Engineer

CSRF

ブラウザーをだまし、利用者が意図しないリクエストを送らせる攻撃

どのようなものか

CSRFは、cross-site request forgeryの略です。被害者がたまたま開いたページによって、被害者自身のブラウザーから、ログイン済みの別のサイトへリクエストを送らせる攻撃です。ブラウザーは、いつもどおり、そのサイトのCookieを添付します。そのため、被害者のセッションを伴って届き、自ら選んだ操作のように見えます。Common Weakness Enumerationは、これをCWE-352として登録し、サーバー側の弱点を説明しています。アプリケーションが、見掛け上の送信者である利用者自身の意図したリクエストかどうかを、十分に確認しない、または確認できないことです。

OWASPの防止チートシートは、攻撃の出発点として、悪意のあるウェブサイト、メール、ブログ投稿、インスタントメッセージ、プログラムを挙げています。実行できることには、送金、パスワードの変更、購入、権限の付与があります。いずれも、サイト上の操作、つまり状態の変更です。チートシートも、状態を変更するすべてのリクエストへトークンを付けるという形で、そこに防御を置きます。CWEによると、攻撃者が被害者の身元を借りるため、攻撃の到達範囲は、被害者の権限で決まります。管理者が対象なら、ウェブアプリケーション全体を制御される場合があります。

CWEは、CSRFを、複数の弱点が重なって初めて存在する、複合的な弱点として扱います。その中には、送信元の検証の誤りであるCWE-346と、意図しない代理、または「混乱した代理人」であるCWE-441があります。ブラウザーが被害者の認証情報を持ち、それを持たないページのために使います。ここが、CWEが同列の弱点に挙げるクロスサイトスクリプティングとの違いです。XSSでは、攻撃者のコードが対象サイト内で動きます。CSRFでは、コードは攻撃者のページにとどまり、リクエストだけを送ります。別のオリジンから何を読めるかを制御するCORSとも異なります。OWASPによると、通常のHTMLフォームは、プリフライトの不要な単純なリクエストを送り、ブラウザーは、どのオリジンへの送信も認めています。CWEの項目名はCross-Site Request Forgeryと表記し、同じものの別名として、XSRFやSession Ridingなどを挙げています。

概念を判断に生かす

「CSRFとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 アカウント、パスワード、権限を整理し、業務に必要なアクセスだけを付与し、退職者の権限は当日に解除します。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 Linuxサーバーを守る1,000件のRBACルールを作成 — 1,000 セキュリティ上の穴を塞ぐアクセスルール

Linuxサーバーを守る1,000件のRBACルールを作成 お問い合わせ →

なぜ重要なのか

CSRFは、最もよく知られた二つの弱点一覧に残り続けています。CWE-352は、2009年、2010年、2011年と、2019年から2025年までのすべての版のCWE Top 25に入りました。OWASPは、2007年と2010年のTop 10でA5、2013年ではA8として、独立した分類を設けました。2021年以降は、アクセス制御の不備を扱うA01、Broken Access Controlに含まれ、2025年版では、その分類が先頭です。OWASPは、検査したすべてのアプリケーションに、何らかのアクセス制御の不備があったと報告し、主な弱点の一つにCWE-352を挙げています。

根絶しにくいのは、ブラウザーが仕事をしてしまうためです。W3CのFetch Metadata草案も、そこから説明を始めています。ブラウザーには、サイトのエンドポイントへリクエストを送り、Cookieや、イントラネット内からアクセスできる特別な立場など、利用者がその場で持つ認証・アクセス上の権限を使わせることが、容易にできます。そのため、アプリケーションは、エンドポイントの振る舞いに細心の注意を払う必要があり、それが難しいのです。CWEは、画像タグやリンクを含む、ほぼどのHTML要素からも攻撃を開始できるとしています。観測された例は、他人のページから行った管理操作の一覧のようです。`img` タグ内のURLで利用者アカウントを追加した例（CVE-2004-1703）、同じ方法で管理者権限を得た例（CVE-2004-1842）、偽造したリクエストでパスワードを変えたり、仮想マシンを停止したりできたウェブ画面（CVE-2009-3759）があります。

スキャナーも、あまり役立ちません。CWEによると、現在、自動手法でCSRFを確実に検出するのは困難です。どのリクエストなら外部の人が起こしてよく、どれには利用者が意図したという強い確証が必要かについて、アプリケーションごとに暗黙の方針があるためです。一方のフォームは無害で、次のフォームは送金するという違いは、ツールには分かりません。したがって、CSRFは設計上の問題です。その操作が利用者の意図したものだと、何によって確かめるかを、リクエストごとに決めます。

ブラウザーは、防御する側に有利な方向へ変わりました。OWASPのチートシートによると、現在の主要ブラウザーは、既定でCookieにSameSite=Laxを適用し、サイトをまたぐ `POST` の大半にCookieを付けません。また、すべての主要ブラウザーが、2023年3月以降、Fetch Metadataヘッダーを送っています。どちらも役立ちますが、それだけで問題は解決しません。SameSiteを定めるCookieの草案は、Laxを、 `POST` のような安全でないメソッドに頼る攻撃への、妥当な多層防御としています。ただし、CSRFという攻撃全般に対する堅牢な防御ではなく、Strictモードでさえ、サイトの防御のすべてにすべきではないと勧めています。OWASPは、古いブラウザーや組み込みブラウザーの利用者には、SameSiteを一切指定しなかったかのように動くCookieが届く可能性も指摘しています。

実際の活用例

`app.example.com` のウェブアプリケーションで、ログイン済みの利用者が、 `/account/email` へのフォームのPOST送信で、アカウントのメールアドレスを変えるとします。セッションCookieは、 `SameSite=None; Secure` に設定しています。同じダッシュボードを提携先のポータルにも埋め込み、サイトをまたぐ状況でもブラウザーにCookieを送らせる値が、Noneであるためです。攻撃者は、 `prizes.example.net` にページを公開し、被害者に開かせます。

```
<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>
```

ブラウザーは、ページを読み込むとすぐにフォームを送信します。Cookieの設定が認めているため、セッションも付きます。

```
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
```

サーバーには、ログイン済みの利用者がアドレスを変更しているように見えます。ほかに検査がなければ、アドレスが変わり、新しい先へ送るパスワード再設定によって、アカウントが攻撃者の手に渡ります。

このリクエストの二つのヘッダーは、攻撃者ではなくブラウザーが管理します。Fetch Standardは、`Origin` と、名前が `Sec-` で始まるすべてのヘッダーを、設定を禁じるリクエストヘッダーに挙げています。そのため、ページ上のスクリプトは、設定も変更もできません。`Origin` は、リクエストを送ったオリジンを示します。Fetch Metadata草案は、`Sec-Fetch-Site` の値を四つ定めています。`cross-site` 、`same-site` 、`same-origin` と、アドレスの入力やブックマークの選択によって利用者自身が始めた画面遷移を示す `none` です。検査では両方を使い、どちらも送らないブラウザーでは、トークンで確認できます。Pythonでの概略は次のとおりです。

```
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)
```

トークンには、`secrets.token_urlsafe(32)` を使います。Pythonのドキュメントによると、`secrets` モジュールは、OSが提供する最も安全な乱数源を使い、32バイト、256ビットで十分だと考えられています。サーバーは、生成するすべてのフォームの隠しフィールドにトークンを入れ、セッションにもコピーを保持します。Cookie、URL、ログには決して入れません。OWASPのチートシートが、同期トークンの保存先として認めない場所だからです。`hmac.compare_digest` は、最初に異なるバイトで比較を止めないように設計されています。そのため、比較時間から、推測がどこまで当たっていたかは分かりません。テキスト文字列を受け入れるのは純粋なASCIIの場合だけなので、両側を先にバイト列へ変換します。`Origin` ヘッダーのないリクエストは、トークンの検査へ進みます。`Origin: null` は失敗します。完全に一致するオリジンだけを認めるためです。

実際のアプリケーションでは、自作する前にフレームワークの保護を使います。OWASPの一覧でも、最初の項目です。たとえば、Djangoのミドルウェアは、`GET`、`HEAD`、`OPTIONS`、`TRACE`以外のすべてのメソッドで、CSRF Cookieと`csrfmiddlewaretoken`フィールドの両方を求めます。どちらかが欠けるか誤っていれば403を返し、`Origin`をホストと`CSRF_TRUSTED_ORIGINS`設定に照らして検査します。アプリケーションを埋め込む必要がなければ、セッションCookieを`__Host-session=...; Path=/; Secure; SameSite=Lax`として送ることもできます。OWASPによると、`__Host-`接頭辞の付いたCookieは、別のサブドメインから上書きできません。

ブラウザーの設定は、一つの防御層であり、防御全体ではありません。当社のnonceベースのCSPでアプリケーションを強化した記録には、プロキシミドルウェアが、Content Security PolicyとともにHSTSとSameSite Cookieを設定するとあります。また、権限は、リクエストごとにサーバーで再確認し、フロントエンドの制御は利用体験のためだけに扱うと説明しています。

問題が起きるところ

GETで状態を変更します。 SameSite=Laxでも、安全なメソッドで、サイトをまたいでトップレベルの画面遷移を行う場合は、Cookieを送ります。そのため、被害者がクリックするリンクや、ウィンドウ全体を`https://app.example.com/account/delete?confirm=yes`へ移動させるページからのリクエストも、ログイン済みとして届きます。OWASPは、これを、SameSiteを使う防御が実際に失敗する、最も一般的な原因としています。Djangoの保護も、同じ前提に立ちます。`GET`などの安全なメソッドに、副作用があってはいけません。`GET`、`HEAD`、`OPTIONS`、`TRACE`は、副作用のないものとし、すべての変更を、検査対象のメソッドで行います。

SameSiteを、防御のすべてにします。 Cookieの草案は、Laxだけでは不十分な理由を説明しています。攻撃者は、新しいウィンドウを開いたり、トップレベルの画面遷移を起こしたりして、リクエストを同一サイトのものとして扱わせられます。事前描画でも、利用者に気付かれずに同じことができます。OWASPは、SameSiteが、完全なオリジンではなく、登録可能なドメインを見ることも指摘しています。example.com配下のどのホストからのリクエストも、`app.example.com`にとっては同一サイトです。そのため、忘れられたサブドメインを誰かが乗っ取れば、既に防御の内側にいます。SameSiteは第二の防御層とし、サーバーでトークンか送信元の検査によって判断します。

攻撃者が仕込めるトークンを使います。 単純な二重送信の方法では、Cookieとフォームのフィールドが、同じ乱数値を持つかだけをサーバーで検査します。OWASPは、現在、この方法を推奨しないものとしています。脆弱な別のサブドメイン、DNSの乗っ取り、平文のHTTP経由でのCookieの注入などによって、そのドメインへCookieを書ける人なら、両方を設定できるためです。CWEの項目は、今も、単純なCookieの二重送信を緩和策に挙げています。この点では、二つの出典が異なり、従うべきなのはOWASPの注意です。トークンをサーバー側のセッションに保持するか、OWASPが示す、HMACでセッションへ結び付けた、署名付きCookieの二重送信を使います。

同じサイトに、クロスサイトスクリプティングがあります。 自分のページ内で動くスクリプトは、フォームからトークンを読み、どの尺度でも同一オリジンとなるリクエストを送れます。OWASPのチートシートは、XSSが「すべてのCSRF緩和技術を破れる」と明言しています。CWEとDjangoも同じことを述べ、CWEの最も有名な例は、両者が連動するものです。Samyワームは、XSSでMySpaceのプロフィールへ自分自身を仕込み、攻撃者を友達へ追加しました。そのプロフィールを見た友達のプロフィールも、同じように変更しました。どの被害者も、その変更を選んでいないため、CWEは、CSRFを根本原因の一つに数えています。XSSの指摘は、CSRFの指摘としても扱い、まず直します。

送信元の検査が、広すぎる範囲に一致します。
`origin.startswith("https://app.example.com")` のような検査は、`https://app.example.com.attacker.example` も受け入れます。OWASPのチートシートが注意する落とし穴です。 `null` も、もう一つの落とし穴です。Fetch Standardによると、参照元ポリシーが `no-referrer` であるページから送ったフォームは、`Origin: null` で届きます。OWASPは、オリジンをまたぐリダイレクトや、サンドボックス化したフレームでも、この値になるとしています。オリジン全体を、完全一致する文字列の固定リストと比較します。ヘッダーがない場合や `null` は、不明として扱い、自分のオリジンとは決して見なしません。

プリフライトを、信用しすぎます。 独自のリクエストヘッダーは、APIにとって有効な防御です。ブラウザーは、それを持つクロスオリジンリクエストの前に、必ずプリフライトを送るためです。ただし、APIがそれ以外をすべて拒否する場合に限ります。OWASPは、JSON APIが `text/plain` も受け入れると、フォームから到達できると注意しています。単純なリクエストで使える三種類のコンテンツタイプの一つだからです。また、正規表現で、すべてのサブドメインから認証情報付きリクエストを許可するCORSポリシーでは、サブドメインを乗っ取った人も、独自のヘッダーを送れます。エンドポイントが想定しないコンテンツタイプは拒否し、認証情報付きリクエストを送ってよいオリジンを、正確に列挙します。

ログインフォームを忘れます。 ログインCSRFは、逆向きに動きます。攻撃者は、被害者を、攻撃者自身のアカウントへログインさせます。OWASPの例では、被害者が買い物サイトでカード情報を入力し、攻撃者が後でそれを使って購入できます。Djangoのドキュメントも、保護対象にログインCSRFを挙げています。ログイン前のセッションに結び付けたトークンでログインフォームを保護し、ログインしたら、そのセッションを破棄して新しいものを開始します。OWASPが勧める方法です。

攻撃者の入力から、リクエストを組み立てます。 クライアント側CSRFは、アプリケーション自身のJavaScriptが、URL、ウィンドウ名、`postMessage`など、攻撃者が制御する入力からリクエストを作る場合に起きます。スクリプト自身がトークンを加え、リクエストも同一サイトになるため、トークンもSameSiteも防げません。OWASPは、混乱した代理人の再来と説明しています。リクエストの宛先とパラメーターを、攻撃者が制御する入力から変更できないようにするか、コード内の固定リストから選びます。

参考資料

- CWE-352：クロスサイトリクエストフォージェリ（CSRF） — 弱点の登録項目です。複合的な構造、結果、緩和策、観測されたCVEを説明します。
- OWASPチートシートシリーズ：クロスサイトリクエストフォージェリの防止 — トークン、Fetch Metadata、送信元の検査、SameSiteの限界、ログインCSRFを説明します。
- W3C：Fetch Metadataリクエストヘッダー — `Sec-Fetch-Site`ヘッダーと、ブラウザーが送れる四つの値を説明します。
- IETF草案：Cookie：HTTPの状態管理機構（draft-ietf-httpbis-rfc6265bis） — SameSite属性と、Laxが根本的な解決ではなく、多層防御である理由を説明します。現在もInternet-Draftです。
- Djangoドキュメント：クロスサイトリクエストフォージェリへの保護 — フレームワーク組み込みの保護、その前提、サブドメインから迂回できる箇所を説明します。
- OWASP Top 10:2025：A01 アクセス制御の不備 — 2025年版でのCWE-352の位置付けを説明します。

用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「CSRFとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

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

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/csrf/