Engineer

CSRF

et angreb, der narrer en browser til en uønsket forespørgsel

Hvad det er

CSRF, kort for cross-site request forgery, er det angreb, hvor en side, som offeret tilfældigvis åbner, får offerets egen browser til at sende en forespørgsel til et andet site, et sted, hvor offeret er logget ind. Browseren sender det sites cookies med, som den altid gør, så forespørgslen når frem med offerets session og ligner noget, offeret selv har valgt at gøre. Common Weakness Enumeration fører det som CWE-352 og beskriver fejlen fra serverens side: applikationen kontrollerer ikke, eller kan ikke kontrollere, godt nok, om en forespørgsel med vilje blev sendt af den bruger, den ser ud til at komme fra.

OWASP's cheat sheet om forebyggelse opregner, hvor angrebet kan begynde, fra et ondsindet website til en e-mail, et blogindlæg, en chatbesked eller et program, og hvad det kan opnå: en pengeoverførsel, en ændret adgangskode, et køb, en tildeling af rettigheder. Hver af dem er en handling på sitet, en ændring af tilstand, og det er dér, cheat sheetet sætter sine forsvar ind: et token på hver forespørgsel, der ændrer tilstand. CWE tilføjer, at rækkevidden kun begrænses af offerets rettigheder, fordi angriberen låner offerets identitet. Mod en administrator kan resultatet være fuld kontrol over webapplikationen.

CWE behandler CSRF som en sammensat svaghed, en svaghed, der kun opstår, hvor flere andre mødes. Blandt dem er en fejl i valideringen af oprindelse, CWE-346, og den utilsigtede proxy eller »confused deputy«, CWE-441: browseren har offerets legitimationsoplysninger og bruger dem på vegne af en side, der ingen har. Det adskiller CSRF fra cross-site scripting, som CWE opfører som en sidestillet svaghed. Ved XSS kører angriberens kode inde på målsitet; ved CSRF bliver den på angriberens side og sender kun forespørgsler. Det adskiller det også fra CORS, der styrer, hvad en side må læse fra en anden oprindelse. OWASP påpeger, at en almindelig HTML-formular sender en simpel forespørgsel, en, der ikke kræver nogen preflight, og at browsere tillader den slags forespørgsler til

enhver oprindelse. CWE staver selv betegnelsen Cross-Site Request Forgery med store forbogstaver, og posten nævner andre navne for det samme, blandt dem XSRF og Session Riding.

Brug begrebet i en beslutning

Brug »Hvad er CSRF?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Konti, adgangskoder og rettigheder sat op, så folk har præcis den adgang, jobbet kræver, og fratrådte mister den samme dag.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har skrevet 1.000 RBAC-regler til Linux-servere — 1,000 adgangsregler skrevet for at lukke et sikkerhedshul

Har skrevet 1.000 RBAC-regler til Linux-servere Kontakt os →

Hvorfor det betyder noget

CSRF har holdt sig på begge de mest kendte lister over svagheder. CWE-352 var med på CWE Top 25 i 2009, 2010 og 2011 og i hver udgave fra 2019 til 2025. OWASP gav det en kategori for sig selv på Top 10 i 2007 og 2010, som A5, og i 2013, som A8. Siden 2021 har det været en del af A01, Broken Access Control, der ligger øverst i 2025-udgaven. OWASP oplyser, at hver eneste af de testede applikationer havde en eller anden form for brudt adgangskontrol, og nævner CWE-352 blandt kategoriens nævneværdige svagheder.

Angrebet er svært at udrydde, fordi browseren gør arbejdet. W3C's udkast til Fetch Metadata tager udgangspunkt netop dér: browsere er nemme at få til at sende forespørgsler til et sites endpoints og til at sende de legitimationsoplysninger med, som brugeren i forvejen har, fx cookies eller en privilegeret position på et intranet, så applikationer må være meget varsomme med, hvordan de endpoints opfører sig, og det viser sig at være svært. CWE bemærker, at angrebet kan iscenesættes fra stort set et hvilket som helst tag eller HTML-konstruktion, billedtags og links medregnet. Postens observerede eksempler læses som en liste over administrative handlinger udført fra en andens side: brugerkonti oprettet via en URL i et `img` -tag (CVE-2004-1703), administratorrettigheder opnået på samme måde (CVE-2004-1842) og en webgrænseflade, hvor en forfalsket forespørgsel kunne ændre adgangskoder eller stoppe en virtuel maskine (CVE-2009-3759).

Scannere hjælper ikke meget. CWE skriver, at CSRF i dag er svært at finde pålideligt med automatiske teknikker, fordi hver applikation har sin egen underforståede politik for, hvilke forespørgsler en udenforstående må kunne

udløse, og hvilke der kræver vished for, at brugeren mente dem. Et værktøj kan ikke vide, at én formular er harmløs, og at den næste flytter penge. Det gør CSRF til et designspørgsmål: Beslut forespørgsel for forespørgsel, hvad der beviser, at brugeren mente det.

Browserne har rykket sig i forsvarerens favør. Ifølge OWASP's cheat sheet håndhæver de gængse browsere i dag SameSite=Lax på cookies som standard, hvilket holder dem ude af de fleste cross-site `POST` -forespørgsler, og alle større browsere har sendt Fetch Metadata-headerne siden marts 2023. Begge dele hjælper, og ingen af dem afgør sagen. Det cookieudkast, der definerer SameSite, kalder Lax et rimeligt dybdeforsvar mod angreb, der bygger på usikre metoder som `POST` , men ikke et robust forsvar mod CSRF som kategori, og råder til, at selv Strict ikke bør være hele et sites forsvar. Brugere med ældre eller indlejrede browsere, tilføjer OWASP, kan få cookies, der opfører sig, som om der slet ikke var sat nogen SameSite-værdi.

Sådan ser det ud i praksis

Tag en webapplikation på `app.example.com` , hvor en indlogget bruger ændrer e-mailadressen på sin konto med en formular, der sendes som `POST` til `/account/email` . Sessionscookien er sat med `SameSite=None; Secure` , fordi det samme dashboard også er indlejret i en partners portal, og None er den værdi, der lader en browser sende en cookie i en cross-site-sammenhæng. En angriber lægger en side op på `prizes.example.net` og får offeret til at åbne den:

```
<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>
```

Browseren sender formularen, så snart siden er indlæst, og da cookien tillader det, følger sessionen med:

```
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
```

For serveren er det en indlogget bruger, der ændrer sin adresse. Bliver intet andet kontrolleret, ændres adressen, og en nulstilling af adgangskoden, der sendes til den nye adresse, giver angriberen kontoen.

To af headerne i den forespørgsel hører til browseren, ikke til angriberen. Fetch-standarden opfører `Origin` blandt de forbudte request-headere sammen med alle headere, hvis navn begynder med `Sec-`, så et script på siden kan hverken sætte eller ændre dem. `Origin` angiver den oprindelse, forespørgslen kom fra, og udkastet til Fetch Metadata definerer fire værdier for `Sec-Fetch-Site`: `cross-site`, `same-site`, `same-origin` og `none` for en navigation, brugeren selv satte i gang ved at taste en adresse eller vælge et bogmærke. Et tjek kan læne sig op ad begge og stadig falde tilbage på et token for browsere, der ikke sender nogen af dem. En skitse i 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)
```

Tokenet kommer fra `secrets.token_urlsafe(32)`. Ifølge Pythons dokumentation bruger `secrets`-modulet den mest sikre kilde til tilfældighed, operativsystemet stiller til rådighed, og 32 bytes, altså 256 bit, anses for at være nok. Serveren lægger tokenet i et skjult felt i hver formular, den genererer, og gemmer en kopi i sessionen. Det havner aldrig i en cookie, en URL eller en log, de steder, OWASP's cheat sheet udelukker for et synchronizer-token. `hmac.compare_digest` er lavet til ikke at stoppe ved den første byte, der er forskellig, så den tid, en sammenligning tager, ikke røber, hvor meget af et gæt der var rigtigt. Begge sider kodes først til bytes, fordi funktionen kun tager imod tekststrenge, når de er ren ASCII. En forespørgsel uden `Origin`-header falder igennem til tokenet, og `Origin: null` dumper, fordi kun den præcise oprindelse slipper igennem.

I en rigtig applikation bør man gribe til frameworkets beskyttelse, før man skriver sin egen; det er det første punkt på OWASP's liste. Djangos middleware kræver fx både sin CSRF-cookie og et `csrfmiddlewaretoken`-felt ved alle andre metoder end `GET`, `HEAD`, `OPTIONS` og `TRACE`, svarer med 403, når et af dem mangler eller er forkert, og kontrollerer `Origin` mod værten og indstillingen `CSRF_TRUSTED_ORIGINS`. Hvor intet behøver at indlejre applikationen, kan sessionscookien også sendes som `__Host-session=...; Path=/; Secure; SameSite=Lax`. OWASP bemærker, at en cookie med præfikset `__Host-` ikke kan overskrives fra et andet subdomæne.

Browserens indstillinger er et lag, ikke selve forsvaret. Vores beskrivelse af, hvordan vi har hærdet applikationen med nonce-baseret CSP, nævner, at proxy-middlewaren sætter HSTS og SameSite-cookies sammen med Content Security Policy, og at rettighederne kontrolleres igen på serveren ved hver forespørgsel, mens kontrollerne i frontenden kun betragtes som UX.

Hvor det går galt

Tilstandsændringer bag GET. SameSite=Lax sender stadig cookien ved en cross-site navigation på øverste niveau med en sikker metode, så et link, offeret klikker på, eller en side, der sender hele vinduet til `https://app.example.com/account/delete?confirm=yes`, ankommer logget ind. OWASP kalder det den mest almindelige måde, SameSite-baserede forsvar svigter på i praksis, og Django bygger sin beskyttelse på samme præmis: `GET` og de andre sikre metoder må ikke have sideeffekter. Hold `GET`, `HEAD`, `OPTIONS` og `TRACE` fri for sideeffekter, og læg enhver ændring bag en metode, dine tjek dækker.

SameSite som hele forsvaret. Cookieudkastet forklarer, hvorfor Lax ikke er nok i sig selv: En angriber kan stadig åbne et nyt vindue eller udløse en navigation på øverste niveau for at få en forespørgsel til at tælle som same-site, og prerendering kan gøre det samme, uden at brugeren lægger mærke til det. OWASP tilføjer, at SameSite er afgrænset til det registrerbare domæne, ikke til oprindelsen. En forespørgsel fra en hvilken som helst vært under example.com tæller som same-site for `app.example.com`, så et glemt subdomæne, som nogen overtager, er allerede inden for hegnet. Behandl SameSite som et ekstra lag, og afgør sagen på serveren med et token eller et tjek af oprindelsen.

Et token, angriberen kan plante. I det naive double-submit-mønster kontrollerer serveren kun, at en cookie og et formularfelt indeholder den samme tilfældige værdi. OWASP fraråder nu det mønster, fordi enhver, der kan skrive cookies på domænet, gennem et sårbart subdomæne under samme domæne, en DNS-overtagelse eller en cookie sprøjtet ind over almindelig HTTP, kan sætte begge halvdele. CWE-posten opfører stadig den almindelige double-submitted cookie blandt sine afhjælpninger, så de to kilder er uenige her, og det er OWASP's advarsel, man skal følge. Gem tokenet i sessionen på serveren, eller brug OWASP's signerede double-submit-cookie, bundet til sessionen med en HMAC.

Cross-site scripting på samme site. Et script, der kører inde på dine egne sider, kan læse tokenet fra formularen og sende en forespørgsel, der er same-origin efter enhver målestok. OWASP's cheat sheet siger det ligeud: XSS kan sætte alle teknikker til at afbøde CSRF ud af spil. CWE og Django siger det samme, og CWE's mest kendte eksempel viser de to arbejde sammen. Samy-ormen brugte XSS til at plante sig selv i en MySpace-profil og tilføje angriberen som ven, og venner, der så profilen, fik deres egne profiler ændret på samme måde; da intet offer selv valgte den ændring, regner CWE CSRF som en grundårsag. Behandl ethvert XSS-fund som et CSRF-fund også, og ret det først.

Et tjek af oprindelsen, der matcher for meget. En test som `origin.startswith("https://app.example.com")` accepterer også `https://app.example.com.attacker.example`, den fælde, OWASP's cheat sheet advarer imod. Værdien `null` er en anden. Efter Fetch-standarden ankommer en formular, der sendes fra en side med referrer-politikken `no-referrer`, med `Origin: null`, og OWASP tilføjer, at omdirigeringer på tværs af oprindelser og sandboxede frames også kan give den. Sammenlign hele oprindelsen med en fast liste af præcise strenge, og behandl en manglende header eller `null` som ukendt, aldrig som din egen oprindelse.

For meget tillid til preflight. En brugerdefineret request-header er et solidt forsvar for et API, fordi browsere sender en preflight før enhver cross-origin-forespørgsel, der har en. Det holder kun, hvis API'et afviser alt andet. OWASP advarer om, at et JSON-API, der også tager imod `text/plain`, kan nås fra en formular, fordi det er en af de tre indholdstyper, en simpel forespørgsel må bruge, og at en CORS-politik, der tillader legitimationsoplysninger fra alle subdomæner via et regulært udtryk, lader enhver, der overtager et subdomæne, sende den brugerdefinerede header. Afvis indholdstyper, endpointet ikke forventer, og opregn de præcise oprindelser, der må sende forespørgsler med legitimationsoplysninger.

Login-formularen bliver glemt. Login-CSRF går den modsatte vej: Angriberen logger offeret ind på angriberens egen konto. På et shoppingsite, lyder OWASP's eksempel, indtaster offeret derefter sine kortoplysninger, og angriberen kan senere købe ting med dem. Djangos dokumentation nævner login-CSRF blandt de angreb, beskyttelsen dækker. Beskyt login-formularen med et token knyttet til en foreløbig session, og nedlæg den session og start en ny, når brugeren er logget ind, som OWASP råder til.

Forespørgsler bygget af angriberens input. Client-side CSRF opstår, når applikationens egen JavaScript bygger en forespørgsel ud fra input, angriberen styrer, fx URL'en, vinduesnavnet eller en `postMessage`. Scriptet tilføjer selv tokenet, og forespørgslen er same-site, så hverken tokens eller SameSite stopper den; OWASP beskriver det som confused deputy-problemet, der vender tilbage. Hold mål og parametre for forespørgsler uden for rækkevidde af input, angriberen styrer, eller vælg dem fra en fast liste i koden.

Læs mere

- CWE-352: Cross-Site Request Forgery (CSRF) — svaghedsposten: den sammensatte struktur, konsekvenserne, afhjælpningerne og de observerede CVE'er.
- OWASP Cheat Sheet Series: Cross-Site Request Forgery Prevention — tokens, Fetch Metadata, tjek af oprindelse, grænserne for SameSite og login-CSRF.
- W3C: Fetch Metadata Request Headers — headeren `Sec-Fetch-Site` og de fire værdier, en browser kan sende.
- IETF draft: Cookies: HTTP State Management Mechanism (draft-ietf-httpbis-rfc6265bis) — SameSite-attributten, og hvorfor Lax er dybdeforsvar og ikke en kur; stadig et Internet-Draft.

- Django documentation: Cross Site Request Forgery protection — et frameworks indbyggede beskyttelse, hvad den forudsætter, og hvor subdomæner kommer uden om den.
- OWASP Top 10:2025: A01 Broken Access Control — hvor CWE-352 står i 2025-udgaven.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring udrulningsforløbet, driftsrisiciene og dem, der har vagten. Vi kan afgrænse en ændring med kontroller, en vej tilbage og tydeligt ansvar.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er CSRF?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

welcome@engineer.company +45 50 10 61 84 København, Danmark

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/csrf/