OWASP
Open Web Application Security Project

Secure JavaScript Programming

OWASP Kansai Chapter
Yosuke HASEGAWA

自己紹介

はせがわようすけ

- OWASP Kansai チャプターリーダー
- OWASP Japan アドバイザリボードメンバー
- 株式会社セキュアスカイ・テクノロジー 常勤技術顧問
- CODE BLUE Security Conference Review board member
- セキュリティキャンプ講師 (Webクラス/高レイヤートラック)
- http://utf-8.jp/
- Author of jjencode and aaencode

OWASP Kansai

- 自分たちの直面するWebセキュリティの問題を自分たちの手で解決したい
  - 日本で2番目のOWASPローカルチャプター
  - 2014年3月から活動開始
  - 3か月に1回のChapter Meeting (勉強会)を開催
    - Webセキュリティの悩み事を気楽に相談し情報共有できる場
    - スキル、役職、業種、国籍、性別、年齢関係なく、遠慮なくお越しください

OWASP Kansai

JavaScriptのセキュリティ

JavaScriptのセキュリティ

JavaScriptに関連するセキュリティ問題

- JavaScriptによるオープンリダイレクタ
- DOM-based XSS
- CORSの設定不備
- クライアントサイドでの不適切なデータ保存
- その他DOM APIの不適切な使用

JavaScriptのセキュリティ

JavaScriptに関連するセキュリティ問題

- JavaScriptによるオープンリダイレクタ
- **DOM-based XSS**
- CORSの設定不備
- クライアントサイドでの不適切なデータ保存
- その他DOM APIの不適切な使用

脆弱性の発生が
圧倒的に多い

OWASP
Open Web Application
Security Project

XSSのおさらい

XSSおさらい

- 対象
  - 動的にHTMLを生成するWebアプリ
- 問題
  - 攻撃者が用意したスクリプトがHTML内に挿入される
- 対策
  - HTMLを生成する時点でエスケープ
  - URLはhttp/httpsのみに限定する

XSSおさらい

http://shop.example.jp/
?item="><script>...

GET /?item="><script>...

<input type="text"
value=""><script>...

HTML生成時に
エスケープされていない

Web
サイト

XSSの被害
JSでできることは何でも。

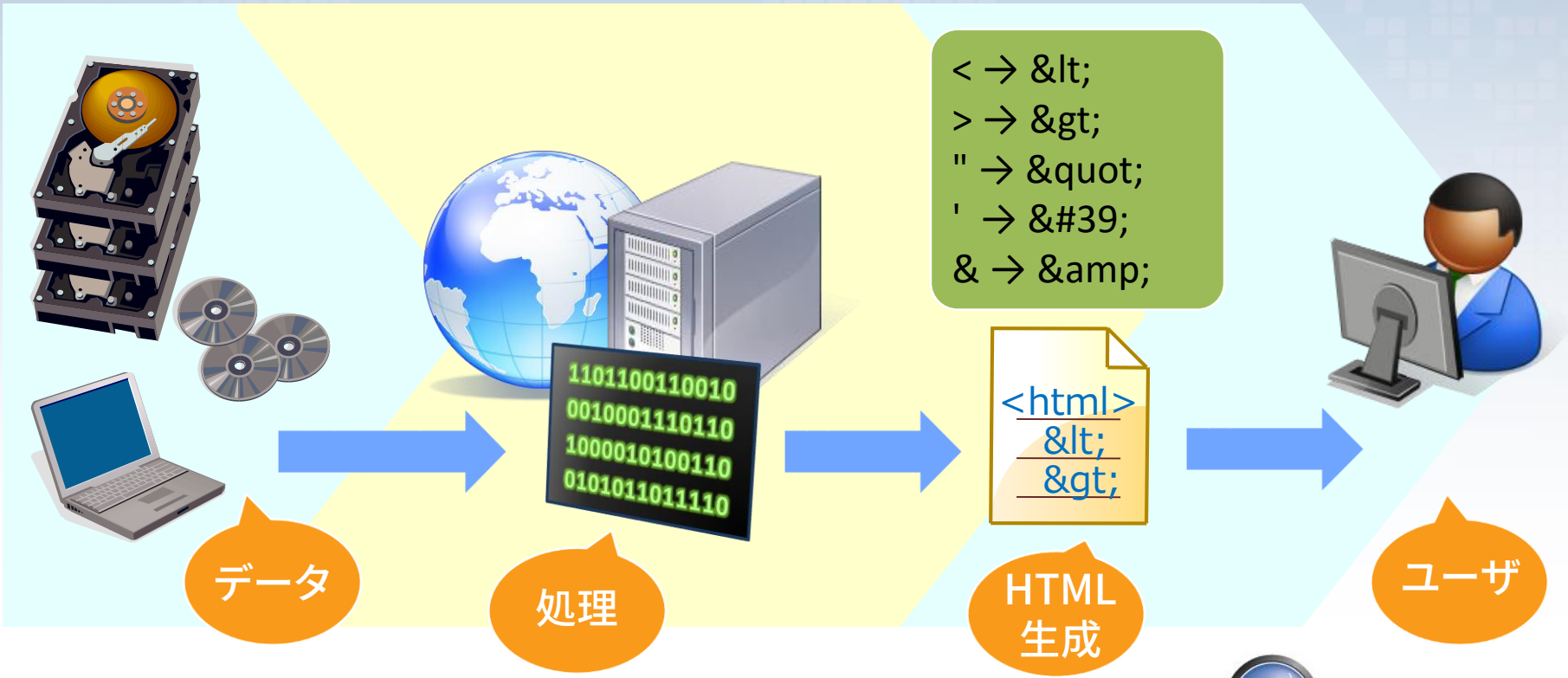
- 偽情報の表示
- Cookie情報の漏えい
- 機密情報の漏えい
- その他

XSSおさらい

- 対象
  - 動的にHTMLを生成するWebアプリ
- 問題
  - 攻撃者が用意したスクリプトがHTML内に挿入される
- **対策**
  - **HTMLを生成する時点でエスケープ**
  - **URLはhttp/httpsのみに限定する**

XSSおさらい

- HTMLを生成する時点でエスケープする



OWASP
Open Web Application
Security Project

XSSおさらい

- URLはhttp/httpsのみに限定する
  - src,hrefなどの属性値の動的生成

```
<a href="javascript:alert(1)">
<iframe src="data:text/html;base64,
 PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4K">
```

  - URLの動的生成時はhttp,https限定とする

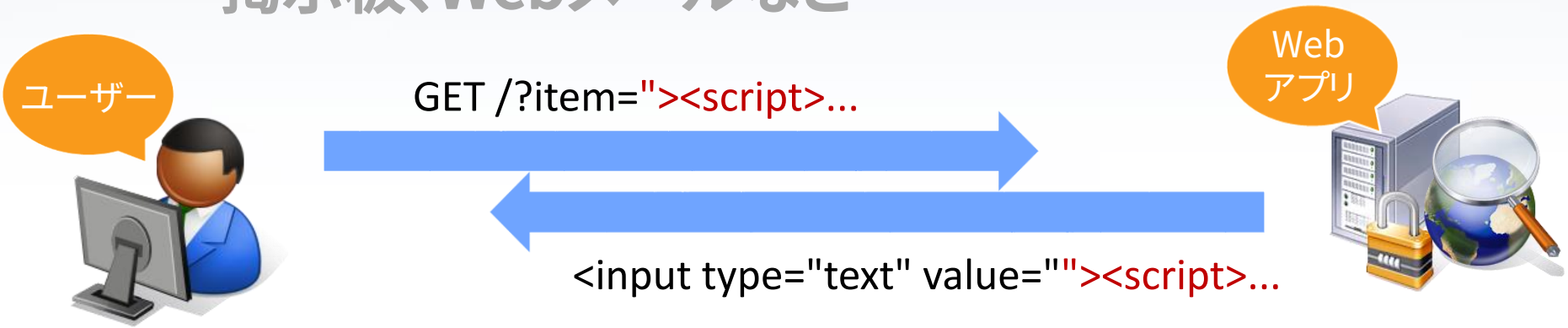
XSSの種類

XSSの種類

- 反射型XSS
  - ユーザーからの送信内容をそのまま表示
  - お問い合わせフォーム、検索フォームなど
  - XSSフィルタである程度防御
- 蓄積型XSS
  - 攻撃者のスクリプトをサーバ内で保持
  - 掲示板、Webメールなど

XSSの種類

- 反射型XSS
  - ユーザーからの送信内容をそのまま表示
  - お問い合わせフォーム、検索フォームなど
  - XSSフィルタである程度防御
- 蓄積型XSS
  - 攻撃者のスクリプトをサーバ内で保持
  - 掲示板、Webメールなど



反射型XSS

- リクエストとレスポンスに同じ内容が含まれる

```
GET /?<script>alert(1)</script> HTTP/1.1
Host: example.jp

HTTP/1.1 200 OK
Content-Type: text/hthml; charst=utf-8

<html>
<body>
<script>alert(1)</script>
</body>
```

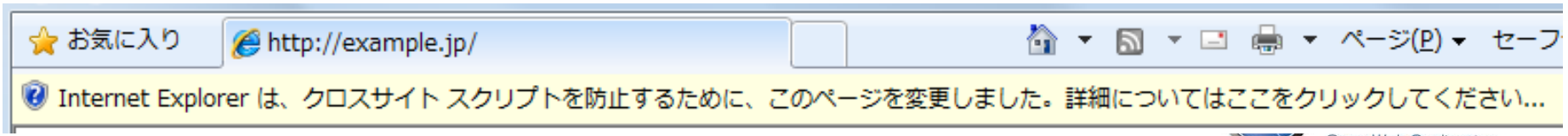
反射型XSS

- リクエストとレスポンスに同じ内容が含まれる

```
GET /?<script>alert(1)</script> HTTP/1.1
Host: example.jp

HTTP/1.1 200 OK
Content-Type: text/hthml; charst=utf-8

<html>
<body>
<script>alert(1)</script>
</body>
```



Open Web Application Security Project

XSSの種類

- 反射型XSS
  - ユーザーからの送信内容をそのまま表示
  - お問い合わせフォーム、検索フォームなど
  - XSSフィルタである程度防御
- **蓄積型XSS**
  - **攻撃者のスクリプトをサーバ内で保持**
  - **掲示板、Webメールなど**

蓄積型XSS

- サーバに攻撃者のスクリプトが保存される
  - 攻撃の永続化
  - 攻撃と被害に時間差
  - 反射型より影響が大きい

XSSの種類

- 反射型XSS
  - ユーザーからの送信内容をそのまま表示
  - お問い合わせフォーム、検索フォームなど
  - XSSフィルタである程度防御
- 蓄積型XSS
  - 攻撃者のスクリプトをサーバ内で保持
  - 掲示板、Webメールなど

XSSの種類

- 反射型XSS
  - ユーザーからの送信内容をそのまま表示
  - お問い合わせフォーム、検索フォームなど
  - XSSフィルタである程度防御
- 蓄積型XSS
  - 攻撃者のスクリプトをサーバ内で保持
  - 掲示板、Webメールなど
- **DOM-based XSS**
  - **JavaScriptが引き起こすXSS**
  - **サーバ側のHTML生成では問題なし**

DOM-based XSS

DOM-based XSS

- JavaScriptが引き起こすXSS
- サーバ側のHTML生成時には問題なし
- JavaScriptによるHTMLレンダリング時の問題

```
//http://example.jp/#<img src=0 onerror=alert(1)>
<html>
<script>
    document.write( location.hash.substring(1) );
</script>
</html>
```

- JavaScriptの利用に合わせて増加

OWASP Open Web Application Security Project

DOM-based XSS

- ブラウザのXSSフィルタを通過することが多い
- location.hash内などの実行コードはサーバ側にログが残らない

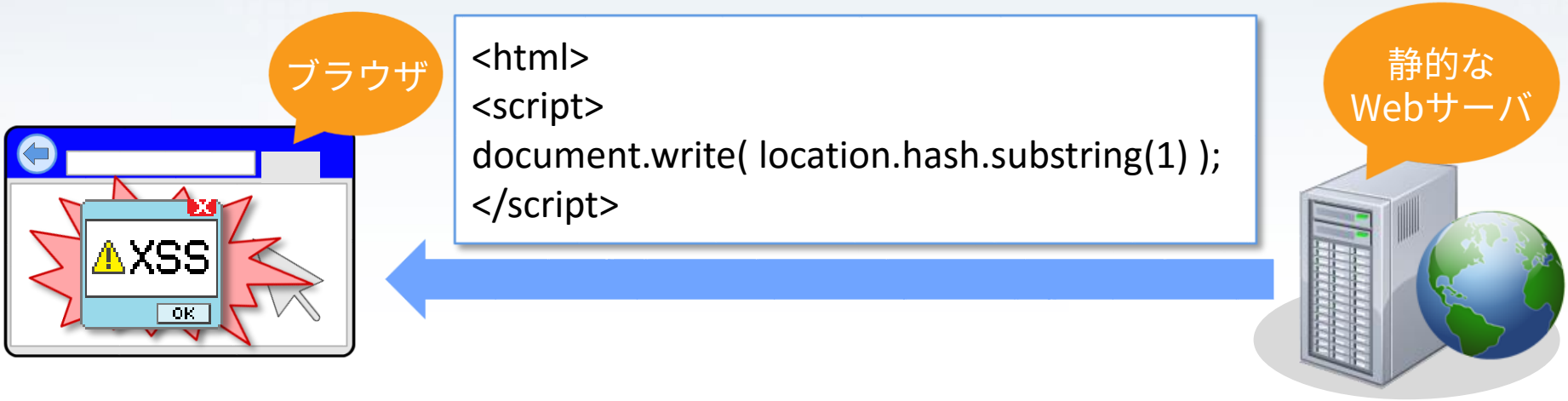
```
//http://example.jp/#<img src=0 onerror=alert(1)>
<html>
<script>
    document.write( location.hash.substring(1) );
</script>
</html>
```

DOM-based XSS

- JavaScriptが実行されるまでXSSの存在がわからない
  - 既存の検査ツールでは検出不可な場合も
  - 生成されるHTML自体には問題はない
  - リクエスト/レスポンスの監視だけでは見つからない

DOM-based XSS

- 静的コンテンツのみでもXSSする可能性
  - 動的にHTMLを生成する「Webアプリケーション」ではなく、*.htmlしか提供してなくてもXSSのある可能性がある

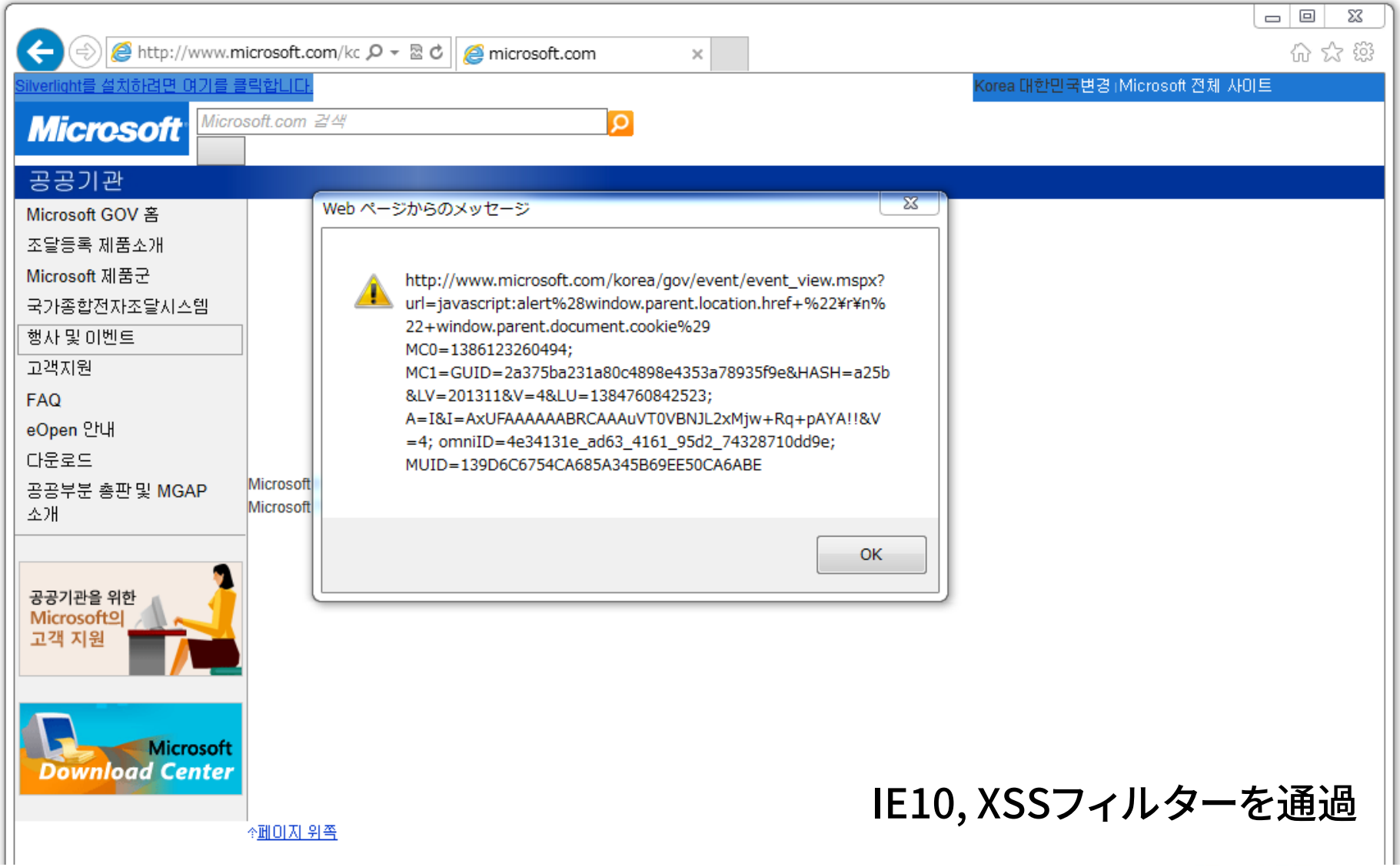


DOM-based XSS

- 攻撃者はJavaScriptを読むことができる
  - じっくり読んで脆弱性を探すことが可能
  - 脆弱性の有無を確認するための試行リクエストは不要
  - 「一撃必殺」でXSSを成功させる

OWASP
Open Web Application Security Project

DOM-based XSS



IE10, XSSフィルターを通過

DOM-based XSS

- 圧倒的に不利な状況
  - JavaScriptコード量の大幅な増加
  - XSSフィルタを通過することがある
  - サーバのログに残らないことがある
  - これまでの検査方法では見つからない
  - 静的コンテンツでもXSSする
  - 攻撃者は時間をかけてXSSを探す
- 開発時点で作りこまない必要性

DOM-based XSS
原因と対策

DOM-based XSS 原因

- 原因
  - 攻撃者の与えた文字列が
  - JavaScript上のコードのどこかで
  - 文字列からHTMLを生成 あるいは JavaScript コードとして実行される

```
//http://example.jp/#<img src=0 onerror=alert(1)>
<html>
<script>
    document.write( location.hash.substring(1) );
</script>
</html>
```

DOM-based XSS 原因

- 原因
  - 攻撃者の与えた文字列が
  - JavaScript上のコードのどこかで
  - 文字列からHTMLを生成 あるいは JavaScript コードとして実行される

```
//http://example.jp/#<img src=0 onerror=alert(1)>
<html>
<script>
    document.write( location.hash.substring(1) );
</script>
</html>
```

DOM-based XSS 原因

- 原因
  - 攻撃者の与えた文字列が
  - JavaScript上のコードのどこかで
  - 文字列からHTMLを生成 あるいは JavaScript コードとして実行される

```
//http://example.jp/#<img src=0 onerror=alert(1)>
<html>
<script>
    document.write( location.hash.substring(1) );
</script>
</html>
```

DOM-based XSS 原因

- 原因
  - 攻撃者の与えた文字列が
  - JavaScript上のコードのどこかで
  - 文字列からHTMLを生成 あるいは JavaScript コードとして実行される

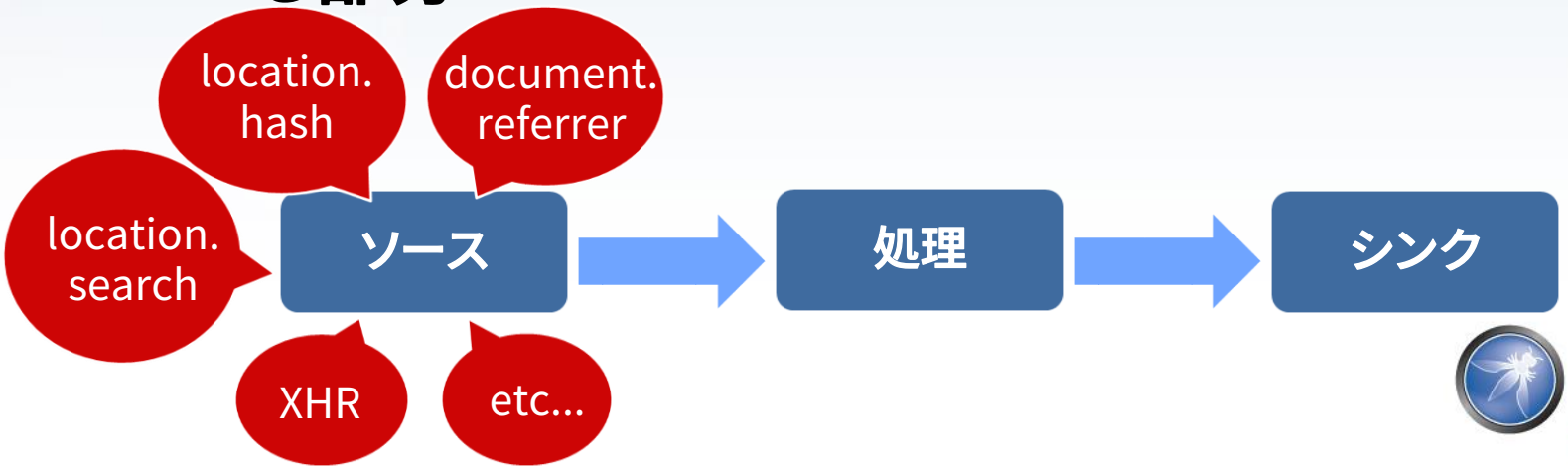
```
//http://example.jp/#<img src=0 onerror=alert(1)>
<html>
<script>
    document.write( location.hash.substring(1) );
</script>
</html>
```

DOM-based XSS 原因

- ソース
  - 攻撃者の与えた文字列の含まれる箇所
- シンク
  - 文字列からHTMLを生成したりコードとして実行する部分

DOM-based XSS 原因

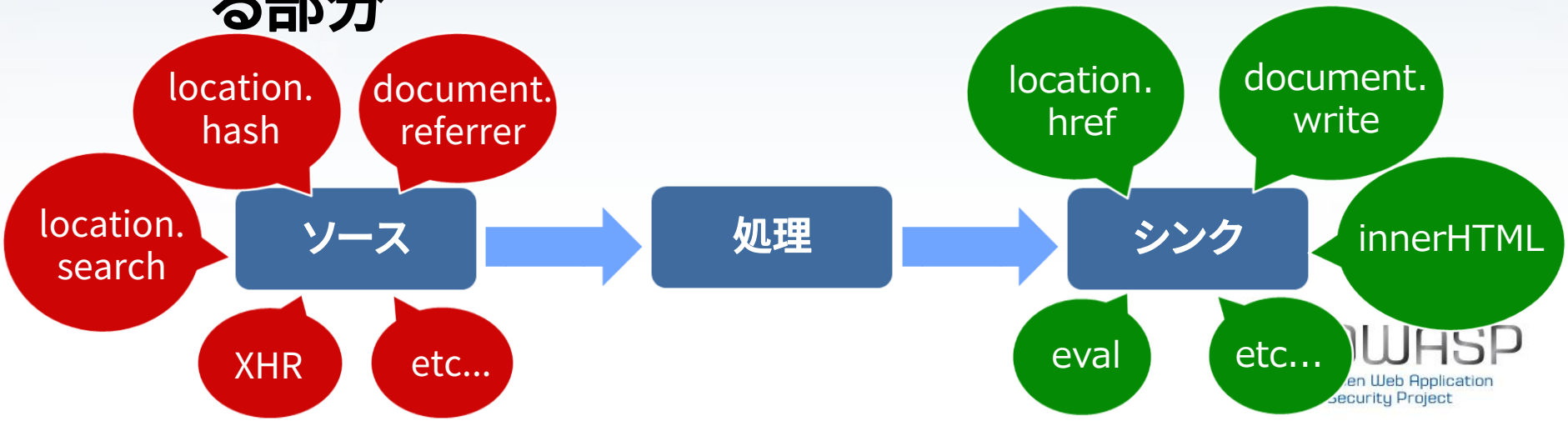
- ソース
  - 攻撃者の与えた文字列の含まれる箇所
- シンク
  - 文字列からHTMLを生成したりコードとして実行する部分



OWASP
Open Web Application
Security Project

DOM-based XSS 原因

- ソース
  - 攻撃者の与えた文字列の含まれる箇所
- シンク
  - 文字列からHTMLを生成したりコードとして実行する部分



DOM-based XSS 対策

- 対策
  - HTML生成時にエスケープ/適切なDOM操作
  - URLの生成時はhttp(s)に限定
  - 使用しているライブラリの更新
- サーバ側でのXSS対策と同じ
  - これまでサーバ上で行っていたことをJavaScript上で行う

DOM-based XSS 対策

- 対策
  - **HTML生成時にエスケープ/適切なDOM操作**
  - URLの生成時はhttp(s)に限定
  - 使用しているライブラリの更新
- サーバ側でのXSS対策と同じ
  - これまでサーバ上で行っていたことをJavaScript上で行う

DOM-based XSS 対策

- HTML生成時に適切なDOM操作
  - JavaScriptでレンダリングされる直前
  - 「エスケープ」ではなく適切なDOM操作関数

```
// bad code
document.write( location.hash.substring( 1 ) );
```

```
var text = document.createTextNode(
    location.hash.substring( 1 )
);
document.body.appendChild( text );
```

DOM-based XSS 対策

- テキストノードだけでなく属性値も

```
// bad code
var text = "....";    //変数textは攻撃者がコントロール可能
form.innerHTML =
 '<input type="text" name="key" value="' + text + '">';
```

<input ... value=""><script>....</script "">

```
var text = "....";    //変数textは攻撃者がコントロール可能
var elm = document.createElement( "input" );
elm.setAttribute( "type", "text" );
elm.setAttribute( "name", "key" );
elm.setAttribute( "value", text );  // 属性値を設定する
form.appendChild( elm );
```

Security Project

DOM-based XSS 対策

- HTML生成時に適切なDOM操作関数
  - テキストノードの生成
    createTextNode, innerText, textContent
  - 属性の設定
    setAttribute
- シンクとなるAPIを不用意に使用しない
  - innerHTML, document.write, ...

DOM-based XSS 対策

- 対策
  - HTML生成時にエスケープ/適切なDOM操作
  - **URLの生成時はhttp(s)に限定**
  - 使用しているライブラリの更新
- サーバ側でのXSS対策と同じ
  - これまでサーバ上で行っていたことをJavaScript上で行う

DOM-based XSS 対策

- URLの生成時はhttp(s)に限定

```
//bad code
// <a id="link">リンク</a>
var url = "....";    //変数textは攻撃者がコントロール可能
var elm = document.getElementById( "link" );
elm.setAttribute( "href", url );
```

```
<a id="link" href=" javascript:alert(1)">リンク</a>
```

```
// urlが「http://」「https://」で始まる場合のみに限定
if( url.match( /^https?:¥/¥// ) ){
    var elm = document.getElementById( "link" );
    elm.setAttribute( "href", url );
}
```

DOM-based XSS 対策

- URLの生成時はhttp(s)に限定
  - 他のスキームが入り込まないように。
    javascript:, vbscript:, data:,
- <a>要素だけでなくlocationオブジェクトの操作時にも注意

```
// bad code
var url = "javascript:alert(1)";
location.href = url;        // XSS
location.assign( url );     // XSS
```

DOM-based XSS 対策

- 対策
  - HTML生成時にエスケープ/適切なDOM操作
  - URLの生成時はhttp(s)に限定
  - **使用しているライブラリの更新**
- サーバ側でのXSS対策と同じ
  - これまでサーバ上で行っていたことをJavaScript上で行う

DOM-based XSS 対策

- 使用してるライブラリの更新
  - JavaScriptライブラリの脆弱性対応
  - 使用しているJSライブラリの更新を把握すること

Masato Kinugawa Security Blog: jQuery Mobile 1.2 Beta未満は読み込んでいるだけでXSS脆弱性を作ります
http://masatokinugawa.l0.cm/2012/09/jquery-mobile-location.href-xss.html

- サーバ側のミドルウェア等の運用と同じ

DOM-based XSS 対策

- 対策
  - HTML生成時にエスケープ/適切なDOM操作
  - URLの生成時はhttp(s)に限定
  - 使用しているライブラリの更新
- サーバ側でのXSS対策と同じ
  - これまでサーバ上で行っていたことをJavaScript上で行う

DOM-based XSS 対策 [応用編]

DOM-based XSS 対策 [応用編]

- DbXSS対策の原則(再掲)
  - 「HTML生成時にエスケープ/適切なDOM操作」
  - URLの生成時はhttp(s)に限定

DOM-based XSS 対策 [応用編]

- DbXSS対策の原則(再掲)
  - 「HTML生成時にエスケープ/適切なDOM操作」
  - URLの生成時はhttp(s)に限定
- 原則だけでは立ちいかない現実
  - 一部のHTMLタグは許容したい
  - 相対URLも使いたい

OWASP
Open Web Application Security Project

DOM-based XSS 対策 [応用編]

- 一部のタグだけは許容したい
  - 装飾やリンクのためのHTMLタグ

```
var s = "**注意** 雨天時は[こちら](http://example.jp/)です。";
```

```
<b>注意</b> 雨天時は<a href="http://example.jp/">こちら</a>です。
```

- 相対URLも使いたい

```
<a href="/next-page">next</a>  // 相対リンクをJSでも生成したい
```

OWASP
Open Web Application Security Project

DOM-based XSS 対策 [応用編]

- 一部のタグだけは許容したい
  - 特定書式の繰り返し。テンプレート的な用途

```
[
  {
    "date" : "2015/09/10",
    "url" : "http://example.jp/news",
    "title" : "新製品発表のお知らせ"
  },
  {
    "date" : "2015/09/19",
    "url" : "http://example.jp/owasp",
    "title" : "Local Chapter Meeing開催"
  }
]
```

```
<div>
    <span>2015/09/10</span>
    <a href="http://example.jp/news">
        新製品発表のお知らせ
    </a>
</div>
<div>
    <span>2015/09/19</span>
    <a href="http://example.jp/owasp">
        Local Chapter Meeting開催
    </a>
</div>
```

  - ユーザーによる自由な入力

```
var markdown = "**注意** 雨天時は[こち
ら](http://example.jp/)です。";
```

```
<b>注意</b> 雨天時は<a
href="http://example.jp/">こちら</a>です。
```

DOM-based XSS 対策 [応用編]

- 一部のタグだけは許容したい
  - 特定書式の繰り返し。テンプレート的な用途

```
[
  {
    "date" : "2015/09/10",
    "url" : "http://example.jp/news",
    "title" : "新製品発表のお知らせ"
  },
  {
    "date" : "2015/09/19",
    "url" : "http://example.jp/owasp",
    "title" : "Local Chapter Meeing開催"
  }
]
```

```
<div>
    <span>2015/09/10</span>
    <a href="http://example.jp/news">
        新製品発表のお知らせ
    </a>
</div>
<div>
    <span>2015/09/19</span>
    <a href="http://example.jp/owasp">
        Local Chapter Meeting開催
    </a>
</div>
```

  - ユーザーによる自由な入力

```
var markdown = "**注意** 雨天時は[こち
ら](http://example.jp/)です。";
```

```
<b>注意</b> 雨天時は<a
href="http://example.jp/">こちら</a>です。
```

DOM-based XSS 対策 [応用編]

- 一部のタグだけは許容したい
  - 特定書式の繰り返し。テンプレート的な用途

```
[
  {
    "date" : "2015/09/10",
    "url" : "http://example.jp/news",
    "title" : "新製品発表のお知らせ"
  },
  {
    "date" : "2015/09/19",
    "url" : "http://example.jp/owasp",
    "title" : "Local Chapter Meeing開催"
  }
]
```

```
<div>
    <span>2015/09/10</span>
    <a href="http://example.jp/news">
        新製品発表のお知らせ
    </a>
</div>
<div>
    <span>2015/09/19</span>
    <a href="http://example.jp/owasp">
        Local Chapter Meeting開催
    </a>
</div>
```

  - HTMLの構造は固定
  - 属性値やテキストノードの部分を動的に生成

DOM-based XSS 対策 [応用編]

- 特定書式の繰り返し。テンプレート的な用途
  - 自分でテンプレート処理を書く?

```
// bad code
function expandTemplate( template, json ){
  var i, s, html = "";
  for( i = 0; i < friends.length; i++ ){
    s = template.replace( /%(¥w+)%/g, function( s, param ){
      if( param === "date" ) return htmlEscape( json[ i ].date );
      else if( param === "url" ) return htmlEscape( json[ i ].url );
      else if( param === "title" ) return htmlEscape( json[ i ].title );
      else return "%" + param + "%";
    } );
    html += s;
  }
  return html;
}

elm.innerHTML = expandTemplate( '<div>' +
    '<span>%date%</span><a href="%url%">%title%</a>', json );
```

DOM-based XSS 対策 [応用編]

- テンプレート処理を自分で書くのはやめるべき
  - 汎用性に欠けるのに見通しの悪いコードが増える
  - 細かな対策全てを自分でケアする必要がある
    - テキストノードにエスケープが必要
    - URLにjavascript:スキーム等が混入しないように注意がいる

DOM-based XSS 対策 [応用編]

- JSのテンプレートエンジンライブラリの導入
- MV*フレームワークの採用 (vue,knockoutなど)
- 各ライブラリの挙動を把握して使用すること
  - テキストノードへ出力するときにエスケープされるか(vueのv-textとv-htmlの違い等)
  - 属性値にjavascript:スキーム等が設定された場合にどうなるか

DOM-based XSS 対策 [応用編]

- 一部のタグだけは許容したい
  - 特定書式の繰り返し。テンプレート的な用途

```
[
  {
    "date" : "2015/09/10",
    "url" : "http://example.jp/news",
    "title" : "新製品発表のお知らせ"
  },
  {
    "date" : "2015/09/19",
    "url" : "http://example.jp/owasp",
    "title" : "Local Chapter Meeing開催"
  }
]
```

```
<div>
    <span>2015/09/10</span>
    <a href="http://example.jp/news">
        新製品発表のお知らせ
    </a>
</div>
<div>
    <span>2015/09/19</span>
    <a href="http://example.jp/owasp">
        Local Chapter Meeting開催
    </a>
</div>
```

  - ユーザーによる自由な入力

```
var markdown = "**注意** 雨天時は[こち
ら](http://example.jp/)です。";
```

```
<b>注意</b> 雨天時は<a
href="http://example.jp/">こちら</a>です。
```

DOM-based XSS 対策 [応用編]

- 一部のタグだけは許容したい
- 特定書式の繰り返し。テンプレート的な用途

```
[
  {
    "date" : "2015/09/10",
    "url" : "http://example.jp/news",
    "title" : "新製品発表のお知らせ"
  },
  {
    "date" : "2015/09/19",
    "url" : "http://example.jp/owasp",
    "title" : "Local Chapter Meeing開催"
  }
]
```

```
<div>
    <span>2015/09/10</span>
    <a href="http://example.jp/news">
        新製品発表のお知らせ
    </a>
</div>
<div>
    <span>2015/09/19</span>
    <a href="http://example.jp/owasp">
        Local Chapter Meeting開催
    </a>
</div>
```

- **ユーザーによる自由な入力**

```
var markdown = "**注意** 雨天時は[こち
ら](http://example.jp/)です。";
```

```
<b>注意</b> 雨天時は<a
href="http://example.jp/">こちら</a>です。
```

DOM-based XSS 対策 [応用編]

- ユーザーによる自由な入力
- HTML構造が事前に定義されていない
- 安全なタグや属性は許可、それ以外を禁止
  - Webメール
  - 掲示板などのリッチエディット
  - markdown

OWASP
Open Web Application Security Project

DOM-based XSS 対策 [応用編]

- ユーザーによる自由な入力
  - 安全なタグや属性だけ許可、それ以外を禁止

```
<div>こんにちは</div>
<script>alert(1)</script>
<img src=# onerror=alert(1)>
<s>取り消し線</s>
```

```
<div>こんにちは</div>

<img src=#>
<s>取り消し線</
```

攻撃者

Web
アプリ

ユーザー

DOM-based XSS 対策 [応用編]

- 危険そうな属性やタグを全て削除する?

```
// bad code
function safeHtml( html ){
  return html
    .replace( /<script>/ig, "")
    .replace( /onerror=/ig, "" )
    .replace( /onload=/ig, "" ) ...
}

var elm.innerHTML = safeHtml(
  "<div>こんにちは</div>" +
  "<script>alert(1)</script>" +
  "<img src=# onerror=alert(1)>" +
  "<s>取り消し線</s>"
);
```

DOM-based XSS 対策 [応用編]

- 危険そうな属性やタグを全て削除する?

```
// bad code
function safeHtml( html ){
  return html
    .replace( /<script>/ig, "")
    .replace( /onerror=/ig, "" )
    .replace( /onload=/ig, "" ) ...
}

var elm.innerHTML = safeHtml(
  "<s<script>cript>alert(1)</script>"
); // <script>alert(1)</script>
```

- このアプローチでは絶対に抜けが発生する

DOM-based XSS 対策 [応用編]

- 安全なタグ、要素だけでHTMLを組み立てなおす
  - 安全なタグ、属性を事前に定めておく
  - 文字列をHTMLとしてパースする
  - 安全なタグ、属性のみでHTMLを再生成する
- といったことを自分でやるのはしんどいので、ライブラリに任せる

DOM-based XSS 対策 [応用編]

- 安全なタグ、要素だけでHTMLを組み立てなおす
  - DOMPurify

https://github.com/cure53/DOMPurify

```
<script src="purify.js"></script>
....
var html =
  "<div>こんにちは</div>" +
  "<script>alert(1)</script>" +
  "<img src=# onerror=alert(1)>" +
  "<s>取り消し線</s>";
elm.innerHTML = DOMPurify.sanitize( html );
```

```
<div>こんにちは</div>

<img src="#">
<s>取り消し線</s>
```

DOM-based XSS 対策 [応用編]

- 相対URLも使いたい

```
// urlが「http://」「https://」で始まる場合に <a id="link">を設定
function setLink( url ){
  if( url.match( /^https?:¥/¥// ) ){
      var elm = document.getElementById( "link" );
      elm.setAttribute( "href", url );
      elm.textContent = url;
  }
}

setLink( "http://example.jp/" );  // ok
setLink( "javascript:alert(1)" ); // ng
setLink( "/foo" ); // ???
setLink( "foo" );  // ???
```

DOM-based XSS 対策 [応用編]

- 相対URLを絶対URLに正規化する
  - URLUtilsインターフェース

```
// Chrome, Firefoxのみ
var url = new URL( "/foo", location.href );
console.log( url.href );
```

  - a要素で代用

```
// IE向け
var a = document.createElement( "a" );
a.setAttribute( "href", "/foo" );
console.log( a.href );
```

DOM-based XSS 対策 [応用編]

```
function getAbsoluteUrl( url ){
    var elm = document.createElement( "a" );
    // hrefプロパティが絶対URLを返すかテストする
    elm.setAttribute( "href", "/test" );
    if( elm.href === "/test" ){
        // IE6, IE7
        elm.setAttribute( "href", url );
        return elm.getAttribute( "href", 4 );
    }else{
        elm.setAttribute( "href", url );
        return elm.href;
    }
}
```

```
function parseUrl( url ){
    try{
        // URLコンストラクタが使用できる場合はそのまま使用する
        var result = new URL( url );
        return result;
    }
    catch( e ){
        // URLコンストラクタが使用できない場合は<a>要素を使用する
        var elm = document.createElement( "a" );
        // IEでは相対URLをhref属性に設定した場合にいくつかのプロパティが正しく
        // 取得できないため、いったん絶対URLに変換してからhref属性に設定する
        elm.setAttribute( "href", getAbsoluteUrl( url ) );
        var result = {
            protocol: elm.protocol,
            host: elm.host,
            hostname: elm.hostname,
            port: elm.port,
            pathname: elm.pathname,
            search: elm.search,
            hash: elm.hash,
            href: elm.href,
            origin: elm.origin
        };
        if( elm.protocol === "http:" ){
            result.host = result.host.replace( /:80$/, "" );
        }else if( elm.protocol === "https:" ){
            result.host = result.host.replace( /:443$/, "" );
        }
        if( result.origin === undefined ){
            result.origin = result.protocol + "//" + result.host;
        }
        return result;
    }
}
```

DOM-based XSS 対策 [応用編]

- 相対URLを絶対URLに変換したのちにプロトコルを確認する

```
var target = parseUrl( "javascript:alert(1)" );
if( target.protocol.match( /^https?:/ ) ){
    /* http、httpsなURLなので処理する */
}
```

OWASP
Open Web Application Security Project

DOM-based XSS 対策 [応用編]

- DbXSS対策に自信がない。万が一に備えたい
  - DbXSSが発生しても被害が及ばないようにする
- HTML5 iframe sandboxが利用可能

```
<iframe sandbox seamless
  style="border-width:0px" id="f"></iframe>

document.getElementById("f").srcdoc =
    "<div><img src=# onerror=alert(1)></div>";
```

  - sandbox属性を付与することでJSの実行が禁止される

DOM-based XSS 対策 [応用編まとめ]

- 一部のHTMLタグは許容したい
  - 特定書式の繰り返しにはテンプレートライブラリやMV*フレームワークを用いる
    - それらのエスケープ有無を把握すること
    - リンクがhttp/httpsに限定されるかを確認すること
- ユーザーによる自由なHTMLタグ入力を許容したい
  - DOMPurifyのようなライブラリを用いる
- 相対URLを使いたい
  - 絶対URLへ変換したのちにプロトコルスキームを確認する
- 保険的対策
  - iframe sandboxの活用

DOM-based XSS 探し方

DOM-based XSS 探し方

- スキャナで静的解析
  - AppScan (よく知らない)
- スキャナで動的解析
  - DOMinatorPro (よく知らない)
- JavaScriptのソースコードを読む
  - 一番効果的?
  - 無料!
  - 攻撃者もできる

OWASP
Open Web Application Security Project

DOM-based XSS 探し方

- ブラウザのデバッガでシンクの使用箇所を探す
  - innerHTML, locationなどが多い
- シンクがコントロール可能か調べる
  - シンク→ソースへの経路を上っていく
- 実際に試す

DOM-based XSS 探し方

- ソースコードを読みながら探す
  - デバッグと同じ技術が要求される
  - 開発者としての能力が要求される
  - 難読化(minify)されているJSを読む能力

まとめ

まとめ

- DOM-based XSSの脅威の増加
  - 攻撃可能箇所の増加
  - 脆弱性診断の技術の不足
- 攻撃者有利な状況
  - 脆弱性を作りこまない必要性

質問

hasegawa@utf-8.jp
hasegawa@securesky-tech.com

@hasegawayosuke

http://utf-8.jp/

OWASP
Open Web Application Security Project