10

272 votes

✓ Accepted

Django Rest Framework remove csrf

DRF's SessionAuthentication uses Django's session framework for authentication which requires **CSRF** to be checked. … This means that only authenticated requests require **CSRF** tokens and…

django django-rest-framework csrf django-csrf

Rahul Gupta **43.9k** answered Jun 16, 2015 at 18:50

The Dangers

Past CSRF Attacks in the Industry

Forbes

CYBERSECURITY • EDITORS' PICK

Meetup Security Flaws Exposed 44 Million Members To Data Loss And Payment Threat

Davey Winder Senior Contributor ⓘ
Co-founder, Straight Talking Cyber

Aug 3, 2020, 09:06am EDT

[...]

Mitigating against untrusted input exploits

Erez Yalon has some advice for all site operators, so they don't end up being the next subject of such a report. "My first suggestion would be to educate the developers and make sure they understand the mechanisms of these attacks," Yalon says, "you can't defend against something you don't understand."

(8)

Terminology

Origin

- **Scheme**, **Host**, and **Port** (Tuple)
- **Same-origin**
  - Tuple of two URLs match
- **Cross-origin**
  - Tuple of two URLs DO NOT match

https://2022.djangocon.eu/about/credits/

URL	Same-Origin
https://2022.djangocon.eu/home/	✔
https://2022.djangocon.eu:**8443**/home/	✘
http://2022.djangocon.eu/about/credits/	✘
https://**2021**.djangocon.eu/about/credits/	✘
https://**2022.djangocon.us**/about/credits/	✘

Terminology

- Effective Top-level Domain (eTLD)
  - a.k.a. Public Suffixes
  - Examples: **.com**, **.pt**, **.co.uk**, **.github.io**, **.pvt.k12.ma.us**
  - List changes over time
- Registrable domain
  - preceding name + eTLD
  - Examples: **example.com**, **djangocon.eu**, **abc123.github.io**

Terminology

Site

- Registrable domain of an URL
  - E.g. site of https://2022.djangocon.eu is djangocon.eu
- **Same-site**
  - If two sites are *identical*
- **Cross-site**
  - If two sites are *different*

https://2022.djangocon.eu/about/credits/

URL	**Same-Site**
https://2022.djangocon.eu/home/	✔
https://2022.djangocon.eu:8443/home/	✔
http://2022.djangocon.eu/about/credits/	✔
https://2022.djangocon.eu/about/credits/	✔
https://2022.**djangocon.us**/about/credits/	✘

Web Security Concepts

- Same-origin Policy (SOP)
  - Limits cross-origin interactions
  - Typically **allowed**: cross-origin **writes** and **embedding**
  - Typically **blocked**: cross-origin **reads**
  - Examples
    - Cross-origin image **embedding** is allowed, **reading** image data is blocked
    - Cross-origin form submission (**write**) is allowed

Web Security Concepts

- Cross-origin Resource Sharing (CORS)
  - Allows opening up Same-origin Policy (JavaScript cross-origin read)
  - Server specifies allowed origins (HTTP headers)
  - Example
    - Cross-origin **read** of image data

Cross-site Request Forgery

Cross-site Request Forgery is a **class of web vulnerabilities** that allows an **attacker** to **trick a victim** into performing **unintended actions**.

CSRF Attack Flow Example

Alice

1 **Claims** best sleeping place

https://spas.org

CSRF Vulner-able

S.P.A.S.

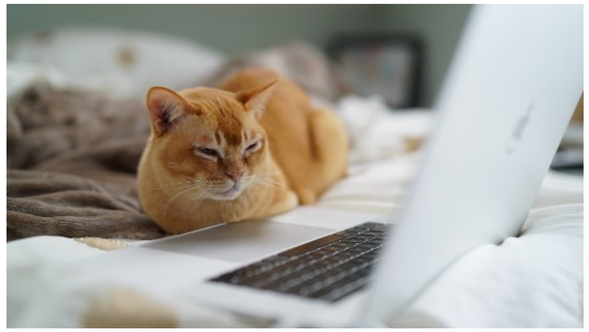
„Sleeping Place Allocation Service"

2 **Sends** link to malicious page

On Sale
„Extra Fluffy Blankies"

3 **Opens** ad page

4 **Exploits vulnerability** undoing Alice's claim



Mallory

5 **Claims** best sleeping place

20% Off

Extra Fluffy Blanky

BUY NOW

13:37:42

Ad

https://all-the-fluff.com

Demo

— CSRF Attack in Action —

Vulnerability Details

Why does this work?

- (Cross-origin) Requests **automatically include** associated credentials
  - **Session cookies**
  - Cached HTTP Basic Authentication
  - Client Certificates
- No unpredictable request parameters
- Backend cannot distinguish **legitimate** from an **illegitimate** request

Vulnerability Details

Which requests are vulnerable?

"Simple Requests"

- **GET** endpoints that perform **state changes**
- **POST** endpoints that accept
  - **application/x-www-form-urlencoded**, **multipart/form-data** or **text/plain**
- **PATCH**, **DELETE**, and **PUT** endpoints in some cases
  - Support in some frameworks via hidden input fields
  - HTML5 draft support

Vulnerability Details

What about XMLHttpRequest?

XMLHttpRequest

- Blocked by Same-origin Policy (SOP)
- Requires Cross-origin Resource Sharing (CORS)
  - **Careful**: only allow trusted origins
    - Trusting an origin defeats most CSRF protections
    - Avoid things like Access-Control-Allow-Origin: *

Vulnerability Details

What about the fetch API?

fetch

- Vulnerable with **mode: "no-cors"**
  - Only allows "Simple requests"
    - Same capabilities as GET and normal forms (see two slide ago)
    - e.g. Content-Type application/json, text/xml etc. are not allowed
- Other modes are protected by the Same-origin Policy

Vulnerability Details

What is not vulnerable?

- Any kind of credential that is not automatically attached
  - e.g. Bearer Tokens (JWT, …)
  - PSA: Don't save tokens in local storage
    - **Vulnerable to Cross-site Scripting!**
    - Can be catastrophic for JWT (no way to invalidate)
- User is not logged in
  - Except Login CSRF

Code

— CSRF Attack in Action —

Prevention Measures

Prevention Measures

How to do defend against CSRF attacks

Common techniques

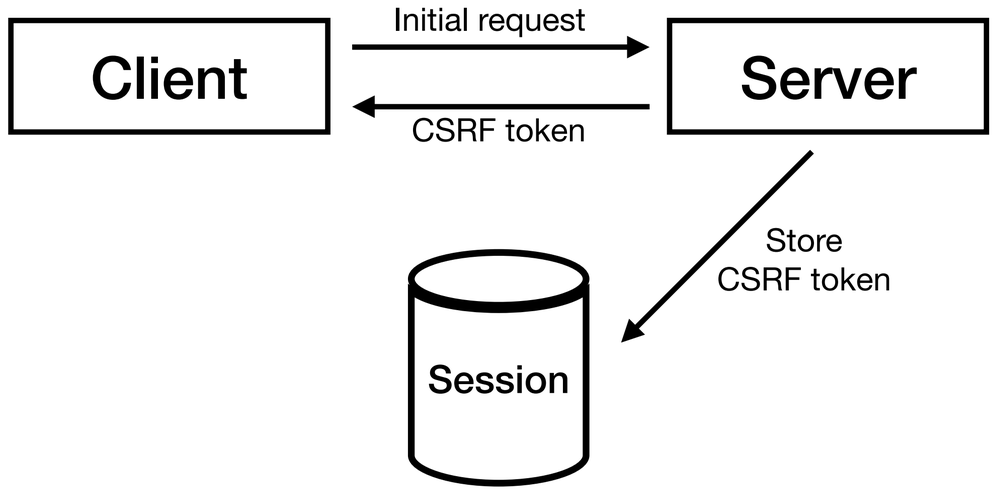
1. Synchronizer Token Pattern
2. Double Submit Cookie
3. SameSite Cookie Attribute
4. Validating Origin and Referer [sic] Headers

Synchronizer Token Pattern

The stateful approach

On first visit, the **server**…

- generates **unique**, **secret** and **unpredictable** CSRF token
- stores the token in the session
- returns the token in the response
  - via form field, header, meta tag, …

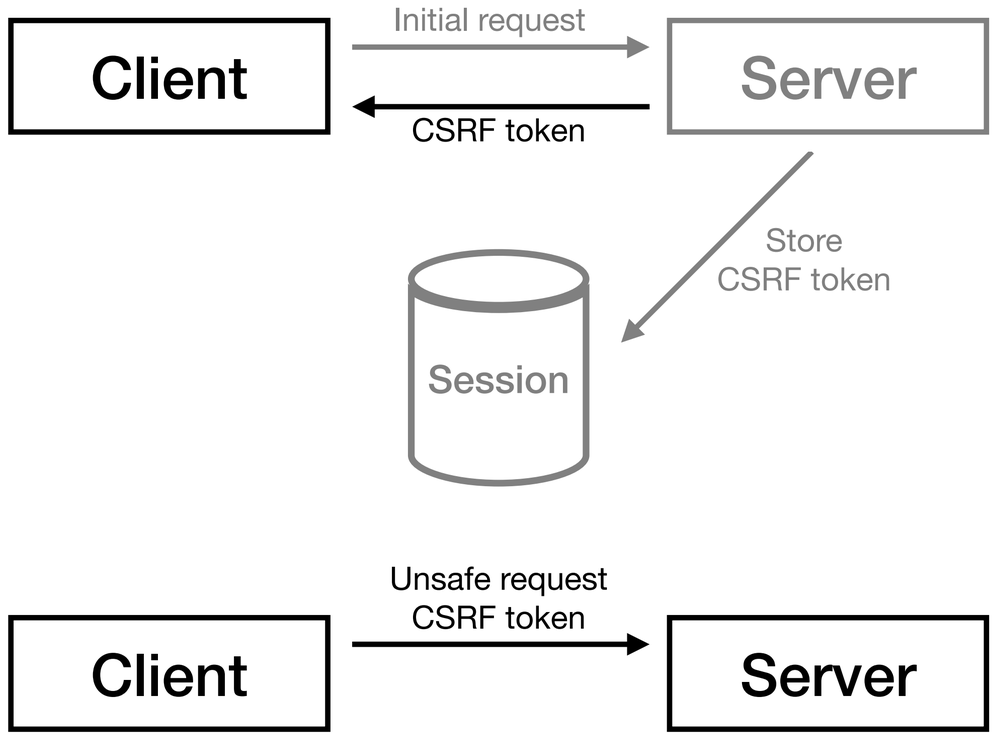


Synchronizer Token Pattern

The stateful approach

For "unsafe" requests, the **client** …

- extracts the CSRF token from response
- attaches the CSRF token to state-changing request
  - via header, form field, …

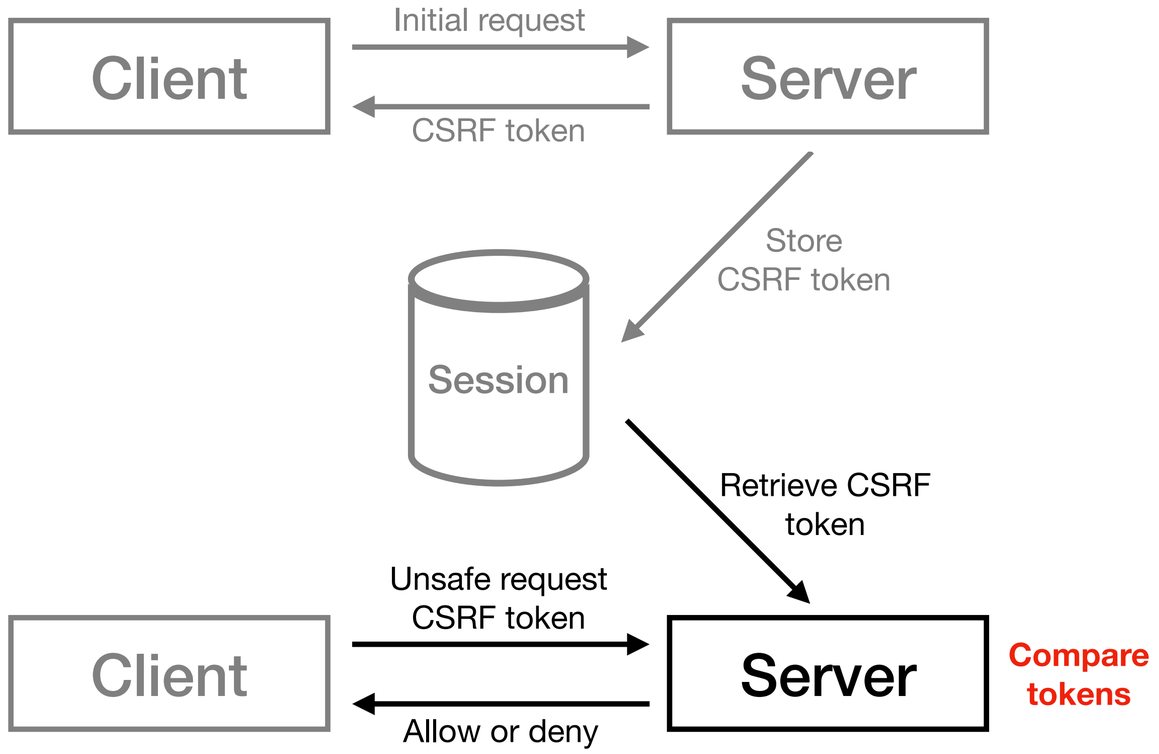


Synchronizer Token Pattern

The stateful approach

For "unsafe" requests, the **server**…

- checks existence of the **token** in the **request**
- compares it to the **token** in the **session**
- aborts if they do not match



Synchronizer Token Pattern

The stateful approach

- Prevents CSRF. How?
  - Attacker cannot create a valid request without the token
  - Token cannot be guessed or retrieved
- Variations
  - Per-session token
  - Per-request token
    - More secure, but degrades usability (e.g. breaks back button)

Double Submit Cookie

The stateless approach

- Similar to Synchronizer Token pattern
  - Difference: Cookie instead of Session
- Attacker cannot read or change the cookie
- Less secure
  - Session and token are not tied together
  - Sub-domains takeover allows overwriting cookie

Double Submit Cookie

The stateless approach

- For improved security
  - Sign/Encrypt cookie (e.g. HMAC)
  - Prevents overwriting from sub-domain without key

SameSite Session Cookie

An overdue improvement

- "New" cookie attribute (introduced 2016)
- Allows restricting cookies to same-site context
  - Powerless against cross-origin, same-site attacks
- Can prevent CSRF when set for session cookies
- Good browser support (94.00% globally)

SameSite Session Cookie

Configuration

- **SameSite=Strict**
  - Cookie is only sent in same-site context
- **SameSite=Lax** (recommended)
  - Cookie is sent in safe cross-site context, e.g. link or GET form
  - Default, if attribute is missing (modern browsers)
- **SameSite=None**
  - Cookie is sent in all contexts
  - Requires secure attribute

There's more…

Other techniques that (partially) work

- Checking Origin HTTP header
- Checking Referer [sic] HTTP header
- One Time Tokens
- Re-Authentication
- Captchas
- …

CSRF Prevention Measures

Recommendation

Combine multiple measures

- Implement **Double Submit Cookie** or **Synchronizer Token Pattern**
- Configure **SameSite** cookie attribute as tight as possible
- Validate **Origin** and **Referer** [sic] headers

CSRF Prevention in Django

CSRF Protection
Built-in & enabled by default

CSRF Prevention in Django

Batteries included

- Double submit cookie (**CSRF_USE_SESSIONS=False**, default)
- Synchronizer token pattern (**CSRF_USE_SESSIONS=True**)
- SameSite: Lax by default
  - csrftoken and session cookie
- Origin and Referer [sic] validation (**CSRF_TRUSTED_ORIGINS**)

CSRF Prevention in Django

The components

- **CsrfViewMiddleware**
  - Token creation
  - CSRF token session and cookie handling
  - Token enforcement and check in unsafe requests
  - Origin and Referer [sic] validation

CSRF Prevention in Django

The components

- **django.middleware.csrf.get_token()**
  - Creates new CSRF token or returns existing one
- **django.middleware.csrf.rotate_token()**
  - Replaces the existing CSRF token with a new one
  - Called on login (contrib.auth)

CSRF Prevention in Django

The components

- **{% csrf_token %}** template tag
  - Renders CSRF token in hidden input field
  - Token value is masked (BREACH attack prevention)

CSRF Prevention in Django

Bird's-eye View

- Activate **CsrfViewMiddleware** (default)
- Don't perform state changes in safe requests (GET, HEAD, …)
- **HTML Forms**: Add **{% csrf_token %}** to forms tag in template
- **XHR/fetch**: Extract token (cookie or body) and set X-CSRFToken header
- More details "How to use Django's CSRF protection"

CSRF Prevention in Django

Limitations

- CSRF token cookie is not signed or encrypted
- Subdomains can circumvent CSRF protection
- Django's Documentation on this: "CSRF Limitations"

CSRF Prevention in Django

Django REST Framework

- **TokenAuthenticationBackend**: Not required
  - No automatically attached credentials
- **SessionAuthenticationBackend**: Required and enforced
  - Only authenticated requests require CSRF token
  - Important: Add **@csrf_protect** decorator to custom login views
    - Prevents Login CSRF attack

CSRF Prevention in Django and Single-page Applications

CSRF Prevention in Django

Single-Page Application Setup: Frontend

- **Problem**: Headless SPAs have no initial response to extract CSRF token from
- **Solution**: Request CSRF token on **page load**, **log-in** and **log-out**
  - View that needs to be called anyway (e.g. /accounts/me/)
  - Dedicated view to request token (e.g. /csrf)

CSRF Prevention in Django

Single-Page Application Setup: Frontend

- Set X-CSRFToken header for unsafe requests
- Frontend requests must **include credentials**
  - XMLHttpRequest
    - **withCredentials = true**
  - fetch
    - **credentials: "include"**

CSRF Prevention in Django

Single-Page Application Setup: Backend

- Synchronizer Token Pattern (**CSRF_USE_SESSIONS=True**)
  - Include CSRF token in response body
  - JSON should be fine (JSON Hijacking seems to not be a thing anymore)
- Double Submit Cookie (**CSRF_USE_SESSIONS=False**)
  - Include CSRF token in response body or extract from cookie

CSRF Prevention in Django

Single-Page Application Setup: Backend

Cross-origin, **same-site** SPAs (e.g. http://web.foo.pt and http://api.foo.pt)

- Configure trusted Origins
  - **CSRF_TRUSTED_ORIGINS** (frontend and backend origins)
- Install and configure **django-cors-headers**
  - **CORS_ALLOWED_ORIGINS** (frontend)
  - **CORS_ALLOW_CREDENTIALS**=True
- For Double Submit Cookie
  - Configure **CSRF_COOKIE_DOMAIN** (e.g. ".foo.pt")

CSRF Prevention in Django

Single-Page Application Setup: Backend

- SPA and backend should be **same-site**
- **Cross-site** setup (not recommended)
  - Set SameSite=None for the session (and csrftoken) cookies
  - Requires CSRF token in response body
    - Can't extract CSRF token from cookie (cross-site)
  - Modern cross-site tracking prevention can interfere

Demo & Code

— SPA & CSRF Prevention in Action —

CSRF Prevention in Django

Most Important Settings

- **CSRF_USE_SESSIONS**
  - **True**: Synchronizer Token Pattern
  - **False**: Double Submit Cookie (default)
- **CSRF_COOKIE_HTTPONLY**
  - **False**: Allows access to CSRF token in cookie via JavaScript (default)
  - **True**: Can't access CSRF token in cookie via JavaScript
  - True offers no security benefit

CSRF Prevention in Django

Most Important Settings

- **CSRF_TRUSTED_ORIGINS**
  - List of trusted origins for unsafe requests
  - Origin and Referer [sic] header (if present) values must match Host header
  - e.g. Single-page application and backend host

CSRF Prevention in Django

Most Important Settings

- **CSRF_COOKIE_SAMESITE** and **SESSION_COOKIE_SAMESITE**
  - Most cases leave the default **"Lax"**
  - **"Strict"** high-security or if linking from another site is not necessary
- **CSRF_COOKIE_SECURE** and **SESSION_COOKIE_SECURE**
  - Set to **True** on production system
  - Cookie is only sent when HTTPS is used

CSRF Prevention in Django

Most Important Settings

- **CSRF_COOKIE_DOMAIN** and **SESSION_COOKIE_DOMAIN**
  - **None** by default (HostOnly cookie, restricted to current domain)
  - Only set, if application needs to be reached via a sub-domain
- **SESSION_COOKIE_PATH** and **CSRF_COOKIE_PATH**
  - Path of your Django application
  - "/" by default
  - Only set, if application root is not "/" (e.g. shared hosting)

CSRF Prevention in Django

Packages & Resources

- **django-cors-headers**
  - Allows configuring Cross-Origin Resource Sharing (CORS)
- **django-ai-kit-auth**
  - Opinionated Session-based Authentication setup for Django REST Framework & React
- **OWASP CSRF**
  - Overview & CSRF Prevention Cheat Sheet

Slides & All Resources

andreas.earth/s/djangocon-22

References

(1) https://nvd.nist.gov/vuln/search/results?form_type=Basic&results_type=overview&query=csrf&search_type=all&isCpeNameSearch=false

(2) https://portswigger.net/daily-swig/horde-webmail-contains-zero-day-rce-bug-with-no-patch-on-the-horizon

(3) https://www.theregister.com/2022/06/30/jenkins_plugins_security_advisories/

(4) https://nakedsecurity.sophos.com/2014/12/05/all-paypal-accounts-were-1-click-away-from-hijacking/

(5) https://securityaffairs.co/wordpress/81219/hacking/facebook-csrf-flaw.html

(6) https://portswigger.net/daily-swig/chain-of-vulnerabilities-led-to-rce-on-cisco-prime-servers

(7) https://portswigger.net/daily-swig/reddit-patches-csrf-vulnerability-that-forced-users-to-view-nsfw-content

(8) https://www.forbes.com/sites/daveywinder/2020/08/03/meetup-security-flaws-exposed-44-million-members-to-data-loss-and-payment-threat-checkmarx-research/?sh=39eee38112d6

Image Attributions

- Image Source: pexels.com
  - Images are links to the source

Thank you!