OWASP
Open Web Application
Security Project

JavaScript セキュリティ

2017-02-15 第12回OWASP Sendaiミーティング

OWASP Kansai Chapter
Yosuke HASEGAWA

自己紹介

長谷川陽介 (はせがわようすけ / @hasegawayosuke)

- 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 CTO
- セキュリティキャンプ講師(2008~)
- CODE BLUE カンファレンスレビューボード
- http://utf-8.jp/

OWASP Kansai チャプターミーティング

- 2017-03-31 京都
  https://owasp-kansai.doorkeeper.jp/events/57646

JavaScriptセキュリティ

- gihyo.jp 連載
  「JavaScriptセキュリティの基礎知識」
  - http://gihyo.jp/dev/serial/01/javascript-security

なぜJavaScriptなのか

- ブラウザの高機能化
  - HTML5による表現力の向上
  - JavaScriptの処理速度の向上
- JavaScriptプログラミング効率の向上
  - 言語仕様の充実化
  - プログラミング環境の改善
- 実行コードのブラウザ上へのシフト
  - ネイティブアプリからWebアプリへ
  - 従来サーバ側で行っていた処理がクライアントのJavaScript上へ

セキュリティ対策もフロントエンドへ

- 脆弱性もフロントエンドで増加
  - JavaScriptコード量や扱うデータが増加
  - 比例して脆弱性も増加
  - XSSやCSRFなどの比重が増加

- Web開発者であるからにはフロントエンドの知識も要求されて普通という時代へ
  - 今だからこそのJavaScript
  - 当然、セキュリティに関連する技術も必要
  - サーバサイドでもセキュアなAPIのデザインなど

フロントエンドでのセキュリティ問題

- ブラウザ上で発生する脆弱性
  - オープンリダイレクタ
  - DOM-based XSS
  - CSRF
  - Ajaxデータの漏えい
  - クライアントサイドでの不適切なデータ保存
  - DOM APIの不適切な使用
  - などなど…
- サイトを訪問することによって発生
  - すなわち受動的攻撃

フロントエンドのセキュリティ対策

- 攻撃側は新しいWeb技術をもっとも活用できる
  - 新しいブラウザの機能、新しいHTML要素、新しいJS API
  - クロスブラウザ対応は不要
  - 誰に遠慮する必要もなく、使いたい技術を選んで使える
  - 多少不安定な技術でも構わない
- 残念ながら「銀の弾丸」は存在しない
  - 「これさえやっておけば」という効果的な対応方法は存在しない
  - 地道な努力、地道な対応あるのみ

今日の話

フロントエンドの比重が高まるなかで、最低限のJavaScriptのセキュリティ対策の話に限定

- JavaScriptに関するセキュリティ問題
  - オープンリダイレクタ
  - DOM-based XSS

その前に…

クロスサイトスクリプティング
強制ブラウズ
書式文字列攻撃
リモートファイルインクルード
SQLインジェクション
LDAPインジェクション
パストラバーサル
バッファオーバーフロー
CSRF
セッションハイジャック

そもそも「脆弱性」って何?

OSコマンドインジェクション
セッション固定攻撃
オープンリダイレクタ
DoS
HTTPレスポンス分割
メモリリーク
XPathインジェクション
HTTPヘッダインジェクション

そもそも「脆弱性」って何？

- 「脆弱性」という言葉を使ったことは？
- 「脆弱性」を見つけたことは？
- 「脆弱性」を説明できる人、挙手！

OWASP
Open Web Application Security Project

「脆弱性」の定義

- 経済産業省告示第235号

> “ソフトウエア等において、コンピュータウイルス、コンピュータ不正アクセス等の攻撃によりその機能や性能を損なう原因となり得る安全性上の問題箇所
>
> ウェブアプリケーションにあっては、ウェブサイト運営者がアクセス制御機能により保護すべき情報等に誰もがアクセスできるような、安全性が欠如している状態を含む”
>
> http://www.meti.go.jp/policy/netsecurity/downloadfiles/vulhandlingG.pdf

「脆弱性」の定義

- IPAによる定義

> 脆弱性とは、ソフトウエア製品やウェブアプリケーション等におけるセキュリティ上の問題箇所です。コンピュータ不正アクセスやコンピュータウイルス等により、この問題の箇所が攻撃されることで、そのソフトウエア製品やウェブアプリケーションの本来の機能や性能を損なう原因となり得るものをいいます。
>
> また、個人情報等が適切なアクセス制御の下に管理されていないなど、ウェブサイト運営者の不適切な運用により、ウェブアプリケーションのセキュリティが維持できなくなっている状態も含みます。
>
> http://www.ipa.go.jp/security/vuln/report/index.html

「脆弱性」の定義

- Microsoftによる定義

> “
>
> セキュリティの脆弱性とは、攻撃者が製品の完全性、可用性、または機密性を侵害する可能性のある製品の弱点です。
>
> ”
>
> http://technet.microsoft.com/ja-jp/library/gg983510.aspx

「脆弱性」の定義

- 脆弱性はただのバグ

脆弱性はバグの一種です。
一般的なバグは「できるはずのことができない」というものですが、脆弱性は「できないはずのことができる」というバグです。もっと言うと、「できてはいけないことができる」ということです

HASHコンサルティング
徳丸浩さん

OWASP
Open Web Application Security Project

脆弱性はただのバグ

- バグの少ないプログラム = 脆弱性も少ない
  - 脆弱性を減らすにはバグを減らせばいい
  - 「バグは少ないのに脆弱性が多い」「バグは多いのに脆弱性が少ない」という例はほとんどない
- まずはプログラムの品質をあげよう!

OWASP
Open Web Application Security Project

本題: JavaScriptのセキュリティ

JavaScriptに関するセキュリティ問題

- ブラウザ上で発生する問題 - 受動的攻撃
  - 攻撃者のしかけた罠をトリガに、ユーザーのブラウザ上で問題が発生する

JavaScriptに関するセキュリティ問題

- 主なセキュリティ上の問題
  - JavaScriptによるオープンリダイレクタ
  - DOM-based XSS
  - XHRを用いたCSRF
  - Ajaxデータの漏えい
  - 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に関するセキュリティ問題

- 主なセキュリティ上の問題
  - JavaScriptによるオープンリダイレクタ
  - DOM-based XSS
  - XHRを用いたCSRF
  - Ajaxデータの漏えい
  - 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

今日話す内容

OWASP
Open Web Application Security Project

JavaScriptに関するセキュリティ問題

- 主なセキュリティ上の問題
  - JavaScriptによるオープンリダイレクタ
  - DOM-based XSS

  今日話す内容

  - **XHRを用いたCSRF**
  - **Ajaxデータの漏えい**
  - **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**

JPCERT/CC「HTML5を利用したWebアプリケーションのセキュリティ問題に関する調査報告書」を参照
http://www.jpcert.or.jp/research/html5.html

JSによるオープンリダイレクタ

JSによるオープンリダイレクタ

- JavaScriptによるリダイレクト(ページ移動)

```
location.href = url;
location.assign( url );
```

- 遷移先ページが攻撃者によってコントロール可能な場合、オープンリダイレクタとなる

```
// bad code. URL中の#より後ろを次のURLとして表示する。
// http://example.jp/#next など。
var url = "/" + location.hash.substr(1); //「/next」に移動
location.href = url;
```

```
攻撃者はhttp://example.jp/#/evil.utf-8.jp/などにユーザーを誘導
location.href = "//evil.utf-8.jp/"
```

JSによるオープンリダイレクタ

- オープンリダイレクタ
  - 任意のサイトにリダイレクトされてしまう
  - それ自体は実質的に大きな問題があるわけではない
- 間接的な影響
  - 元サイト内のコンテンツのように見せかけてユーザーを誘導
  - フィッシングサイトへの誘導
  - ドメインを信頼して訪問したユーザーを裏切ることにもなる

JSによるオープンリダイレクタ

- オープンリダイレクタとならないために
  - 遷移先を固定リストで持つ

```
// URL中の#より後ろを次のURLとして表示する。
// http://example.jp/#next など。
const pages = { next:"/next", foo:"/foo", bar:"/bar" };
const hash = location.hash.substr(1);
let url = pages[ hash ];
if (url === undefined || !pages.hasOwnProperty[hash])
    url = "/notfound";
location.href = url;
```

  - 遷移先URLとして自サイトのドメイン名を先頭に付与する

```
const url = location.origin + "/" + location.hash.substr(1);
location.href = url;
```

JSによるオープンリダイレクタ

- オープンリダイレクタとならないために(続き)
  - Chrome,FirefoxではURLオブジェクトを利用してオリジンを確認

```
// 相対URL等を絶対URLのURLオブジェクトに変換
const url = new URL( text, location.href );
if (url.origin === "http://example.jp") {
   location.href = url;
}
```

  - IEではa要素を使って同種のことが実現可能
    コードは割愛
    http://d.hatena.ne.jp/hasegawayosuke/20151204/p1

DOM-based XSS

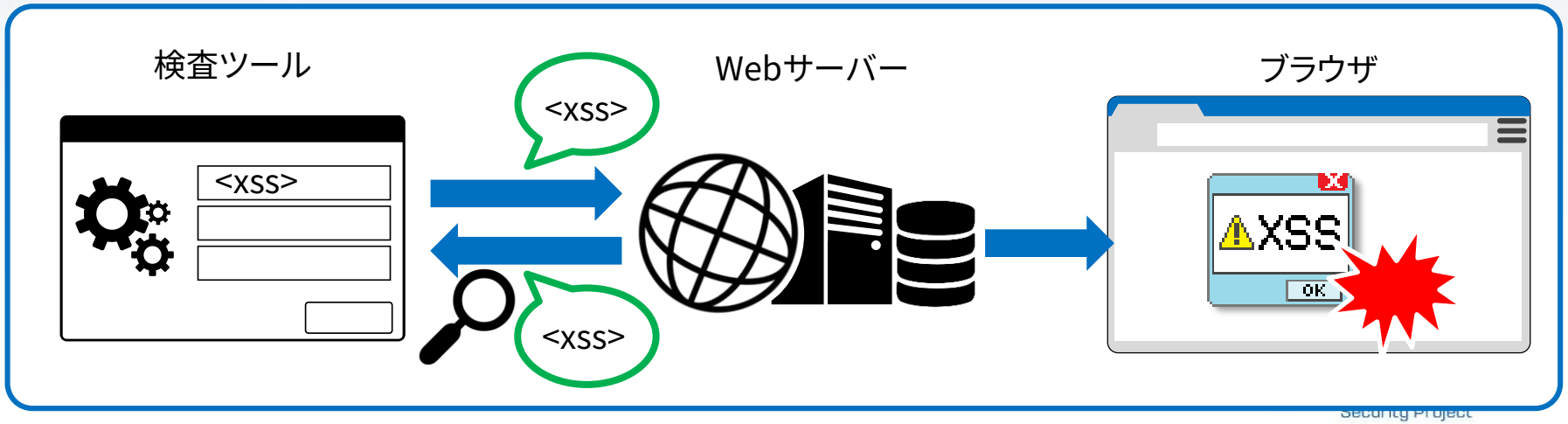
DOM-based XSS

- JavaScriptが引き起こすXSS
  - サーバ上でのHTML生成には問題なし
- JavaScriptによるレンダリング時にブラウザ上で問題が発生する

```
// bad code
// 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

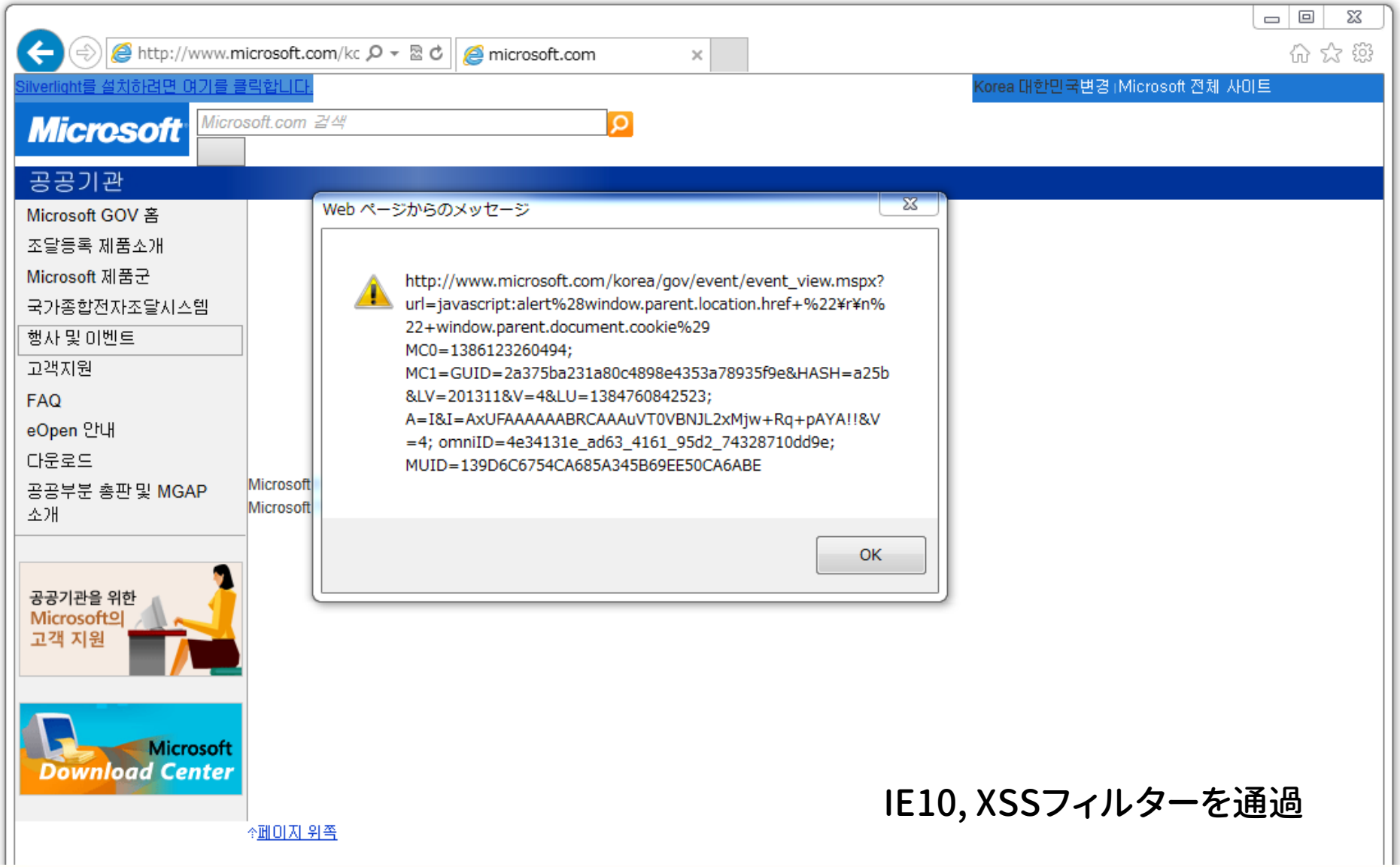
- 静的コンテンツのみでもXSSする可能性
  - 動的にHTMLを生成する「Webアプリケーション」ではなく、*.htmlしか提供してなくてもXSSのある可能性がある

```
<html>
<script>
    document.write( location.hash.substring(1) );
</script>
</html>
```

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

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

- 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 原因と対策

- 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 原因と対策

- 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 原因と対策

- 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>
```

シンク

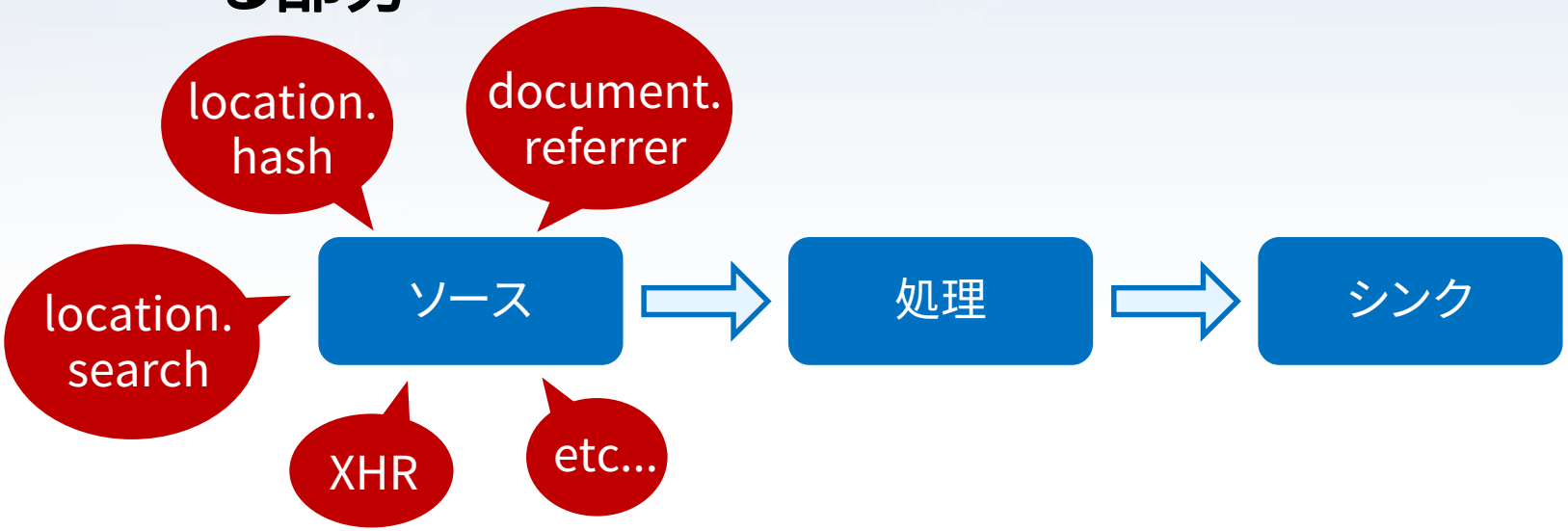
ソース

DOM-based XSS 原因と対策

- 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

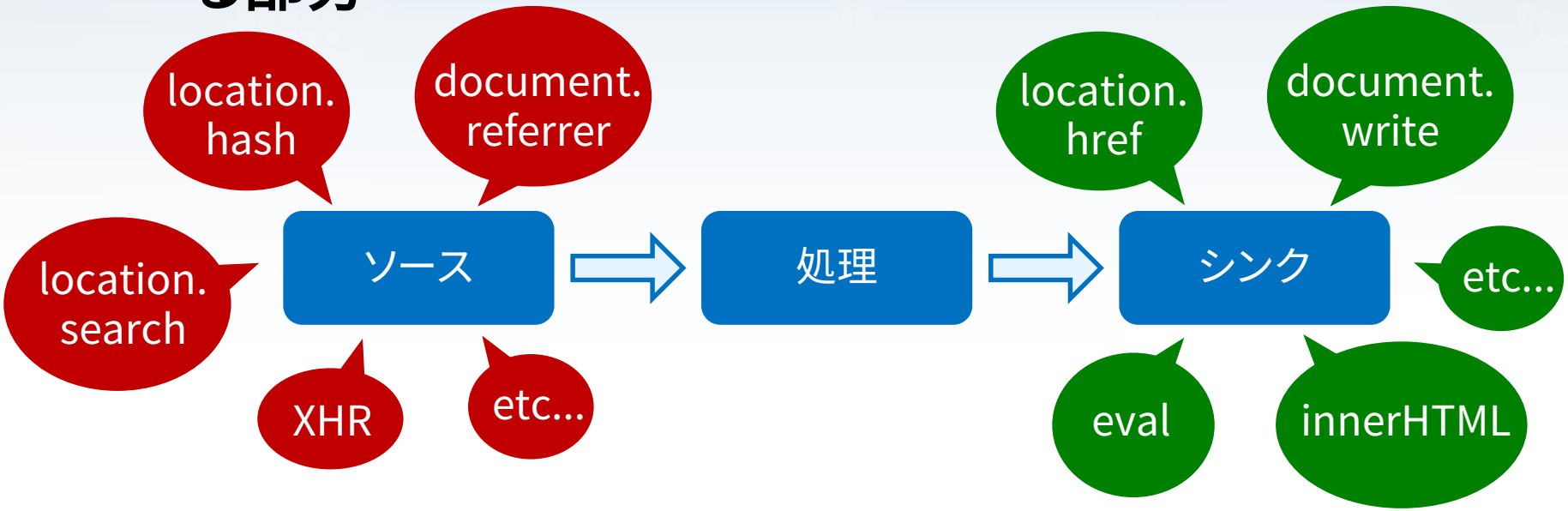
DOM-based XSS 原因と対策

- 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 原因と対策

- 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 原因と対策

- 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 原因と対策

- 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 原因と対策

- 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 ) );
```

```
const text = document.createTextNode(
    location.hash.substr( 1 )
);
document.body.appendChild( text );
```

DOM-based XSS 原因と対策

- 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 "">

```
const text = "....";  //変数textは攻撃者がコントロール可能
const 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 );
```

DOM-based XSS 原因と対策

- 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 原因と対策

- とはいえinnerHTMLを使わざるを得ないケースもある
  - サーバからHTML断片をXHRで取得しHTML内に挿入する等

```
// bad code
// http://example.jp/#news のようなURLでアクセスすると
// /news の内容をXHRで取得してHTMLとして挿入
var url = "/" + location.hash.substr(1);
var xhr = new XMLHttpRequest();
xhr.open( "GET", url, true );
xhr.onload = function(){
    document.getElementById( "news-list" ).innerHTML =
        xhr.responseText
}
xhr.send( null );
```

XMLHttpRequest経由でのXSS

- 攻撃者が
  http://example.jp/#/attacker.example.com/
  のようなURLに誘導することで本来とは異なるサーバからHTML断片がロードされてしまう

```
// bad code
// http://example.jp/#news のようなURLでアクセスすると
// /news の内容をXHRで取得してHTMLとして挿入
var url = "/" + location.hash.substr(1);
var xhr = new XMLHttpRequest();
xhr.open( "GET", url, true );
xhr.onload = function(){
    document.getElementById( "news-list" ).innerHTML =
        xhr.responseText
}
xhr.send( null );
```

url = "//attacker.example.com/"

XMLHttpRequest経由でのXSS

- サーバ側で生成済みのHTML断片をブラウザ内に流し込みたい
- HTML断片なのでテキストノードとして扱えない innerHTMLを使うしかない
- 対策：自身のサーバ以外とは接続できないようURLを限定する
  - オープンリダイレクタ対策と同様
    - URLを固定リストで持つ
    - 自サイトのドメイン名を先頭に付与する
    - URLオブジェクトを使って絶対URLを生成

XMLHttpRequest経由でのXSS

- 対策 - 自身のサーバ以外とは接続できないようにする
  - URLを固定リストで持つ

```
// URL中の#より後ろを次のURLとして表示する。
// http://example.jp/#next など。
const pages = { news:"/news", info:"/info", foo:"/foo" };
const url = pages[ location.hash.substr(1) ];
if( url ){
    xhr = new XMLHttpRequest();
    xhr.open( "GET", url, true );
    xhr.onload = function(){ elm.innerHTML = xhr.responseText; }
    xhr.send( null );
}
```

XMLHttpRequest経由でのXSS

- 対策 - 自身のサーバ以外とは接続できないようにする
  - URL先頭に自身のホスト名を付与する方法はオープンリダイレクタが存在していると攻撃者に回避されてしまうのであまり勧められない

```
// あまりよくないコード
const url = location.origin + "/" + location.hash.substr(1);
if( url ){
    xhr = new XMLHttpRequest();
    xhr.open( "GET", url, true );
    ...
```

http://example.jp/redir?url=http://utf-8.jp/ のようなオープンリダイレクタが存在していると
http://example.jp/#redir?url=http://utf-8.jp/ のような指定で他サイトからXHRで取得してしまう

DOM-based XSS 原因と対策

- 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 原因と対策

- URLの生成時はhttp(s)に限定

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

リンク

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

DOM-based XSS 原因と対策

- 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
```

```
if( url.match( /^https?:¥/¥// ) ){
    locatoin.href = url;
}
```

DOM-based XSS 原因と対策

- Chrome,FirefoxであればURLオブジェクトも利用可能

```
const url = new URL( text, location.href );
if( url.protocol.match( /^https?/ ) ){
    // http or https
}
```

- IEではa要素を使って同種のことが実現可能
  - コードは割愛

    http://gihyo.jp/dev/serial/01/javascript-security/0005

DOM-based XSS 原因と対策

- 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 原因と対策

- 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 原因と対策

- 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

まとめ

- ブラウザ上、JavaScript上の脆弱性が増加
  - JSコード量、処理量の増加
- 脆弱性はただのバグ
  - バグを減らす = 脆弱性が減る
- 攻撃者有利な状況
  - 脆弱性を作りこまない必要性
- 参考
  - JavaScriptセキュリティの基礎知識
    http://gihyo.jp/dev/serial/01/javascript-security

質問?

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

@hasegawayosuke

http://utf-8.jp/