Webセキュリティー入門

CSRF と CORS

吉村 優（Yoshimura Hikaru）

hikaru_yoshimura@r.recruit.co.jp

株式会社リクルート
Recruit Co., Ltd

July 15, 2021 @ Web セキュリティー勉強会

https://github.com/y-yu (cfa51ea)

Table of contents

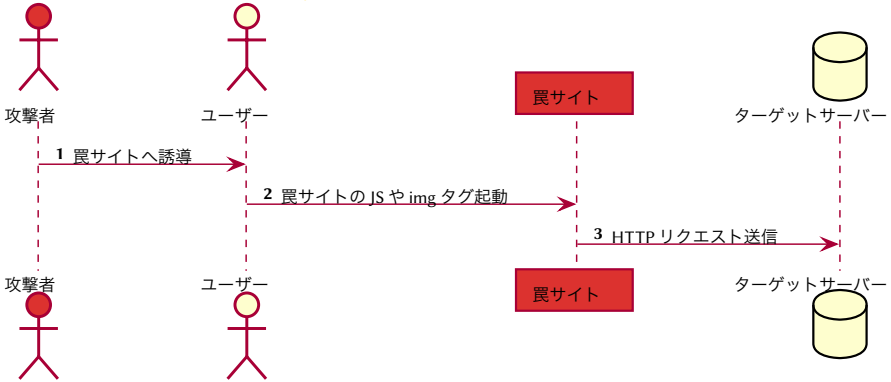
自己紹介

- 株式会社ドワンゴ（新卒）
- 株式会社リクルート（中途）
- セキュリティー・暗号
- CTF（https://urandom.team/）
  - SECCON 2014 オンライン予選 優勝
  - SECCON 2015 x CEDEC CHALLENGE ゲームクラッキング＆チートチャレンジ 優勝
  - IWSEC Cup 2015 Gold Prize
  - サイバーコロッセオ ×SECCON 2016 準優勝
  - SECCON CTF 2018 International 3rd place
  - SECCON CTF Beginners 2020 29th

Slack	@y-yu
Twitter	@_yyu_
Qiita	yyu
GitHub	y-yu

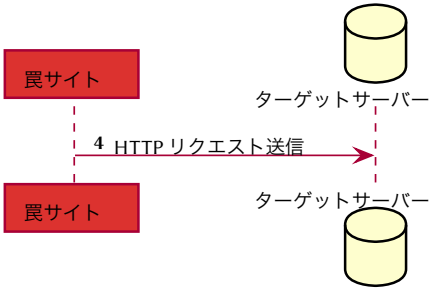
CSRF とは？

- 攻撃者がユーザーの意図しないリクエストを送信させる攻撃
- 典型的にはこんな感じ👇



- 今日はこの CSRF と、それを**クロスサイト通信**は許可しつつ防御する話

CSRF とは？



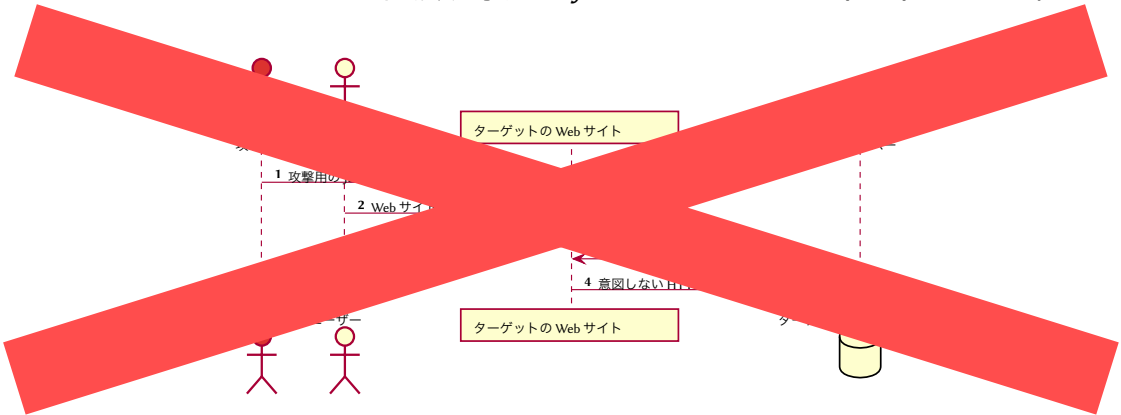
なぜ攻撃者は罠サイトを使うのか？ 🤔

JS を動かしたいけど正規のページに
<script>とかを入れられないから

- 正規のページに JS を無理やり入れる攻撃は XSS となる

CSRF *vs* XSS

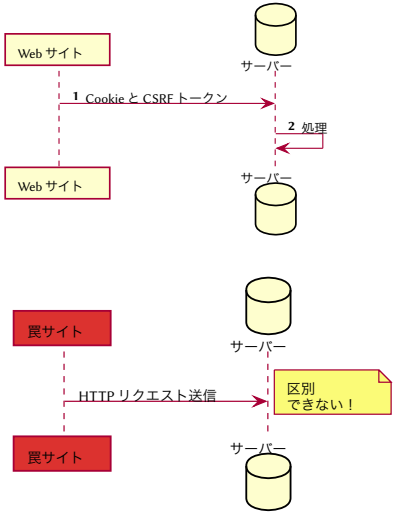
- ターゲットの Web サイトで直接攻撃用の JS を埋め込めれば（XSS）それが早い



- しかしそれができないので、攻撃者は罠サイトを利用する

CSRF トークン

- **CSRF トークン**を使えばどうだろうか？

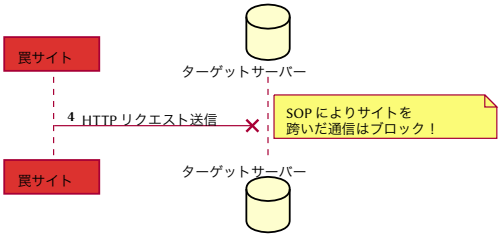


Web サイトとサーバーは別ドメインなので、
罠サイトと区別できない😇

クロスサイトの通信を基本的に禁止とするか
（過激派）

Same Origin Policy(SOP)

- Web ブラウザーはこのような罠サイトを跨いだ HTTP リクエストが攻撃に発展すると考えて、*Same Origin Policy(SOP)* を持っている
- SOP によってオリジンに跨がる *Cross-site Request* を基本的にはできない



- ただし<img>タグとかで画像を読みこんだりするのはクロスサイトでもできる
- GET で副作用が起きてしまうのは SOP の恩恵が受けられず、CSRF に発展する危険性がある

とはいえこれでは不便！ 無理やり回避するか！

JSONP(JSON with Padding)

- （他にもいろいろあるかもしれないけど）SOP を回避する**かつてあった**代表的な方法が *JSONP*（*JSON with Padding*）である

```
val json = {'json': ...} /* 渡したいJSON！  */
val callback = request.query.get('callback')
Results
  .Ok { s"$callback($json)" }
  .withHeader("application/javascript")
```

Listing: サーバー

```
<script>
function callback(json) {
  /* サーバーのJSONをつかった処理 */
}
</script>
<script src="https://server.example.com/?
     callback=callback">
```

Listing: HTML

- すると `callback( {'json': ...} )` というような JS がロードされる
  - `{'json': ...}`はサーバーが作ったデータ
- サーバーが作った JSON を JS の関数に入れて実行できる

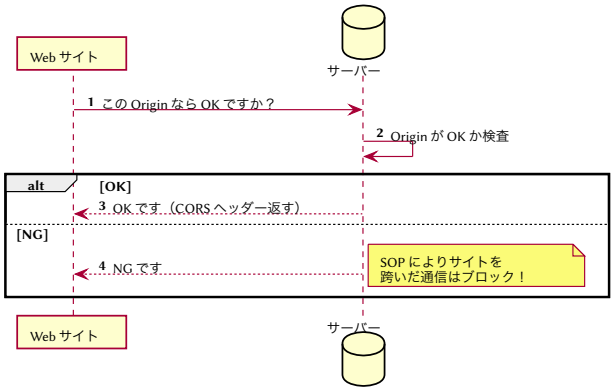
JSONP *vs* CORS

JSONP でクロスサイト通信は解決！ なぜ CORS（？）があるの？ 🤔

JSONP は XSS 脆弱性の踏み台になる😈
（この話は *Content Security Policy* などが絡み高度なので割愛）

そこでより洗練された方法が CORS！

CORS(Cross-Origin Request Sharing)

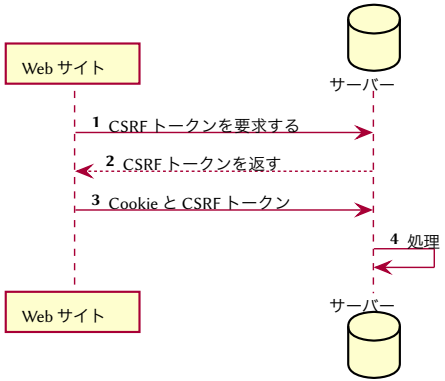


- 👈 こんな感じで事前に OK なオリジンかどうかをサーバーに問合せる
- OK なときサーバーが専用の CORS のヘッダーを返すと、ブラウザーがそれを検出して SOP による制限を緩和してくれる

- この `Origin`ヘッダーはブラウザーが付けるので攻撃者が改竄することはできない

CSRF トークン

- 冒頭で紹介した CSRF トークンは XHR や Fetch API ではない `form`タグなどで利用する
  - このときにクロスドメインなら CORS を使う
- ブラウザーは XHR などにしか `Origin`ヘッダーを付与しない



まとめ

- CORS は CSRF の理解が前提となるし、かつ JSONP がなぜ死んだのかも知っておかないといけないから難しい
- JSONP が XSS の踏み台になるなど、CSRF と XSS は実は微妙に関係している
  - とはいえ基本的には別ものな脆弱性
- CSRF は XSS より自由度が低いけど、サーバーの API 呼び出しによって可能な任意の操作ができてしまう危険な脆弱性
  - 任意コード実行のような雰囲気で考えている
- XSS については別の機会で紹介したい

参考文献

[1] 米内貴志.
Web ブラウザセキュリティ—Web アプリケーションの安全性を支える仕組みを整理する.
ラムダノート, 単行本（ソフトカバー）, 1 2021.

Thank you for the attention!