**１．同一オリジンポリシーとは**

**（１）定義**

同一オリジンポリシーとは、Webブラウザが異なるオリジン（Origin）間でのアクセスを制限する仕組みです。

これは 「ブラウザ側で強制されるセキュリティモデル」 の一つで、悪意あるサイトが別サイトの機密情報を盗むことを防ぐ目的で導入されました。

**２．オリジン（Origin）の構成要素**

オリジンは以下の３要素で決まります。

|  |  |
| --- | --- |
| **要素** | **例** |
| スキーム（protocol） | http / https / file |
| ホスト名（host） | [www.example.com](https://www.example.com) |
| ポート番号（port） | 80, 443, 8080など |

**同一オリジンの条件：**

* スキーム・ホスト名・ポート番号の全てが一致すること

**例：**

https://example.com:443 と https://example.com は同一オリジン（省略時のポートはデフォルト443） https://example.com と http://example.com は異なるオリジン（スキームが違う）

**３．同一オリジンポリシーの対象範囲**

SOPはすべてのブラウザリソースに一律適用されるわけではありません。リソース種類ごとに制限内容が異なります。

**（１）JavaScriptによるDOMアクセス**

制限が最も厳しい分野です。 異なるオリジンの iframe 内のDOMには直接アクセスできません。

**例：**

example.com 上のページが、evil.com の iframe 内テキストを .contentWindow.document で読む → 禁止

**（２）Cookie・LocalStorage・IndexedDB**

同一オリジンでのみ共有可能です。 例外的にサブドメイン間共有は Domain 属性設定で可能です（例：sessionidをexample.com全体で共有）。

**（３）XMLHttpRequest / fetch（AJAX通信）**

デフォルトでは同一オリジンのみ通信可能です。 異なるオリジンと通信する場合はCORS設定が必要です。

**（４）画像・スクリプト・CSSの読み込み**

読み込み自体は異なるオリジンから可能です（CDN利用など）。 ただし、JavaScriptでピクセルデータを取得（Canvas API）する場合はクロスオリジン制約があります。

**４．例外と緩和策**

同一オリジンポリシーは厳格ですが、Webの利便性のために例外・緩和策が存在します。

|  |  |
| --- | --- |
| **緩和策** | **目的・特徴** |
| CORS（Cross-Origin Resource Sharing） | サーバーが Access-Control-Allow-Origin ヘッダで許可するオリジンを明示 |
| postMessage API | 異なるオリジン間で安全にメッセージをやり取り（iframe間通信） |
| JSONP（古い） | <script> タグでスクリプトとして取得し、コールバック関数でデータ受け取り |
| サーバーサイドプロキシ | 同一オリジン内サーバーが外部リソースを取得して返却 |

**５．同一オリジンポリシーが必要な理由**

もし同一オリジンポリシーが無ければ、以下のような攻撃が可能になります。

* **CSRF（クロスサイトリクエストフォージェリ）強化版**：悪意あるサイトから利用者がログイン中の別サイトへ勝手にデータ送信
* **機密情報の窃取**：他オリジンのDOMから直接ユーザー情報やトークンを読み取る
* **セッションハイジャック**：Cookieを盗み取り別人になりすます

**６．用語の由来**

**Origin**：英語で「起源・発端」。Webでは「通信の出発点＝スキーム＋ホスト＋ポート」を意味します。 **Same-Origin Policy**：直訳すると「同一の起源ポリシー」。

要するに「同じ出発点からのリソースは自由にやりとりしてよいが、違う出発点とは制限する」というルールです。

**７．関連技術との位置付け**

Webブラウザのセキュリティモデル

├─ 同一オリジンポリシー（SOP）

│ ├─ DOMアクセス制限

│ ├─ Cookie/Storage制限

│ ├─ AJAX通信制限

│ └─ Canvas画像データ制限

├─ CORS（SOPの例外許可）

├─ サンドボックス（iframeの隔離）

└─ コンテンツセキュリティポリシー（CSP）

はい、承知いたしました。ご提示いただいた優れた枠組みはそのままに、各項目の解説を、ウェブ開発初心者の方でもイメージしやすいように、比喩や具体例を交えて加筆・修正します。

---

**ウェブセキュリティの基本：同一オリジンポリシー（SOP）をマスターしよう！**

インターネットの世界では、様々なウェブサイトが共存しています。もし、悪意のあるサイトが、あなたがログインしている銀行サイトやSNSの情報を勝手に盗み見たり、操作したりできたら大変ですよね。そうした混乱を防ぐために、ブラウザには\*\*「同一オリジンポリシー（SOP）」\*\*という、とても重要な基本ルールが備わっています。

これは、\*\*「ウェブサイトは、自分と同じ出身地（オリジン）の仲間としか自由にやりとりできない」\*\*という、ブラウザの門番のようなルールです。この門番（SOP）が何を許可し、何を禁止するのかを理解することが、安全なウェブサイト作りの第一歩です。

**１ 前提モデル（SOPの効く範囲）**

SOPを理解するための、3つの大きな原則から見ていきましょう。

**１－１ SOPの３原則（記憶フック）**

SOPの働きは、次の3つのイメージで覚えると簡単です。

**(１) 読めないが送れる：手紙は送れるが、返事は読めない**

* **解説：** あなたのサイト（Aサイト）から、全く別のサイト（Bサイト）へ、リクエストという「手紙」を送ること自体は可能です。しかし、Bサイトからの「返信の中身」をJavaScriptで読み取ることは、原則としてブロックされます。
* **例：** 悪意のあるサイトが、あなたの銀行サイトに「送金しろ」というリクエストを送ることはできてしまいます。しかし、あなたの「口座残高」が書かれた返信を盗み見ることはできません。

**(２) 埋め込めるが覗けない：美術館の絵のように、飾れるが分析はできない**

* **解説：** <img>（画像）や<iframe>（窓）タグを使えば、他のサイトのコンテンツを自分のページに「飾る」ように埋め込むことができます。しかし、その中身を詳しく調べたり（例：<iframe>内の文字を読み取る）、改造したりすることはできません。
* **例：** YouTubeの動画を自分のブログに埋め込めますが、その動画データ自体を抜き出したり、動画内のボタンをプログラムで勝手に押したりすることはできません。<iframe>は「ガラス窓」のようなもので、向こうの景色は見えますが、手を伸ばして家具を動かすことはできないのです。

**(３) 同一オリジンだけ共有：家族だけの共有金庫**

* **解説：** CookieやWeb Storageといったブラウザ内の「金庫」は、とても大切な情報を保管する場所です。この金庫は、同じオリジン（家族）のページからしか開けることができません。
* **例：** example.comで保存したログイン情報は、同じexample.com内のページでしか使えません。全く関係ないevil.comからは、その金庫の中身を覗くことすらできません。

**１－２ 用語の由来**

**(１) Origin（オリジン）＝ インターネット上の「住所」**

* **解説：** オリジンとは、ウェブサイトの「出身地」を特定するための住所のようなものです。次の3つの要素が**すべて**一致して、初めて「同一オリジン」とみなされます。
  1. **scheme（スキーム）：** http:// や https:// の部分。「～市」に相当。
  2. **host（ホスト）：** www.example.com の部分。「～町〜丁目」に相当。
  3. **port（ポート）：** :80 や :443 の部分。「～番地」に相当。（通常は省略されます）
* http://example.com と https://example.com は、スキームが違うので「別オリジン（他人）」です。

**(２) SOP（Same-Origin Policy）＝ 同一起源ポリシー**

* **解説：** その名の通り、「同じオリジン（Same-Origin）の仲間とのやりとりにだけ、特別な許可を与える（Policy）」というルールです。

**２ SOP単独で保護されるリスク（=SOPの門番が防いでくれること）**

SOPがあるおかげで、自動的に防がれている基本的な攻撃です。

**２－１ クロスオリジンDOM読み取り**

* **(１) 内容：** 悪意のあるサイトが、あなたの開いている別のサイト（例：ネット銀行）を透明な<iframe>でページに埋め込み、あなたが入力したIDやパスワードをJavaScriptで盗み取ろうとする攻撃。
* **(４) SOPの遮断点：** SOPの「埋め込めるが覗けない」ルールにより、<iframe>の中身（DOM）にアクセスしようとすると、ブラウザが「それは別オリジンだからダメ！」とエラーを出してブロックします。

**２－２ クロスオリジンCookie/Storage窃取**

* **(１) 内容：** 悪意のあるサイトが、別のサイトであなたのブラウザに保存されている情報（ログイン情報など）をJavaScriptで直接盗み見ようとすること。
* **(２) 遮断：** SOPの「同一オリジンだけ共有」ルールにより、他人の家の金庫（Cookie/Storage）は開けられないため、失敗します。

**２－３ クロスオリジンAJAXレスポンス窃取**

* **(１) 内容：** JavaScriptのfetch()などを使って、裏側でこっそり外部のAPI（例：会員情報API）にリクエストを送り、返ってきた個人情報（レスポンス）を盗み見ようとすること。
* **(２) 遮断：** SOPの「読めないが送れる」ルールが発動。リクエストは届くかもしれませんが、相手のサーバーが「あなたからのアクセスを許可しますよ（CORS設定）」という特別な許可を出さない限り、ブラウザはレスポンスの中身をJavaScriptに渡してくれません。

**２－４ Canvasピクセル窃取**

* **(１) 内容：** 他のサイトにある画像を、自分のお絵かきキャンバス（<canvas>）に読み込み、その画像の色情報を1ピクセル単位で分析して情報を盗み出す攻撃。（例：画像に含まれるQRコードや文字を解析する）
* **(２) 遮断：** 他のオリジンから持ってきた画像でcanvasを操作しようとすると、canvasが「汚染された（tainted）」状態になります。これは「出所不明の絵の具が混ざった」ような状態で、安全のため色情報の抜き取り（getImageData()）ができなくなります。

**３ SOP＋追加対策が必要なリスク（=SOPだけでは防ぎきれないこと）**

SOPは強力ですが、万能ではありません。SOPのルールの穴を突くような、より巧妙な攻撃には追加の対策が必要です。

**３－１ CSRF（Cross-Site Request Forgery：リクエスト強要）**

* **(１) 内容：** 「リクエスト偽造」とも呼ばれます。あなたがログインしている状態で、罠サイトにアクセスさせ、あなたの意図しない操作（例：勝手に商品を買う、退会する）をさせるリクエストを、裏で勝手に送らせる攻撃です。
* **(４) SOPの限界：** SOPは「返事を読むな」とは言いますが、「手紙を送るな」とは言いません。この\*\*「リクエストは送れてしまう」\*\*という点を悪用します。リクエストが送られる際、ブラウザは自動的にそのサイト用のCookie（ログイン情報など）を添付してしまうため、サーバーは「本人からの正規のリクエストだ」と勘違いしてしまいます。
* **(５) 追加対策：**
  + **CSRFトークン：** サーバーが発行する「本人しか知らない合言葉」。この合言葉がリクエストに含まれていないと、サーバーは処理を拒否します。
* **(６) 覚え方：SOPは“読む”のには強いが、“送る”のは止められない。だから「合言葉（トークン）」で本人確認する。**

**３－２ クリックジャッキング**

* **(１) 内容：** 罠サイトの上に、あなたが利用している正規サイトを\*\*透明な<iframe>\*\*で重ねて表示します。「景品が当たりました！」というボタンをクリックさせると、実はその真下には正規サイトの「退会する」ボタンが隠れていて、意図せず退会させられてしまう、といった攻撃です。
* **(２) SOPの限界：** SOPは<iframe>の中身を「覗く」のは防ぎますが、「埋め込む」こと自体は禁止しません。この\*\*「埋め込めてしまう」\*\*点を悪用し、ユーザーのクリックを乗っ取ります。
* **(３) 追加対策：**
  + **X-Frame-Optionsヘッダ：** サーバー側から「うちのサイトは<iframe>の中に埋め込まないでください！」という指示をブラウザに送る仕組み。
* **(４) 覚え方：“覗けないが、押させられてしまう”。だから「そもそも埋め込ませない」ようにする。**

**３－３ クロスオリジンAPI連携の安全化（CORS）**

* **(１) 背景：** ときには、SOPの壁を越えて、正当な目的で他のオリジンとデータをやり取りしたい場合があります。（例：天気予報サイトが、気象庁のAPIからデータを取得する）
* **(３) 安全化：** そのための公式な仕組みが\*\*CORS（Cross-Origin Resource Sharing）\*\*です。APIサーバー側が「このオリジンからなら、データを読んでもいいですよ」という許可（Access-Control-Allow-Originヘッダ）を出すことで、SOPの制限を安全に解除できます。
* **(４) 誤設定リスク：** CORSは「SOPの壁に開ける公式な窓口」です。この窓口を「誰にでも（\*）」「何でも（Allow-Credentials: true）」許可するように設定すると、悪意のあるサイトも自由にデータを読めるようになり、非常に危険です。窓口は必要最小限の相手にだけ開けるべきです。
* **(５) 覚え方：CORSは“壁の公式な窓口”。口を開けすぎないことが重要。**

**３－４ XSS（Cross-Site Scripting）**

* **(１) 内容：** 攻撃者がウェブサイトの脆弱性を利用して、サイト内に悪意のあるJavaScriptコード（スクリプト）を埋め込む攻撃。
* **(３) SOPの限界：** SOPは「外部の敵（別オリジン）」から身を守るためのルールです。しかし、XSSで埋め込まれたスクリプトは\*\*「内部の人間（同一オリジン）」として\*\*実行されてしまいます。城壁は外敵には強いですが、城内に紛れ込んだスパイには無力なのと同じです。内部のスパイ（悪性スクリプト）は、そのサイトの権限でやりたい放題（Cookie窃取、DOM改ざんなど）できてしまいます。
* **(４) 追加対策：**
  + **エスケープ処理：** ユーザーが入力した<script>などの危険な文字を、無害な文字列（例：&lt;script&gt;）に変換してから表示する。
  + **CSP（Content Security Policy）：** 「このサイトでは、許可された場所からしかスクリプトを読み込みません」という厳格なルールを設定する。
* **(５) 覚え方：“同一オリジンの裏切り者”はSOPの想定外。だから「危険な文字を無害化（エスケープ）」したり、「実行できるプログラムを制限（CSP）」したりする。**

**４ SOPでは保護されない代表リスク（=SOPの守備範囲ではない問題）**

以下のリスクは、ブラウザのルールであるSOPとは別のレイヤーで起こる問題です。

* **４－１ 通信盗聴・改ざん（MITM）：** あなたとサーバーの間の通信経路（ネットワーク）で情報を盗まれたり、書き換えられたりする問題。
  + **対策：** 通信を暗号化する\*\*TLS（HTTPS）\*\*が必須です。
* **４－２ サーバ側脆弱性悪用：** サイトを動かしているサーバーのプログラム自体の欠陥を突かれる攻撃。
  + **対策：** 安全なプログラミング手法（プレースホルダなど）の実践が必要です。
* **４－３ フィッシング：** 有名企業を装った偽サイトにユーザーを誘導し、IDやパスワードを直接入力させる詐欺。
  + **対策：** 利用者への教育や、そもそも偽サイトにアクセスさせない技術が必要です。

**５ DOMの体系（SOPとの接点）**

**５－１ DOMの正体**

* **(１) 定義：** HTML文書を、JavaScriptが操作（色を変える、文字を追加するなど）できるように、\*\*部品の集合体（オブジェクトのツリー構造）\*\*として表現したものです。ウェブページの「設計図」や「骨格」のようなイメージです。

**５－２ SOPとDOMの交点**

* **(１) 同一オリジンなら自由：** 自分のサイト内（同一オリジン）であれば、JavaScriptでDOMを自由に書き換えられます。
* **(２) 異オリジンは不可：** <iframe>で埋め込んだ別サイトのDOMを直接操作しようとすると、SOPによってブロックされます（「隣の家の設計図は勝手にいじれない」）。
* **(３) 正規連携はpostMessage：** どうしても<iframe>内外で安全に通信したい場合は、postMessageという公式な「手紙のやりとり」機能を使います。受信側は、手紙が信頼できる相手（オリジン）から来たかを必ず確認する必要があります。

**６ “出題で迷わない”判定フロー（口頭で即答するための型）**

ある脆弱性がどの攻撃に該当するか迷ったときは、この流れで考えてみましょう。

**(１) 攻撃は「同じ家の中（同一オリジン）」で起きているか？**

* **Yes →** 身内の犯行だ！ **XSS**をまず疑う。対策はCSPやエスケープ。
* **No →** (2)へ。

**(２) 攻撃は、外の家（異オリジン）から何かを「読む」か、「送る」か？**

* **① 読む（情報窃取）→ SOPが原則ブロック！** 正規に読みたいなら**CORS**が必要。
* **② 送る（不正操作）→ SOPは止めない！** **CSRF**を疑う。対策はトークン。

**(３) 攻撃は、外の家を「埋め込んで」いるか？**

* **① 埋め込み(<iframe>) →** 透明に重ねてクリックさせる**クリックジャッキング**を疑う。対策はX-Frame-Options。
* **② 直接参照しようとしている →** それはSOPの基本ルールでブロックされる。DOM、Storage、Canvas汚染など。