Engineer

CSRF

inducir al navegador a enviar una petición no deseada

Qué es

CSRF, abreviatura de cross-site request forgery o falsificación de peticiones entre sitios, es el ataque en el que una página que la víctima abre hace que su propio navegador envíe una petición a otro sitio donde tiene una sesión iniciada. El navegador adjunta las cookies de ese sitio como siempre, por lo que la petición llega con la sesión de la víctima y parece algo que esta decidió hacer. Common Weakness Enumeration lo cataloga como CWE-352 y describe el fallo desde el lado del servidor: la aplicación no comprueba con suficiente rigor si la petición fue realizada intencionadamente por el usuario del que parece proceder, o no puede hacerlo.

La guía de prevención de OWASP enumera los puntos de partida del ataque, desde un sitio web malicioso hasta un correo electrónico, una publicación de blog, un mensaje instantáneo o un programa, y lo que puede conseguir: una transferencia de fondos, un cambio de contraseña, una compra o la concesión de privilegios. Cada uno es una acción en el sitio, un cambio de estado, y ahí sitúa la guía sus defensas: un token en cada petición que cambie el estado. CWE explica que los permisos de la víctima determinan el alcance del ataque, porque el atacante toma prestada su identidad. Contra un administrador, el resultado puede ser el control completo de la aplicación web.

CWE trata CSRF como una debilidad compuesta, que solo existe cuando confluyen otras. Entre ellas están un error de validación del origen, CWE-346, y el proxy involuntario o «representante confundido», CWE-441: el navegador posee las credenciales de la víctima y las utiliza en nombre de una página que no tiene ninguna. Eso distingue CSRF de cross-site scripting, que CWE incluye como una debilidad del mismo nivel. Con XSS, el código del atacante se ejecuta dentro del sitio objetivo; con CSRF, permanece en la página del atacante y solo envía peticiones. También lo distingue de CORS, que gobierna qué puede leer una página de otro origen. OWASP señala que un formulario HTML sencillo envía una

petición simple, que no necesita consulta previa, y que los navegadores permiten esas peticiones a cualquier origen. El propio título de CWE desarrolla el término como Cross-Site Request Forgery, y la entrada recoge otros nombres para lo mismo, entre ellos XSRF y Session Riding.

Utilice el concepto en una decisión

Aplique «¿Qué es CSRF?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Cuentas, contraseñas y permisos organizados para que cada persona tenga exactamente el acceso que necesita y lo pierda el mismo día que deja la empresa.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. 1.000 reglas RBAC para servidores Linux — 1,000 reglas de acceso escritas para cerrar una brecha de seguridad

1.000 reglas RBAC para servidores Linux Póngase en contacto →

Por qué importa

CSRF ha permanecido en las dos listas de debilidades más conocidas. CWE-352 estuvo en el CWE Top 25 de 2009, 2010 y 2011, y en todas las ediciones de 2019 a 2025. OWASP le dedicó una categoría propia en los Top 10 de 2007 y 2010, como A5, y de 2013, como A8. Desde 2021 forma parte de A01, Broken Access Control, que encabeza la edición de 2025. OWASP informa de que todas las aplicaciones analizadas presentaban alguna forma de control de acceso defectuoso y menciona CWE-352 entre sus debilidades destacadas.

El ataque es difícil de erradicar porque el navegador hace el trabajo. El borrador Fetch Metadata del W3C parte de ahí: es fácil convencer a los navegadores de que envíen peticiones a los endpoints de un sitio y de que incluyan las credenciales implícitas del usuario, como cookies o una posición privilegiada en una intranet, así que las aplicaciones tienen que ser muy cuidadosas con el comportamiento de esos endpoints, y resulta difícil serlo. CWE señala que casi cualquier elemento HTML puede iniciar el ataque, incluidas las etiquetas de imagen y los enlaces. Sus ejemplos observados parecen una lista de acciones administrativas realizadas desde una página ajena: cuentas de usuario añadidas mediante una URL en una etiqueta `img` (CVE-2004-1703), privilegios de administrador obtenidos de la misma forma (CVE-2004-1842) y una interfaz web que permitía a una petición falsificada cambiar contraseñas o detener una máquina virtual (CVE-2009-3759).

Los escáneres ayudan poco. CWE indica que actualmente es difícil detectar CSRF de forma fiable con técnicas automatizadas, porque cada aplicación tiene su propia política implícita sobre qué peticiones puede provocar alguien ajeno y cuáles requieren un alto grado de certeza de que el usuario quiso realizarlas. Una herramienta no puede saber que un formulario es inofensivo y el siguiente mueve dinero. Eso convierte CSRF en una cuestión de diseño: decidir, petición por petición, qué demuestra la intención del usuario.

Los navegadores han evolucionado a favor de la defensa. La guía de OWASP dice que los principales navegadores actuales aplican SameSite=Lax a las cookies de forma predeterminada, lo que evita que se envíen en la mayoría de las peticiones `POST` entre sitios, y que todos los navegadores principales envían las cabeceras Fetch Metadata desde marzo de 2023. Ambas medidas ayudan y ninguna resuelve por sí sola el problema. El borrador de cookies que define SameSite considera Lax una defensa en profundidad razonable contra ataques que dependen de métodos no seguros como `POST`, pero no una defensa robusta contra CSRF como categoría, y aconseja que ni siquiera el modo Strict sea toda la defensa de un sitio. OWASP añade que los usuarios de navegadores antiguos o integrados pueden recibir cookies que se comporten como si no se hubiera establecido ningún valor SameSite.

Cómo se aplica en la práctica

Considere una aplicación web en `app.example.com` donde un usuario con sesión iniciada cambia la dirección de correo electrónico de su cuenta mediante el envío de un formulario por POST a `/account/email`. La cookie de sesión se establece con `SameSite=None; Secure` porque el mismo panel también se incrusta en el portal de un socio, y None es el valor que permite al navegador enviar una cookie en un contexto entre sitios. Un atacante publica una página en `prizes.example.net` y consigue que la víctima la abra:

```
<form action="https://app.example.com/account/email" method="post">
  <input type="hidden" name="email" value="mallory@example.net">
</form>
<script>document.forms[0].submit();</script>
```

El navegador envía el formulario en cuanto se carga la página y, como la cookie lo permite, la sesión va con él:

```
POST /account/email HTTP/1.1
Host: app.example.com
Origin: https://prizes.example.net
Sec-Fetch-Site: cross-site
Content-Type: application/x-www-form-urlencoded
Cookie: session=...

email=mallory%40example.net
```

Para el servidor, es un usuario con sesión iniciada que cambia su dirección. Si no se comprueba nada más, la dirección cambia y un mensaje para restablecer la contraseña enviado a esa nueva dirección pone la cuenta en manos del atacante.

Dos cabeceras de esa petición pertenecen al navegador, no al atacante. Fetch Standard incluye `Origin` entre las cabeceras de petición prohibidas, junto con todas aquellas cuyo nombre empieza por `Sec-`, así que un script de la página no puede establecerlas ni modificarlas. `Origin` indica el origen que envió la petición, y el borrador Fetch Metadata define cuatro valores para `Sec-Fetch-Site`: `cross-site`, `same-site`, `same-origin` y `none` para una navegación iniciada por el usuario al escribir una dirección o elegir un marcador. Una comprobación puede apoyarse en ambas y seguir recurriendo a un token para los navegadores que no envían ninguna. Este es un esquema en Python:

```
import hmac
import secrets

SAFE_METHODS = {"GET", "HEAD", "OPTIONS", "TRACE"}
OUR_ORIGIN = "https://app.example.com"


def new_csrf_token(session: dict) -> str:
    session["csrf_token"] = secrets.token_urlsafe(32)
    return session["csrf_token"]


def csrf_ok(method: str, headers: dict, form: dict, session: dict) -> bool:
    if method in SAFE_METHODS:
        return True
    if headers.get("Sec-Fetch-Site") == "cross-site":
        return False
    origin = headers.get("Origin")
    if origin is not None and origin != OUR_ORIGIN:
        return False
```

```
    expected = session.get("csrf_token", "").encode()
    sent = form.get("csrf_token", "").encode()
    return bool(expected) and hmac.compare_digest(sent, expected)
```

El token procede de `secrets.token_urlsafe(32)`. La documentación de Python dice que el módulo `secrets` utiliza la fuente de aleatoriedad más segura que proporciona el sistema operativo y que se considera que 32 bytes, 256 bits, son suficientes. El servidor coloca el token en un campo oculto de cada formulario que renderiza y conserva una copia en la sesión. Nunca lo pone en una cookie, una URL o un registro, los lugares que la guía de OWASP descarta para un token sincronizador. `hmac.compare_digest` está diseñado para no detenerse en el primer byte diferente, de modo que el tiempo de comparación no revela qué parte de una conjetura era correcta. Ambas partes se codifican primero como bytes porque la función solo acepta cadenas de texto si son exclusivamente ASCII. Una petición sin cabecera `Origin` pasa a la comprobación del token, y `Origin: null` falla, porque solo se acepta el origen exacto.

En una aplicación real, utilice la protección del framework antes de escribir la suya; es el primer punto de la lista de OWASP. El middleware de Django, por ejemplo, exige tanto su cookie CSRF como un campo `csrfmiddlewaretoken` en todos los métodos salvo `GET`, `HEAD`, `OPTIONS` y `TRACE`, responde con 403 cuando falta alguno o es incorrecto y comprueba `Origin` frente al host y la configuración `CSRF_TRUSTED_ORIGINS`. Cuando no sea necesario incrustar la aplicación en ningún otro sitio, la cookie de sesión también puede enviarse como `__Host-session=...; Path=/; Secure; SameSite=Lax`. OWASP señala que una cookie con el prefijo `__Host-` no puede sobrescribirse desde otro subdominio.

La configuración del navegador es una capa, no la defensa. El caso sobre el refuerzo de la aplicación con CSP basada en nonces señala que el middleware proxy establece HSTS y cookies SameSite junto con Content Security Policy y que los permisos se vuelven a comprobar en el servidor en cada petición, mientras los controles del frontend solo cuentan para la experiencia de usuario.

Dónde surgen los problemas

Cambios de estado mediante GET. SameSite=Lax sigue enviando la cookie en una navegación entre sitios de nivel superior con un método seguro, así que un enlace que pulse la víctima, o una página que dirija toda la ventana a `https://app.example.com/account/delete?confirm=yes`, llega con la sesión iniciada. OWASP dice que es la forma más común de que fallen en la práctica las defensas

basadas en SameSite, y Django construye su protección sobre la misma premisa: `GET` y los demás métodos seguros no deberían tener efectos secundarios. Mantenga `GET` , `HEAD` , `OPTIONS` y `TRACE` sin efectos secundarios y sitúe cada cambio tras un método cubierto por sus comprobaciones.

SameSite como única defensa. El borrador de cookies explica por qué Lax no basta por sí solo: un atacante todavía puede abrir una ventana nueva o iniciar una navegación de nivel superior para que una petición cuente como del mismo sitio, y el prerenderizado puede hacer lo mismo sin que el usuario se dé cuenta. OWASP añade que SameSite considera el dominio registrable en lugar del origen completo. Una petición desde cualquier host de example.com cuenta como del mismo sitio para `app.example.com` , así que un subdominio olvidado del que alguien tome el control ya está dentro de la barrera. Trate SameSite como segunda capa y decida en el servidor mediante un token o una comprobación del origen.

Un token que el atacante puede colocar. En el patrón ingenuo de doble envío, el servidor solo comprueba que una cookie y un campo del formulario lleven el mismo valor aleatorio. OWASP lo desaconseja ahora porque cualquiera que pueda escribir cookies en el dominio, mediante un subdominio vecino vulnerable, la toma de control del DNS o una cookie inyectada por HTTP sin cifrar, puede fijar ambas partes. La entrada de CWE sigue incluyendo la cookie de doble envío simple entre sus mitigaciones, así que las dos fuentes discrepan y hay que seguir la advertencia de OWASP. Guarde el token en la sesión del servidor o utilice la cookie de doble envío firmada de OWASP, vinculada a la sesión mediante un HMAC.

Cross-site scripting en el mismo sitio. Un script que se ejecuta dentro de sus propias páginas puede leer el token del formulario y enviar una petición que es del mismo origen según todos los criterios. La guía de OWASP lo dice claramente: XSS «puede derrotar todas las técnicas de mitigación de CSRF». CWE y Django dicen lo mismo, y el ejemplo más conocido de CWE muestra ambos ataques trabajando juntos. El gusano Samy utilizó XSS para instalarse en un perfil de MySpace y añadir al atacante como amigo; los amigos que veían el perfil sufrían después el mismo cambio en los suyos. Como ninguna víctima eligió hacer ese cambio, CWE cuenta CSRF como una causa raíz. Trate cada hallazgo de XSS también como un hallazgo de CSRF y corríjalo primero.

Una comprobación del origen que acepta demasiado. Una prueba como `origin.startswith("https://app.example.com")` también acepta `https://app.example.com.attacker.example`, la trampa sobre la que advierte la guía de OWASP. El valor `null` es otra. Según Fetch Standard, un formulario enviado desde una página cuya política de referencia sea `no-referrer` llega con `Origin: null`, y OWASP añade que las redirecciones entre orígenes y los marcos aislados también pueden producirlo. Compare el origen completo con una lista fija de cadenas exactas y trate una cabecera ausente o `null` como desconocidos, nunca como su propio origen.

Confiar demasiado en la consulta previa. Una cabecera de petición personalizada es una defensa sólida para una API, porque los navegadores envían una consulta previa antes de cualquier petición entre orígenes que la lleve. Solo funciona si la API rechaza todo lo demás. OWASP advierte que una API JSON que también acepta `text/plain` puede alcanzarse mediante un formulario, porque ese es uno de los tres tipos de contenido que admite una petición simple, y que una política CORS que permite credenciales desde todos los subdominios mediante una expresión regular deja enviar la cabecera personalizada a cualquiera que tome el control de un subdominio. Rechace los tipos de contenido que el endpoint no espere y enumere los orígenes exactos que pueden enviar peticiones con credenciales.

Olvidar el formulario de inicio de sesión. El CSRF de inicio de sesión funciona al revés: el atacante inicia la sesión de la víctima en la cuenta del propio atacante. En una tienda, según el ejemplo de OWASP, la víctima introduce después los datos de su tarjeta y el atacante puede comprar cosas con ellos más tarde. La documentación de Django incluye el CSRF de inicio de sesión entre los ataques que cubre su protección. Proteja el formulario con un token vinculado a una sesión previa, destrúyala e inicie una nueva cuando el usuario haya accedido, como aconseja OWASP.

Peticiones construidas a partir de entradas del atacante. El CSRF del lado del cliente ocurre cuando el JavaScript de la propia aplicación construye una petición a partir de entradas que controla el atacante, como la URL, el nombre de la ventana o un `postMessage`. El script añade el token por sí mismo y la petición es del mismo sitio, por lo que ni los tokens ni SameSite lo detienen; OWASP lo describe como el regreso del representante confundido. Mantenga los destinos y los parámetros de las peticiones fuera del alcance de las entradas controladas por el atacante o elíjalos de una lista fija en el código.

Lecturas adicionales

- CWE-352: falsificación de peticiones entre sitios (CSRF) — la entrada sobre la debilidad: estructura compuesta, consecuencias, mitigaciones y CVE observadas.
- Serie de guías de OWASP: prevención de la falsificación de peticiones entre sitios — tokens, Fetch Metadata, comprobaciones del origen, límites de SameSite y CSRF de inicio de sesión.
- W3C: cabeceras de petición Fetch Metadata — la cabecera `Sec-Fetch-Site` y los cuatro valores que puede enviar un navegador.
- Borrador del IETF: cookies y mecanismo de gestión del estado HTTP (draft-ietf-httpbis-rfc6265bis) — el atributo SameSite y por qué Lax es defensa en profundidad, no un remedio completo; sigue siendo un Internet-Draft.
- Documentación de Django: protección contra la falsificación de peticiones entre sitios — la protección incorporada de un framework, sus supuestos y cómo la eluden los subdominios.
- OWASP Top 10:2025: A01, control de acceso defectuoso — dónde se sitúa CWE-352 en la edición de 2025.

Glosario

Elija según sus restricciones reales

Describa la ruta de despliegue, los riesgos operativos y quiénes se encargan de las guardias. Podemos definir el alcance de un cambio con comprobaciones, una vía de recuperación y responsabilidades claras.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es CSRF?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/csrf/