Engineer

CSRF

einen Browser zu einer ungewollten Anfrage verleiten

Was es ist

CSRF, kurz für Cross-Site Request Forgery, ist ein Angriff, bei dem eine Seite, die das Opfer gerade öffnet, dessen eigenen Browser eine Anfrage an eine andere Website senden lässt, bei der es angemeldet ist. Der Browser hängt wie immer die Cookies dieser Website an. Die Anfrage kommt deshalb mit der Sitzung des Opfers an und sieht wie eine von ihm gewählte Handlung aus. Die Common Weakness Enumeration führt das als CWE-352 und beschreibt den Fehler aus Sicht des Servers: Die Anwendung prüft nicht ausreichend oder kann nicht ausreichend prüfen, ob eine Anfrage von dem scheinbaren Absender absichtlich gestellt wurde.

OWASPs Merkblatt zur Vermeidung nennt mögliche Ausgangspunkte des Angriffs: eine bösartige Website, eine E-Mail, einen Blogbeitrag, eine Sofortnachricht oder ein Programm. Es nennt auch mögliche Ergebnisse: eine Überweisung, ein geändertes Passwort, einen Kauf oder die Vergabe von Rechten. Jede dieser Handlungen verändert den Zustand der Website. Genau dort setzt der Schutz des Merkblatts an: ein Token bei jeder zustandsändernden Anfrage. Laut CWE bestimmen die Berechtigungen des Opfers die Reichweite des Angriffs, weil der Angreifer dessen Identität ausleiht. Bei einem Administrator kann das zur vollständigen Kontrolle über die Webanwendung führen.

CWE behandelt CSRF als zusammengesetzte Schwäche, die nur beim Zusammentreffen mehrerer anderer besteht. Dazu gehören ein Fehler bei der Origin-Validierung, CWE-346, und der unbeabsichtigte Proxy oder „Confused Deputy“, CWE-441: Der Browser hält die Anmeldedaten des Opfers und setzt sie im Auftrag einer Seite ein, die selbst keine besitzt. Das unterscheidet CSRF von Cross-Site-Scripting, das CWE als gleichrangige Schwäche aufführt. Bei XSS läuft der Code des Angreifers innerhalb der Zielwebsite. Bei CSRF bleibt er auf der Angreiferseite und sendet nur Anfragen. Es unterscheidet CSRF auch von CORS,

das bestimmt, was eine Seite von einer anderen Origin lesen darf. OWASP weist darauf hin, dass ein schlichtes HTML-Formular eine einfache Anfrage ohne Preflight sendet und Browser solche Anfragen an jede Origin erlauben. CWEs eigener Titel schreibt den Begriff als Cross-Site Request Forgery aus. Der Eintrag nennt weitere Bezeichnungen für dasselbe, darunter XSRF und Session Riding.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist CSRF?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Konten, Passwörter und Berechtigungen werden so geordnet, dass alle genau den beruflich erforderlichen Zugriff haben und ihn am Tag ihres Ausscheidens verlieren.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 1.000 RBAC-Regeln für sichere Linux-Server — 1,000 Zugriffsregeln zum Schließen einer Sicherheitslücke

1.000 RBAC-Regeln für sichere Linux-Server Kontakt aufnehmen →

Warum es wichtig ist

CSRF blieb auf beiden bekanntesten Schwachstellenlisten vertreten. CWE-352 stand 2009, 2010 und 2011 sowie in jeder Ausgabe von 2019 bis 2025 in den CWE Top 25. OWASP gab ihm in den Top 10 von 2007 und 2010 als A5 sowie 2013 als A8 eine eigene Kategorie. Seit 2021 gehört es zu A01, Broken Access Control, das die Ausgabe 2025 anführt. Laut OWASP hatte jede der getesteten Anwendungen irgendeine Form fehlerhafter Zugriffskontrolle. CWE-352 wird als eine der bemerkenswerten Schwächen genannt.

Der Angriff lässt sich schwer ausrotten, weil der Browser die Arbeit erledigt. Der W3C-Entwurf zu Fetch Metadata setzt dort an: Browser lassen sich leicht dazu bringen, Anfragen an die Endpunkte einer Website zu senden und die ohne ausdrückliche Übergabe verfügbaren Anmeldemittel des Nutzers einzubeziehen, etwa Cookies oder eine privilegierte Position im Intranet. Anwendungen müssen deshalb sehr sorgfältig mit dem Verhalten dieser Endpunkte umgehen, und das erweist sich als schwierig. CWE merkt an, dass nahezu jedes HTML-Element den Angriff auslösen kann, auch Bild-Tags und Links. Die beobachteten Beispiele lesen sich wie eine Liste von Verwaltungsaktionen, die von fremden Seiten aus ausgeführt wurden: über eine URL in einem `img` -Tag hinzugefügte Nutzerkonten (CVE-2004-1703), auf dieselbe Weise erlangte Administratorrechte (CVE-2004-1842) und eine Weboberfläche, bei der eine gefälschte Anfrage Passwörter ändern oder eine virtuelle Maschine stoppen konnte (CVE-2009-3759).

Scanner helfen wenig. Laut CWE lässt sich CSRF derzeit mit automatisierten Techniken nur schwer zuverlässig erkennen. Jede Anwendung hat ihre eigene implizite Regel dafür, welche Anfragen ein Außenstehender auslösen darf und welche hohe Sicherheit über die Absicht des Nutzers erfordern. Ein Werkzeug kann nicht wissen, dass ein Formular harmlos ist und das nächste Geld bewegt. Damit ist CSRF eine Entwurfsfrage: Entscheiden Sie Anfrage für Anfrage, was die Absicht des Nutzers belegt.

Browser haben sich zugunsten der Verteidiger weiterentwickelt. Laut OWASPs Merkblatt setzen aktuelle verbreitete Browser für Cookies standardmäßig SameSite=Lax durch. Dadurch bleiben Cookies bei den meisten websiteübergreifenden `POST`-Anfragen weg. Seit März 2023 senden zudem alle großen Browser die Fetch-Metadata-Header. Beides hilft, und keines löst das Problem abschließend. Der Cookieentwurf, der SameSite definiert, nennt Lax einen sinnvollen zusätzlichen Schutz gegen Angriffe mit unsicheren Methoden wie `POST`, aber keinen robusten Schutz gegen CSRF als Angriffskategorie. Selbst Strict sollte laut Entwurf nicht den gesamten Schutz einer Website ausmachen. Bei älteren oder eingebetteten Browsern, ergänzt OWASP, können sich Cookies so verhalten, als sei überhaupt kein SameSite-Wert gesetzt.

Wie es in der Praxis aussieht

Nehmen Sie eine Webanwendung unter `app.example.com`, bei der ein angemeldeter Nutzer seine Konto-E-Mail-Adresse durch ein Formular-POST an `/account/email` ändert. Das Sitzungscookie ist mit `SameSite=None; Secure` gesetzt, weil dasselbe Dashboard auch in das Portal eines Partners eingebettet wird. None ist der Wert, der das Senden eines Cookies im websiteübergreifenden Kontext erlaubt. Ein Angreifer veröffentlicht eine Seite unter `prizes.example.net` und bringt das Opfer dazu, sie zu öffnen:

```
<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>
```

Der Browser sendet das Formular ab, sobald die Seite lädt. Weil das Cookie es erlaubt, geht die Sitzung mit:

```
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
```

Für den Server ändert hier ein angemeldeter Nutzer seine Adresse. Wird nichts anderes geprüft, ändert sie sich. Ein über die neue Adresse abgewickeltes Zurücksetzen des Passworts übergibt das Konto dem Angreifer.

Zwei Header dieser Anfrage gehören dem Browser, nicht dem Angreifer. Der Fetch Standard zählt `Origin` zu den verbotenen Anfrageheadern, ebenso jeden mit `Sec-` beginnenden Headernamen. Ein Skript auf der Seite kann sie deshalb nicht setzen oder ändern. `Origin` nennt die Origin, von der die Anfrage kam. Der Fetch-Metadata-Entwurf definiert vier Werte für `Sec-Fetch-Site`: `cross-site`, `same-site`, `same-origin` sowie `none` für eine vom Nutzer durch Eingabe einer Adresse oder Auswahl eines Lesezeichens gestartete Navigation. Eine Prüfung kann sich auf beide stützen und bei Browsern, die keines von beiden senden, weiterhin auf ein Token zurückfallen. Eine Skizze in 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)
```

Das Token stammt aus `secrets.token_urlsafe(32)`. Laut Python-Dokumentation nutzt das Modul `secrets` die sicherste vom Betriebssystem bereitgestellte Zufallsquelle. 32 Byte, also 256 Bit, gelten demnach als ausreichend. Der Server fügt das Token als verborgenes Feld in jedes gerenderte Formular ein und behält eine Kopie in der Sitzung. Er schreibt es nie in ein Cookie, eine URL oder ein Log. Genau diese Orte schließt OWASPs Merkblatt für ein Synchronizer-Token aus. `hmac.compare_digest` ist so entworfen, dass es nicht beim ersten abweichenden Byte abbricht. Die Vergleichsdauer verrät deshalb nichts darüber, wie viel eines geratenen Werts richtig war. Beide Seiten werden zuerst in Bytes kodiert, weil die Funktion Zeichenfolgen nur akzeptiert, wenn sie ausschließlich ASCII enthalten. Eine Anfrage ohne `Origin`-Header gelangt zur Tokenprüfung. `Origin: null` scheitert, weil nur die exakte Origin besteht.

Nutzen Sie in einer echten Anwendung den Schutz des Frameworks, bevor Sie eigenen schreiben. Das ist der erste Punkt auf OWASPs Liste. Djangos Middleware verlangt etwa sowohl ihr CSRF-Cookie als auch ein Feld `csrfmiddlewaretoken` für jede Methode außer `GET`, `HEAD`, `OPTIONS` und `TRACE`. Fehlt eines oder ist es falsch, antwortet sie mit 403. Sie prüft `Origin` gegen den Host und die Einstellung `CSRF_TRUSTED_ORIGINS`. Muss die Anwendung nirgendwo eingebettet werden, kann das Sitzungscookie auch als `__Host-session=...; Path=/; Secure; SameSite=Lax` gesendet werden. OWASP hält fest, dass ein Cookie mit dem Präfix `__Host-` nicht von einer anderen Subdomain überschrieben werden kann.

Browsereinstellungen sind eine Schicht, nicht der gesamte Schutz. Der Bericht des Unternehmens über die Anwendungshärtung mit Nonce-basierter CSP hält fest, dass die Proxy-Middleware HSTS und SameSite-Cookies neben der Content Security Policy setzt. Die Berechtigungen werden bei jeder Anfrage erneut auf dem Server geprüft. Frontend-Sperren dienen allein dem Nutzungserlebnis.

Wo es schiefgeht

Zustandsänderungen hinter GET. SameSite=Lax sendet das Cookie weiterhin bei einer websiteübergreifenden Navigation auf oberster Ebene mit einer sicheren Methode. Ein vom Opfer angeklickter Link oder eine Seite, die das ganze Fenster zu `https://app.example.com/account/delete?confirm=yes` schickt, kommt deshalb angemeldet an. OWASP nennt das den häufigsten Grund, warum SameSite-

basierter Schutz in der Praxis scheitert. Django baut seinen Schutz auf dieselbe Annahme: `GET` und die anderen sicheren Methoden dürfen keine Nebenwirkungen haben. Halten Sie `GET`, `HEAD`, `OPTIONS` und `TRACE` frei von Nebenwirkungen und legen Sie jede Änderung hinter eine von Ihren Prüfungen erfasste Methode.

SameSite als gesamter Schutz. Der Cookieentwurf erklärt, warum Lax allein nicht genügt: Ein Angreifer kann weiterhin ein neues Fenster öffnen oder eine Navigation auf oberster Ebene auslösen, damit eine Anfrage als same-site zählt. Prerendering kann dasselbe unbemerkt vom Nutzer bewirken. OWASP ergänzt, dass SameSite die registrierbare Domain statt der vollständigen Origin betrachtet. Eine Anfrage von jedem Host unter example.com gilt für `app.example.com` als same-site. Eine vergessene, von jemandem übernommene Subdomain liegt deshalb bereits innerhalb der Schutzgrenze. Behandeln Sie SameSite als zweite Schicht und entscheiden Sie auf dem Server mit einem Token oder einer Origin-Prüfung.

Ein Token, das der Angreifer platzieren kann. Beim naiven Double-Submit-Verfahren prüft der Server nur, ob ein Cookie und ein Formularfeld denselben Zufallswert enthalten. OWASP rät inzwischen davon ab: Wer Cookies auf der Domain schreiben kann, über eine verwundbare benachbarte Subdomain, eine DNS-Übernahme oder ein über unverschlüsseltes HTTP eingeschleustes Cookie, kann beide Hälften setzen. Der CWE-Eintrag führt das einfache doppelt gesendete Cookie weiterhin als Gegenmaßnahme auf. Die Quellen widersprechen sich hier; folgen Sie OWASPs Warnung. Halten Sie das Token in der serverseitigen Sitzung oder verwenden Sie OWASPs signiertes Double-Submit-Cookie, das per HMAC an die Sitzung gebunden ist.

Cross-Site-Scripting auf derselben Website. Ein Skript in Ihren eigenen Seiten kann das Token aus dem Formular lesen und eine Anfrage senden, die nach jedem Maßstab von derselben Origin stammt. OWASPs Merkblatt sagt es deutlich: XSS „kann alle Techniken zur CSRF-Abwehr überwinden“. CWE und Django sagen dasselbe. CWEs bekanntestes Beispiel zeigt ihr Zusammenwirken. Der Samy-Wurm nutzte XSS, um sich in einem MySpace-Profil zu platzieren und den Angreifer als Freund hinzuzufügen. Bei Freunden, die das Profil ansahen, wurden die eigenen Profile genauso geändert. Weil kein Opfer diese Änderung wählte, zählt CWE CSRF zu den Grundursachen. Behandeln Sie jeden XSS-Befund auch als CSRF-Befund und beheben Sie ihn zuerst.

Eine Origin-Prüfung passt auf zu viel. Eine Prüfung wie `origin.startswith("https://app.example.com")` akzeptiert auch `https://app.example.com.attacker.example`, die von OWASPs Merkblatt beschriebene Falle. Eine weitere ist der Wert `null`. Nach dem Fetch Standard kommt ein von einer Seite mit der Referrer-Richtlinie `no-referrer` gesendetes Formular mit `Origin: null` an. Laut OWASP können auch Weiterleitungen über Origin-Grenzen und Sandbox-Frames diesen Wert erzeugen. Vergleichen Sie die vollständige Origin mit einer festen Liste exakter Zeichenfolgen. Behandeln Sie einen fehlenden Header oder `null` als unbekannt, nie als eigene Origin.

Zu großes Vertrauen in den Preflight. Ein benutzerdefinierter Anfrageheader ist ein solider Schutz für eine API. Browser senden vor jeder originübergreifenden Anfrage mit einem solchen Header einen Preflight. Das funktioniert nur, wenn die API alles andere ablehnt. OWASP warnt, dass eine JSON-API, die auch `text/plain` annimmt, von einem Formular erreicht werden kann: Das ist einer der drei für einfache Anfragen erlaubten Inhaltstypen. Eine CORS-Richtlinie, die Anmeldedaten aus jeder Subdomain per regulärem Ausdruck erlaubt, lässt zudem jeden nach einer Subdomain-Übernahme den benutzerdefinierten Header senden. Lehnen Sie vom Endpunkt nicht erwartete Inhaltstypen ab und führen Sie die exakten Origins auf, die Anfragen mit Anmeldedaten senden dürfen.

Das Anmeldeformular wird vergessen. Login-CSRF läuft umgekehrt: Der Angreifer meldet das Opfer im eigenen Konto des Angreifers an. In OWASPs Beispiel gibt das Opfer auf einer Einkaufswebsite danach Kartendaten ein, mit denen der Angreifer später einkaufen kann. Djangos Dokumentation nennt Login-CSRF als einen der von seinem Schutz erfassten Angriffe. Schützen Sie das Anmeldeformular mit einem an eine vorläufige Sitzung gebundenen Token. Zerstören Sie diese Sitzung nach der Anmeldung und beginnen Sie eine neue, wie OWASP empfiehlt.

Anfragen aus Angreifereingaben. Clientseitiges CSRF entsteht, wenn das eigene JavaScript der Anwendung eine Anfrage aus vom Angreifer kontrollierten Eingaben aufbaut, etwa aus der URL, dem Fensternamen oder einer `postMessage`-Nachricht. Das Skript ergänzt das Token selbst, und die Anfrage ist same-site. Weder Token noch SameSite stoppen das deshalb. OWASP beschreibt es als Rückkehr des Confused Deputy. Halten Sie Anfrageziele und Parameter außerhalb der Reichweite angreiferkontrollierter Eingaben oder wählen Sie sie aus einer festen Liste im Code.

Weiterführende Lektüre

- CWE-352: Cross-Site Request Forgery (CSRF) — der Schwachstelleneintrag: zusammengesetzte Struktur, Folgen, Gegenmaßnahmen und beobachtete CVEs.
- OWASP-Merkblattserie: Vermeidung von Cross-Site Request Forgery — Token, Fetch Metadata, Origin-Prüfungen, Grenzen von SameSite und Login-CSRF.
- W3C: Fetch Metadata Request Headers — der Header `Sec-Fetch-Site` und die vier Werte, die ein Browser senden kann.
- IETF-Entwurf: Cookies: HTTP State Management Mechanism (draft-ietf-httpbis-rfc6265bis) — das SameSite-Attribut und der Grund, warum Lax zusätzlicher Schutz statt einer vollständigen Lösung ist; weiterhin ein Internet-Draft.
- Django-Dokumentation: Schutz vor Cross Site Request Forgery — der eingebaute Framework-Schutz, seine Annahmen und die Umgehung durch Subdomains.
- OWASP Top 10:2025: A01 Broken Access Control — die Einordnung von CWE-352 in der Ausgabe 2025.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Bereitstellungsweg, die Betriebsrisiken und die Bereitschaftsverantwortlichen. Wir können eine Änderung mit Prüfungen, einem Wiederherstellungsweg und klaren Zuständigkeiten abgrenzen.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist CSRF?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/csrf/