HTML5時代のWebセキュリティ

May 10 2014

Yosuke HASEGAWA

#sitw23

自己紹介

はせがわようすけ

- ネットエージェント株式会社
- 株式会社セキュアスカイ・テクノロジー 技術顧問
- セキュリティ・キャンプ Webセキュリティクラス
- OWASP Kansai Chapter Leader
- OWASP Japan board member
- http://utf-8.jp/

@hasegawayosuke

utf-8.jp/public/jjencode.html

jjencode demo

Enter any JavaScript source:

```
alert("Hello, JavaScript" )
```

global variable name used by jjencode : $ palindrome

jjencode

```
$=~[];$={___:++$,$$$$:(![]+"")[$],__$:++$,$_$_:(![]+"")[$],_$_:++$,$_$$:({}+"")[$],$$_$:($[$]+"")
[$],_$$:++$,$$$_:(!""+"")[$],$__:++$,$_$:++$,$$__:({}+"")[$],$$_:++$,$$$:++$,$___:++$,$__$:++$};$.$_=
($.$_=$+"")[$.$_$]+($._$=$.$_[$.__$])+($.$$=($.$+"")[$.__$])+((!$)+"")[$._$$]+($.__=$.$_[$.$$_])+($.$=
(!""+"")[$.__$])+($._=(!""+"")[$._$_])+$.$_[$.$_$]+$.__+$._$+$.$;$.$$=$.$+(!""+"")
[$._$$]+$.__+$._+$.$+$.$$;$.$=($.___)[$.$_][$.$_];$.$($.$($.$$+"¥""+$.$_$_+(![]+"")
[$._$_]+$.$$$_+"¥¥"+$.__$+$.$$_+$._$_+$.__+"(¥¥¥"¥¥"+$.__$+$.__$+$.___+$.$$$_+(![]+"")[$._$_]+(![]+"")
[$._$_]+$._$+",¥¥"+$.$__+$.___+"¥¥"+$.__$+$._$_+$.$_$_+"¥¥"+$.__$+$.$$_+$.$$_+$.$_$_+"¥¥"+$.__$+$
._$_+$._$$+$.$$__+"¥¥"+$.__$+$.$$_+$._$_+"¥¥"+$.__$+$.$_$+$.__$+"¥¥"+$.__$+$.$$_+$.___+$.__+"¥¥¥"¥¥"+$.
$__+$.___+")"+"¥"")())();
```

798 letters

eval [Permalink]

utf-8.jp/public/aaencode.html

aaencode demo

aaencode - Encode any JavaScript program to Japanese style emoticons (^_^)

Enter JavaScript source:

```
alert("Hello, JavaScript")
```

aaencode

```
ﾟωﾟﾉ= / ｀ｍ´) ﾉ ~┻━┻   //*´∇｀*/ ['_']; o=(ﾟｰﾟ)  =_=3; c=(ﾟΘﾟ) =(ﾟｰﾟ)-(ﾟｰﾟ); (ﾟДﾟ) =(ﾟΘﾟ)=
(o^_^o)/ (o^_^o);(ﾟДﾟ)={ﾟΘﾟ: '_' ,ﾟωﾟﾉ : ((ﾟωﾟﾉ==3) +'_') [ﾟΘﾟ] ,ﾟｰﾟﾉ :(ﾟωﾟﾉ+ '_')[o^_^o -(ﾟ
Θﾟ)] ,ﾟДﾟﾉ:((ﾟｰﾟ==3) +'_')[ﾟｰﾟ] }; (ﾟДﾟ) [ﾟΘﾟ] =((ﾟωﾟﾉ==3) +'_') [c^_^o];(ﾟДﾟ) ['c'] = ((ﾟ
Дﾟ)+'_') [ (ﾟｰﾟ)+(ﾟｰﾟ)-(ﾟΘﾟ) ];(ﾟДﾟ) ['o'] = ((ﾟДﾟ)+'_') [ﾟΘﾟ];(ﾟoﾟ)=(ﾟДﾟ) ['c']+(ﾟДﾟ) ['o']+
(ﾟωﾟﾉ +'_')[ﾟΘﾟ]+ ((ﾟωﾟﾉ==3) +'_') [ﾟｰﾟ] + ((ﾟДﾟ) +'_') [(ﾟｰﾟ)+(ﾟｰﾟ)]+ ((ﾟｰﾟ==3) +'_') [ﾟΘﾟ]+
((ﾟｰﾟ==3) +'_') [(ﾟｰﾟ) - (ﾟΘﾟ)]+(ﾟДﾟ) ['c']+((ﾟДﾟ)+'_') [(ﾟｰﾟ)+(ﾟｰﾟ)]+ (ﾟДﾟ) ['o']+((ﾟｰﾟ==3)
+'_') [ﾟΘﾟ];(ﾟДﾟ) ['_'] =(o^_^o) [ﾟoﾟ] [ﾟoﾟ];(ﾟεﾟ)=((ﾟｰﾟ==3) +'_') [ﾟΘﾟ]+ (ﾟДﾟ) .ﾟДﾟﾉ+((ﾟ
Дﾟ)+'_') [(ﾟｰﾟ) + (ﾟｰﾟ)]+((ﾟｰﾟ==3) +'_') [o^_^o -ﾟΘﾟ]+((ﾟｰﾟ==3) +'_') [ﾟΘﾟ]+ (ﾟωﾟﾉ +'_') [ﾟΘﾟ];
(ﾟｰﾟ)+=(ﾟΘﾟ); (ﾟДﾟ)[ﾟεﾟ]='¥¥'; (ﾟДﾟ).ﾟΘﾟﾉ=(ﾟДﾟ+ ﾟｰﾟ)[o^_^o -(ﾟΘﾟ)];(oﾟｰﾟo)=(ﾟωﾟﾉ +'_')
[c^_^o];(ﾟДﾟ) [ﾟoﾟ]='¥"';(ﾟДﾟ) ['_'] ( (ﾟДﾟ) ['_'] (ﾟεﾟ+(ﾟДﾟ)[ﾟoﾟ]+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ (ﾟｰﾟ)+
(ﾟΘﾟ)+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ ((ﾟｰﾟ) + (ﾟΘﾟ))+ (ﾟｰﾟ)+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ (ﾟｰﾟ)+ ((ﾟｰﾟ) + (ﾟΘﾟ))+
(ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ ((o^_^o) +(o^_^o))+ ((o^_^o) - (ﾟΘﾟ))+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ ((o^_^o) +
(o^_^o))+ (ﾟｰﾟ)+ (ﾟДﾟ)[ﾟεﾟ]+((ﾟｰﾟ) + (ﾟΘﾟ))+ (c^_^o)+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟｰﾟ)+ ((o^_^o) - (ﾟΘﾟ))+ (ﾟ
Дﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ (ﾟΘﾟ)+ (c^_^o)+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ (ﾟｰﾟ)+ ((ﾟｰﾟ) + (ﾟΘﾟ))+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟ
Θﾟ)+ ((ﾟｰﾟ) + (ﾟΘﾟ))+ (ﾟｰﾟ)+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ ((ﾟｰﾟ) + (ﾟΘﾟ))+ (ﾟｰﾟ)+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+
((ﾟｰﾟ) + (ﾟΘﾟ))+ ((ﾟｰﾟ) + (o^_^o))+ (ﾟДﾟ)[ﾟεﾟ]+((ﾟｰﾟ) + (ﾟΘﾟ))+ (ﾟｰﾟ)+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟｰﾟ)+
(c^_^o)+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ (ﾟΘﾟ)+ ((o^_^o) - (ﾟΘﾟ))+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ (ﾟｰﾟ)+ (ﾟΘﾟ)+ (ﾟДﾟ)
```

[eval] [Permalink]

[utf-8.jp/]

宣伝

セキュリティ・キャンプ

未来を守るカギを見つけろ。

セキュリティ・キャンプの紹介

セキュリティ・キャンプ全国大会

2014.08.12-2014.08.16 開催決定！

セキュリティ・キャンプとは

中央大会/全国大会

- 22歳以下の学生を対象とした合宿型講習会
- 業界の第一線で活躍するセキュリティ技術者が講師
- 2004年から2013年まで10回開催、438名の学生が参加
- 主催：セキュリティ・キャンプ実施協議会、独立行政法人情報処理推進機構(IPA)

セキュリティ・キャンプとは

セキュリティ・キャンプ実施協議会

- 2012年2月設立、30の団体・企業が参加(2014年2月時点)

(ISC)2Japan	サイボウズ株式会社	株式会社日立製作所
株式会社イーセクター	株式会社セキュアソフト	富士通エフ・アイ・ピー株式会社
伊藤忠テクノソリューションズ株式会社	ソニー株式会社	フューチャーアーキテクト株式会社
株式会社インテリジェントウェイブ	トレンドマイクロ株式会社	株式会社三菱総合研究所
SCSK株式会社	一般財団法人日本情報経済社会推進協会	三菱電機株式会社
エヌ・ティ・ティ・コミュニケーションズ株式会社	一般社団法人日本情報システム・ユーザー協会	ヤフー株式会社
株式会社エヌ・ティ・ティ・データ	日本電気株式会社	LINE株式会社
株式会社オービックビジネスコンサルタント	日本電信電話株式会社	楽天株式会社
グーグル株式会社	株式会社野村総合研究所	株式会社ラック
株式会社サイバーエージェント	パナソニック・アドバンストテクノロジー株式会社	株式会社リクルートテクノロジーズ

参加者数

開催年	参加者数	講師数	備考
2004	30人(男性28、女性2)	8人	U-20
2005	30人(男性26、女性4)	11人	5泊6日
2006	36人(男性32、女性4)	14人	U-22化
2007	35人(男性32、女性3)	14人	
2008	47人(セ:29、プ:18)	24人	+プログラミングコース
2009	61人(セ:31、プ:30)	29人	
2010	59人(セ:30、プ:29)	37人	CTF
2011	60人(セ:45、プ:15)	28人	大阪開催
2012	40人(男性39、女性1)	20人	プログラミングコース廃止
2013	41人(男性36、女性5)	18人	

クラス分け

- ソフトウェア・セキュリティ・クラス
- Web・セキュリティ・クラス
- ネットワーク・セキュリティ・クラス
- セキュアなシステムを作ろうクラス

出てきた人たち

- キャンプ講師になった
- キャンプで出会って結婚した
- 自分の言語作った
- 目でgrepした
- キャンプで出会った仲間と起業した
- 未踏やった
- 日本学生科学賞、科学技術振興機構賞取った
- 最近若いのにすごい人や変な人多くて楽しいヽ(´ー｀)ノ

目指したいもの

- 世の中の解決できていない課題を認識させたい→ビジネスチャンス
- 物作り、コード書きをしながらのセキュリティ
- アドホックやバッドノウハウではないセキュリティ
- ブラウザやOSなど、あんまり誰も見てないところのセキュリティ
- コミュニティーパワーになる人を増やしたい
- 変な人の「変さ」を増幅（笑）したい

そろそろ応募開始！

応募資格

- 日本国内に居住する、2014年3月31日時点において22歳以下の学生・生徒

セキュリティキャンプ　検索

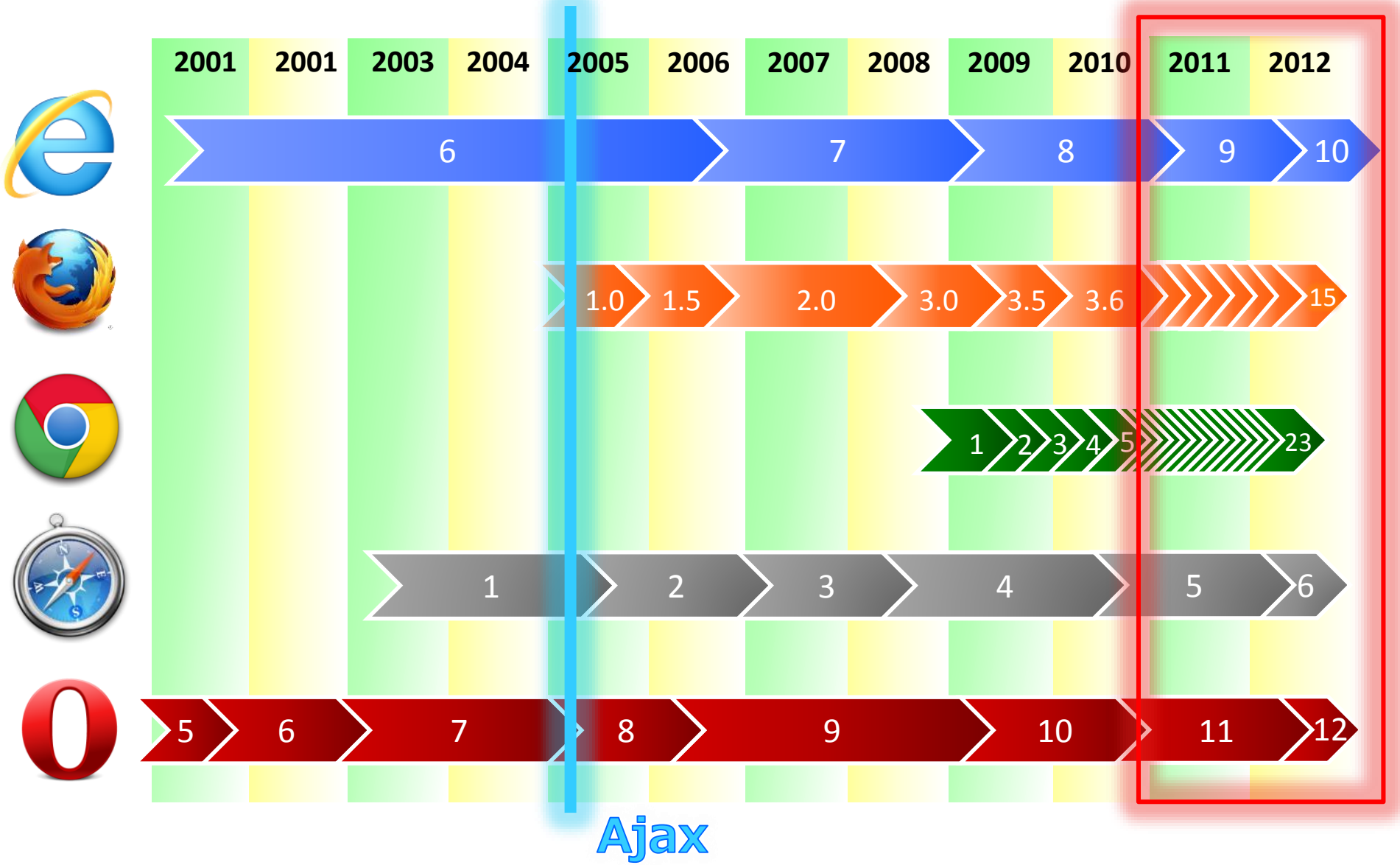
- 大人の人はスポンサーに…

本題
HTML5時代のWebセキュリティ

HTML5時代のWebアプリ

- 次々とリリースされるブラウザ
  - 多数の新しい要素と属性
    - canvas, video, audio, input…
  - 多数の新しいAPI
    - Web Sockets, Web Storage, XHR Lv.2…
  - 最適化されたJavaScriptエンジン
  - 高速化された描画エンジン
- どのブラウザにどの機能が実装されているのか把握できない

次々リリースされるブラウザ



HTML5の新機能

- マルチメディアのサポート
  <video> <audio> <canvas>...
- 文書構造を表す要素
  <section> <header> <footer> ...
- フォームの拡張
  <input type="email"> ...
- JavaScript API
  Web Workers, WebSocket, File...
- その他…

HTML5時代のWebアプリ

- HTML5時代のブラウザ
  - 高速化、高機能化
- 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
  - サーバ側で実行されていた処理がブラウザのJavaScript上へ
- 攻撃もクライアントサイドへシフト
  - JavaScript上の問題点の増加
  - XSSやCSRFなどの比重が増加

Webの技術 楽しいですよね！


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
Webの技術 楽しいですよね！
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

HTML5を使った攻撃

- 攻撃側こそ新しいWebの技術をもっとも活用できる
  - クロスブラウザ対応不要！
  - 誰に遠慮すること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かまわない！

HTML5で増加する脅威

- XSS
  - HTML5の新要素によるXSS
  - JSコード量の増加 – DOM Based XSS
  - AjaxデータによるXSS
- CSRF
  - XMLHttpRequestで攻撃者有利
- オープンリダイレクタ
  - JavaScriptによるリダイレクトの増加
- その他
  - Ajaxデータからの情報漏えい
  - APIの使い方の問題
    - Web Storage、WebWorkers、XDM…

HTML5で増加する脅威

- 攻撃もクライアントサイドへシフト
- JavaScriptを通じた攻撃の比重が増加
- XSSのリスクも増加

> 多くの点から見て、XSS 脆弱性の危険性はバッファ オーバーフローに匹敵します。

セキュリティに関するブリーフィング：Web に対する SDL の適用
http://msdn.microsoft.com/ja-jp/magazine/cc794277.aspx

5分でわかる対策

HTML5 調査報告 from JPCERT/CC

@IT連載 HTML5時代の「新しいセキュリティ・エチケット」

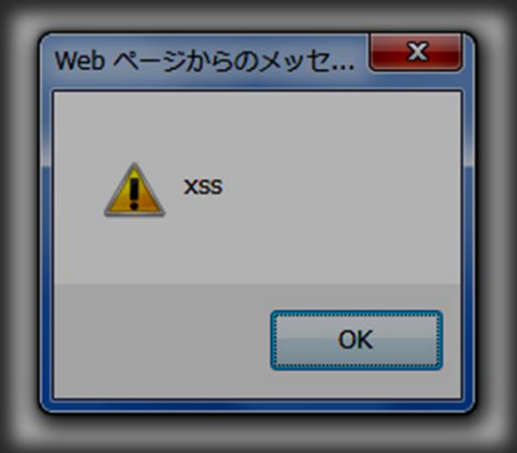
- HTML5時代の「新しいセキュリティ・エチケット」
  - (1) 重要！ まずは「オリジン」を理解しよう
    http://www.atmarkit.co.jp/ait/articles/1311/26/news007.html
  - (2) 単純ではない、最新「クロスサイトスクリプティング」事情
    http://www.atmarkit.co.jp/ait/articles/1312/17/news010.html
  - (3) 知っていれば恐くない、XMLHttpRequestによるXSSへの対応方法
    http://www.atmarkit.co.jp/ait/articles/1403/24/news005.html

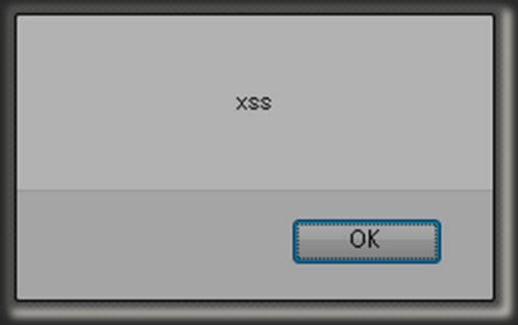
読みましょう！
以上！

HTML5時代の脆弱性

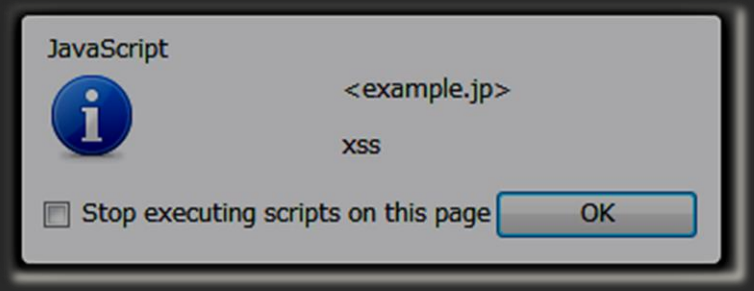
HTML5で増加する脅威

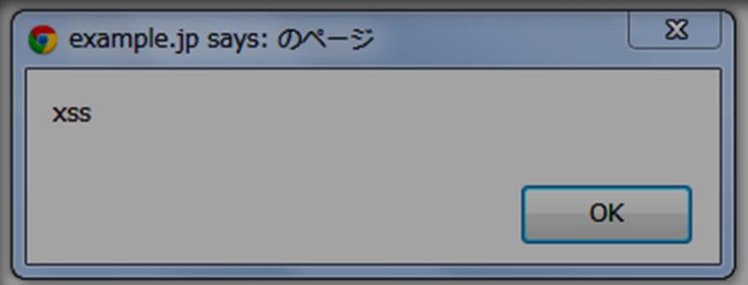
- XSS
  - HTML5の新要素によるXSS
  - JSコード量の増加 – DOM Based XSS
  - AjaxデータによるXSS
- CSRF
  - XMLHttpRequestで攻撃者有利
- オープンリダイレクタ
  - JavaScriptによるリダイレクトの増加
- その他
  - Ajaxデータからの情報漏えい
  - APIの使い方の問題
    - Web Storage、WebWorkers、XDM…





そもそもXSSって?
→簡単におさらい

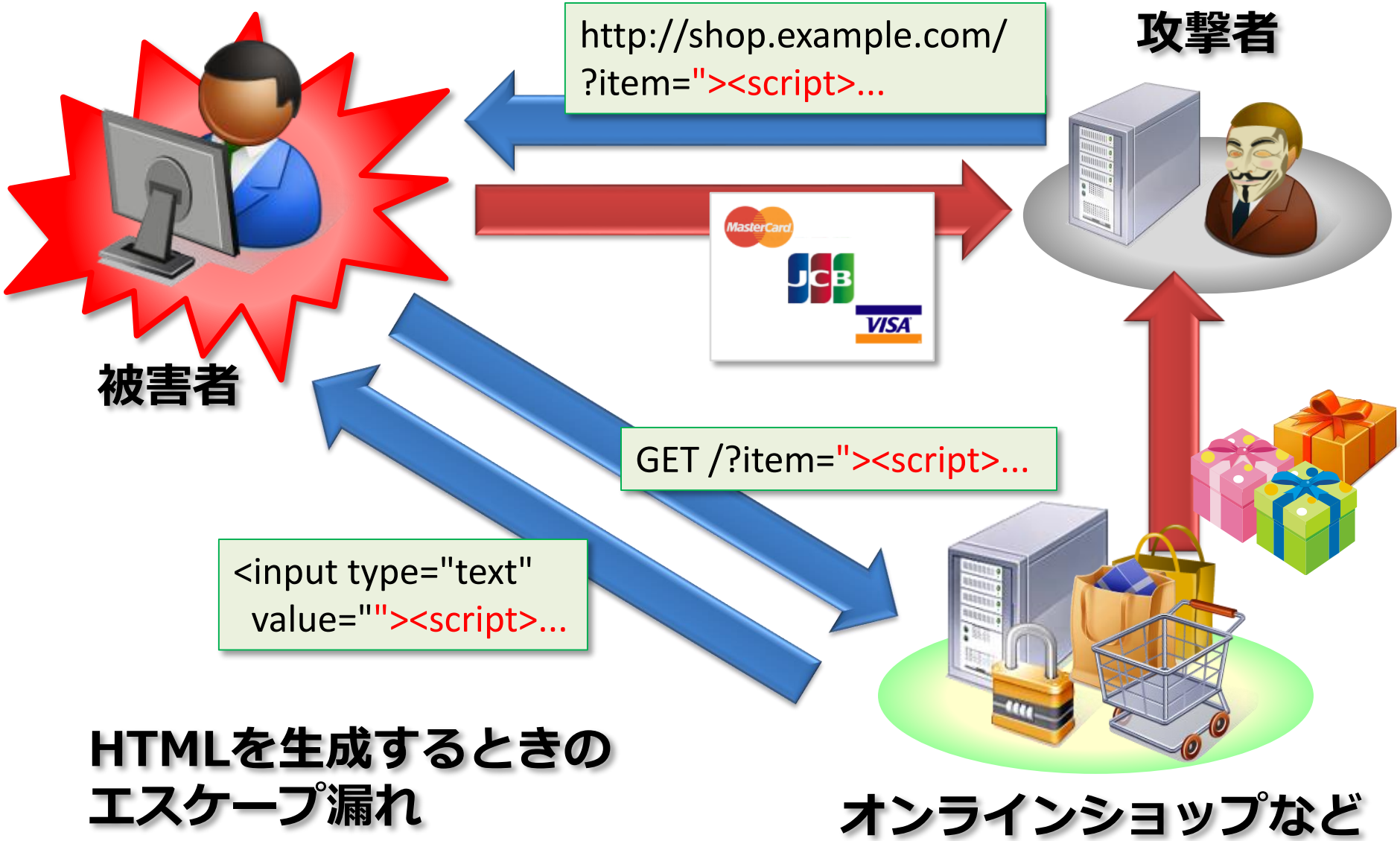




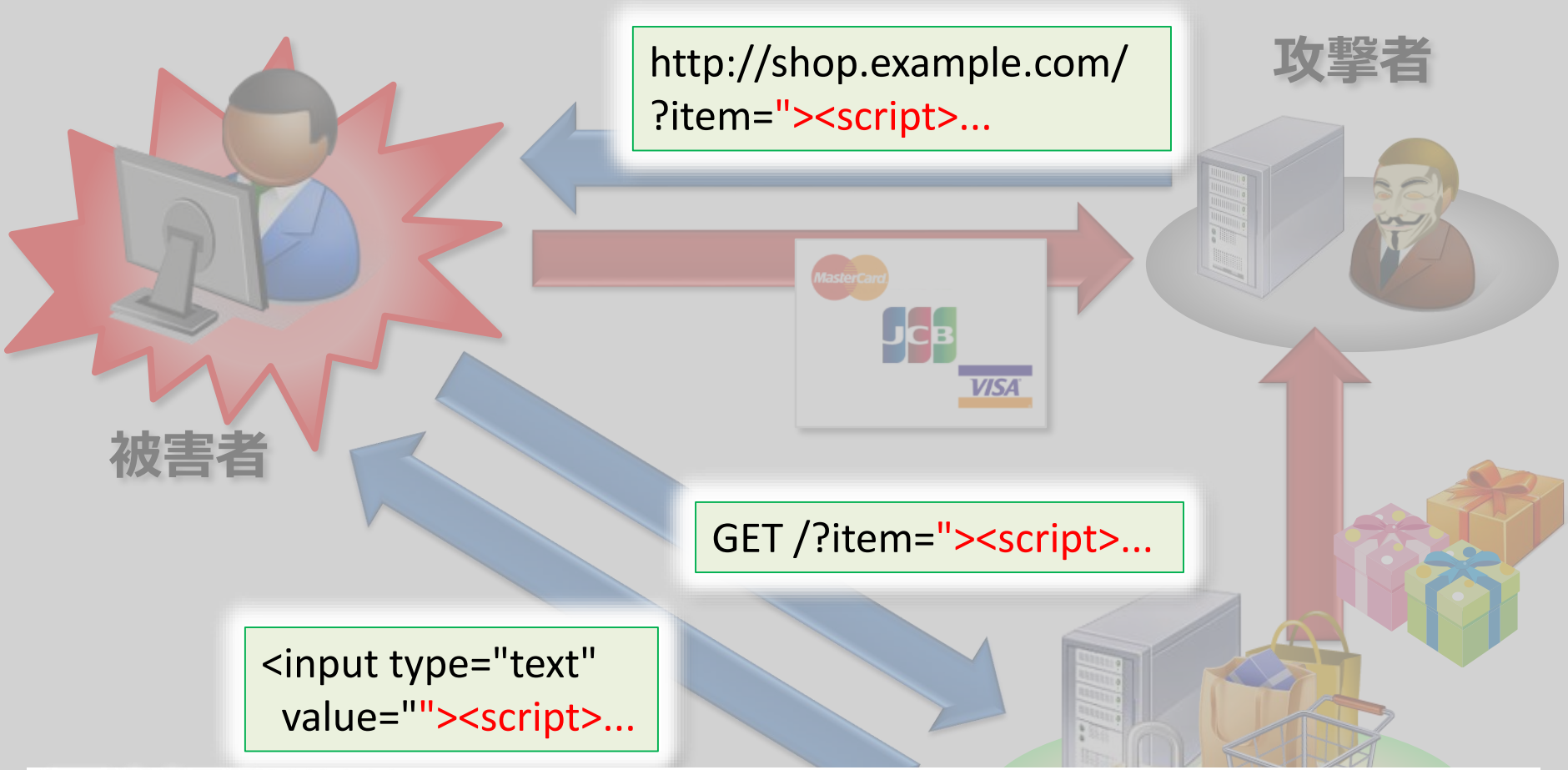
XSSおさらい

- 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

XSSおさらい


http://shop.example.com/
?item="><script>...
攻撃者
被害者
GET /?item="><script>...
<input type="text"
value=""><script>...
HTMLを生成するときの
エスケープ漏れ
オンラインショップなど

XSSおさらい



反射型XSS

ユーザの送信内容をそのまま表示する

XSSおさらい

被害者

攻撃者

Subject: test mail

<html><script>...

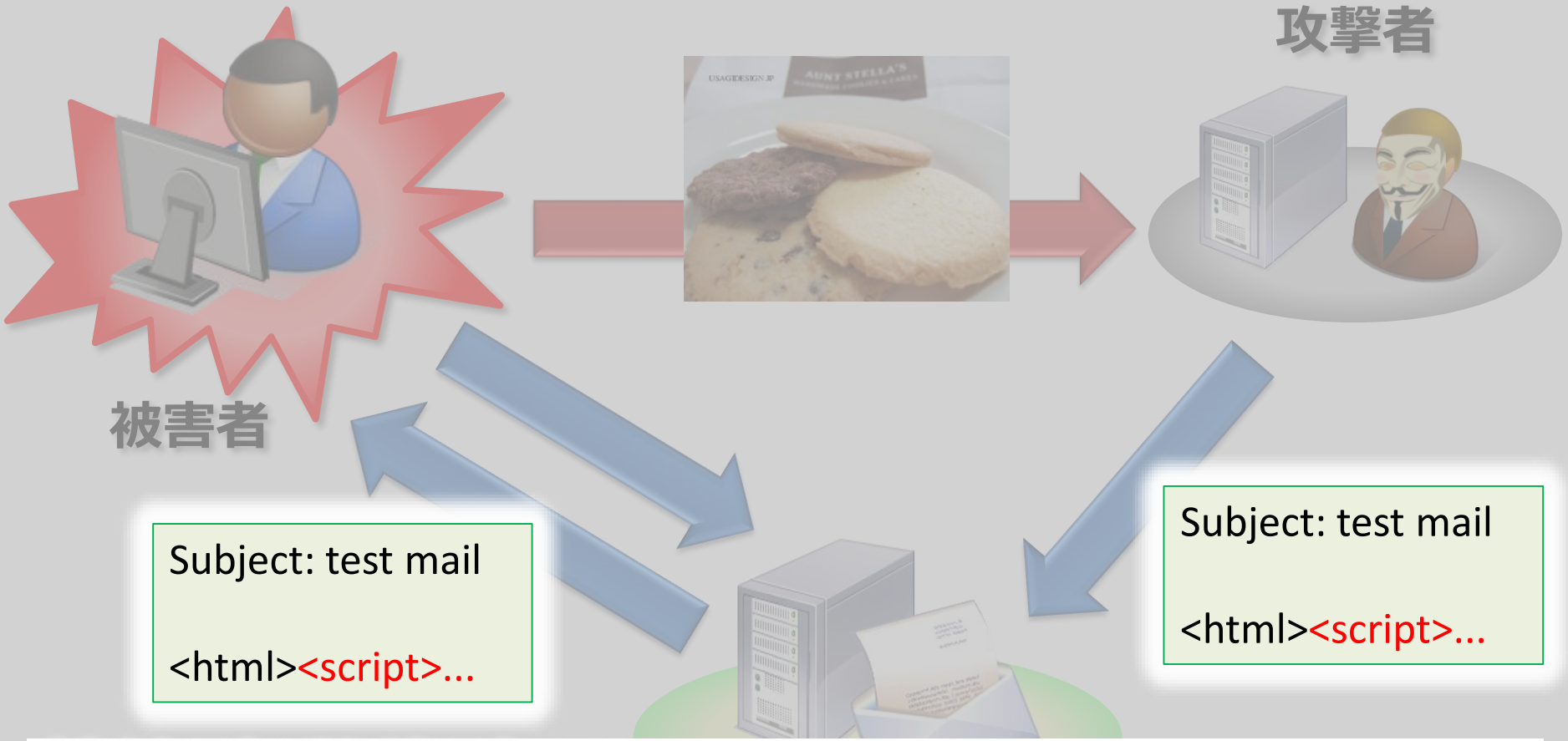
Subject: test mail

<html><script>...

HTMLを生成するときの
エスケープ漏れ

Webメールなど

XSSおさらい



持続型/蓄積型XSS

攻撃者のスクリプトはサーバ内で保持されている

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

大原則

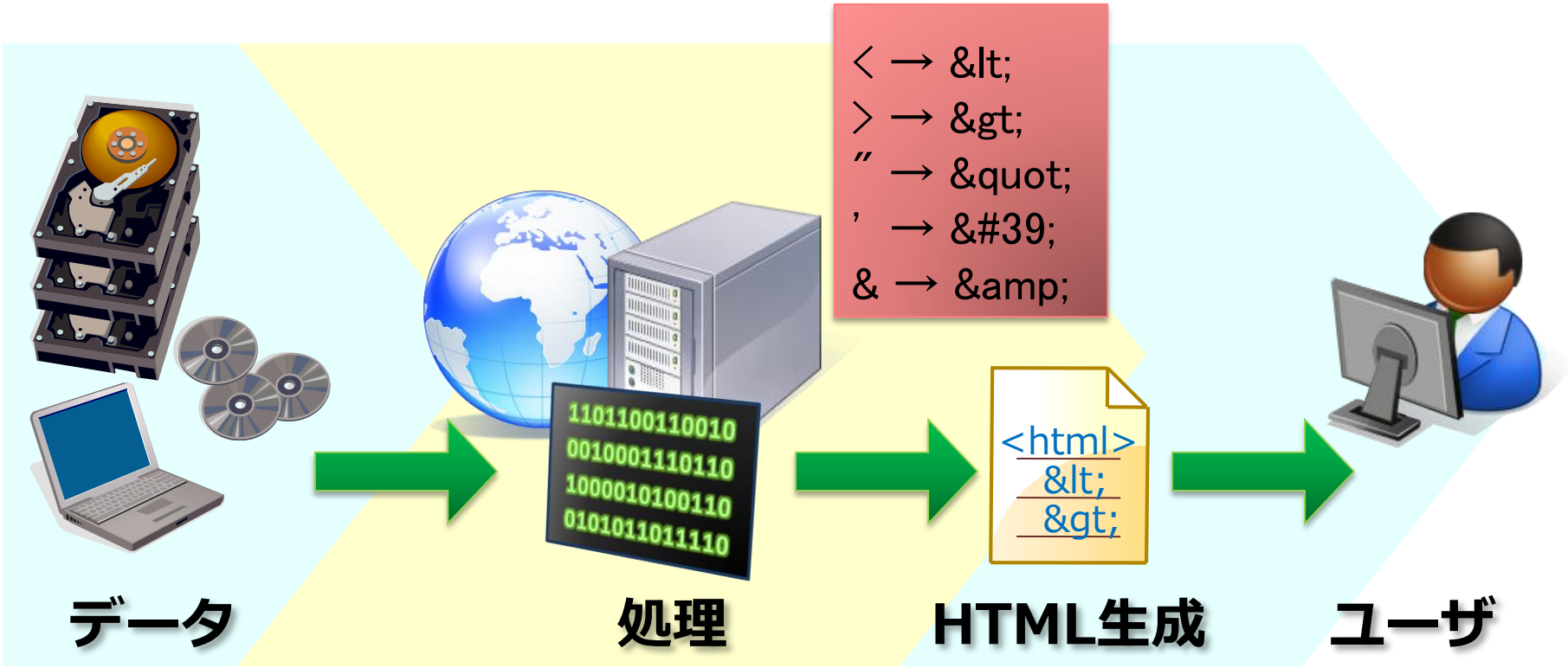
HTMLを生成する時点で
エスケープ

って何だっけ?

→ おさらい

XSSおさらい

- HTMLを生成する時点でエスケープ!



エスケープの例外

- href、src等のURLの動的生成

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

  - http以外のスキームに注意

- JavaScript内の動的生成

```
<script>
var s="</script><script>alert(1)//";
</script>
```

  - JS内の文字列リテラルはHTMLとは異なるエスケープ

XSSおさらい

- HTMLを生成する時点でエスケープ
  - コンテキスト(文脈)に応じたエスケープ
  - コンテキストが入れ子になっているときはエスケープも入れ子に。

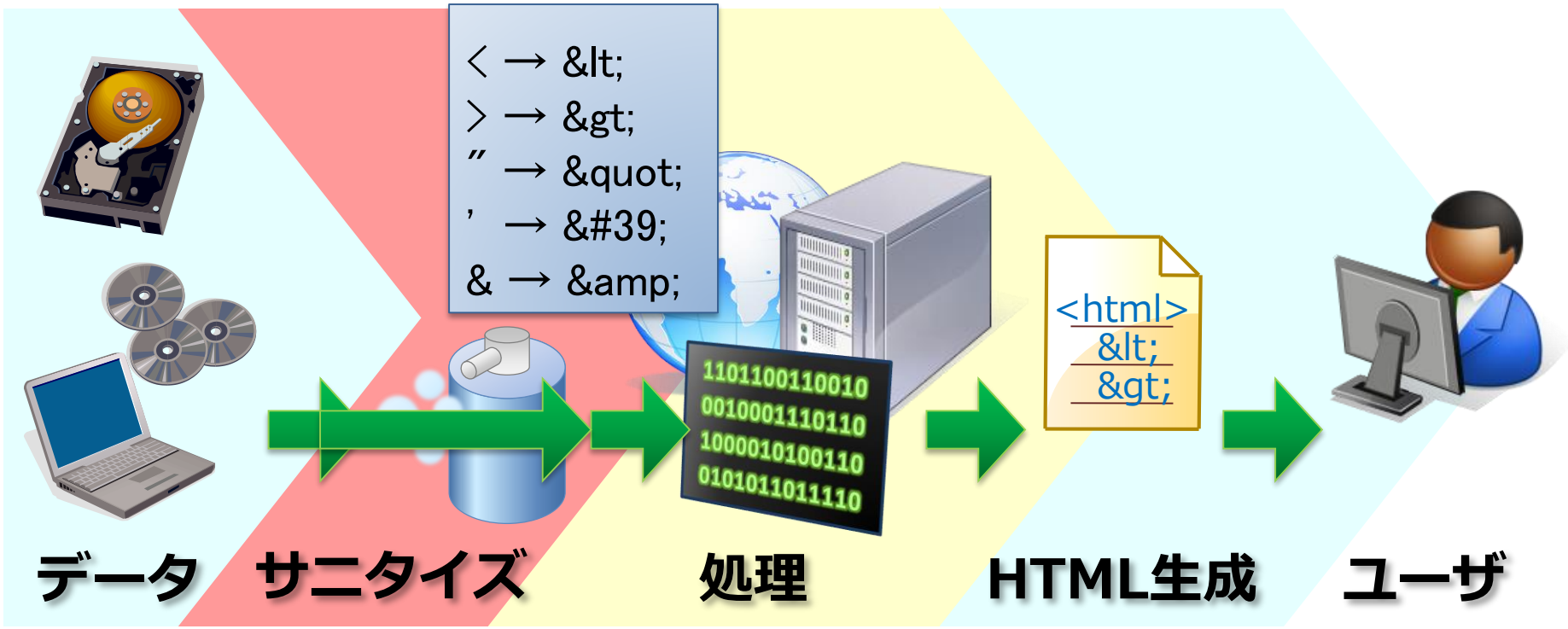
```
<div onclick="foo('¥u0022><script>…');">
```

- URLの動的生成
  - href、src等はhttp,https限定とする

やってはいけない
XSS対策

やってはいけないXSS対策

❖入力時のサニタイズ



やってはいけない：入力時のサニタイズ

- 入力時点でのデータの加工はプログラム規模が大きくなると破綻する
- 「サニタイズ」という語が本来の意味を失って意味不明になっている
  - 続きはWebで

XSSの種類

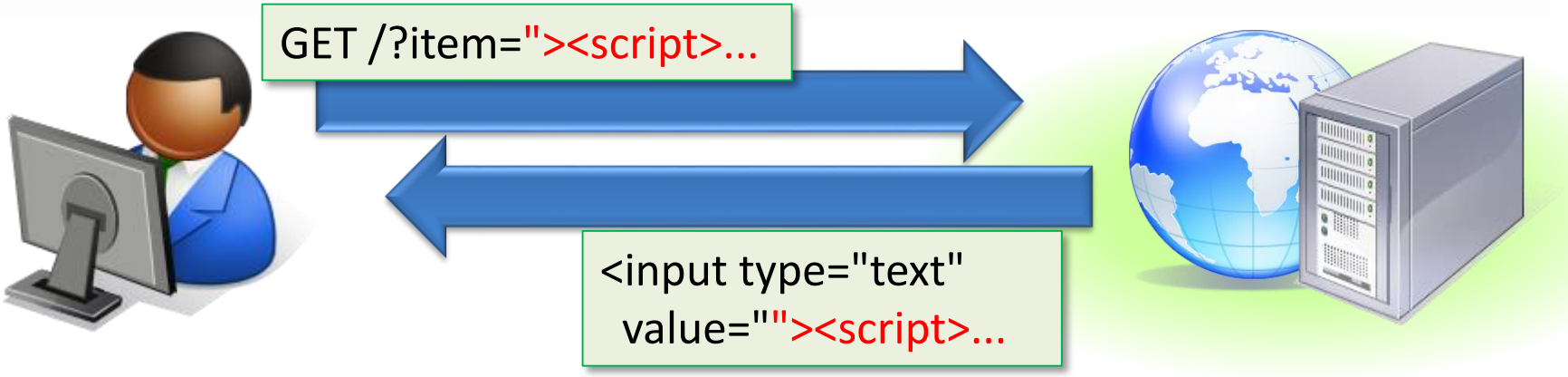
XSSの種類

- 反射型XSS / Type-1
  - ユーザからの送信内容をそのまま表示
  - XSSフィルタ等である程度防御
  - お問い合わせフォーム、サイト内検索
- 持続型XSS / Type-2
  - 攻撃者のスクリプトがサーバ内で保持
  - 掲示板、Webメール

XSSの種類

- 反射型XSS / Type-1
  - ユーザからの送信内容をそのまま表示
  - XSSフィルタ等である程度防御
  - お問い合わせフォーム、サイト内検索
- 持続型XSS / Type-2
  - 攻撃者のスクリプトがサーバ内で保持
  - 掲示板、Webメール

※GETだけでなくPOSTもあり得る



XSSの種類 - 反射型XSS

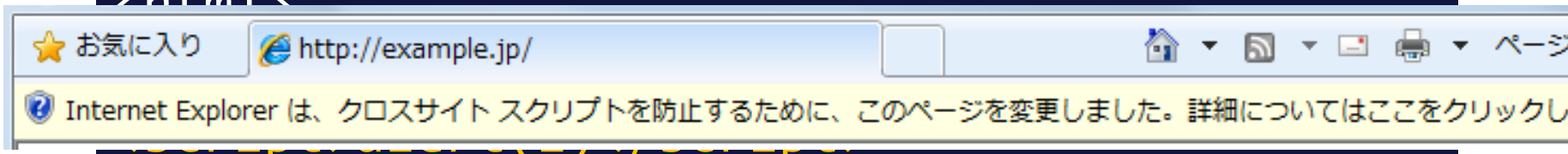
- 反射型XSSの場合、リクエストとレスポンスに同じ内容が含まれる

```
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>
```

XSSの種類 - 反射型XSS

- 反射型XSSの場合、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>
```

XSSの種類 - 反射型XSS

XSSの種類 - 反射型XSS

XSSの種類 - 反射型XSS

- 反射型XSS / Type-1
  - 2007年8月 Twitter「こんにちはこんにちは」

XSSの種類

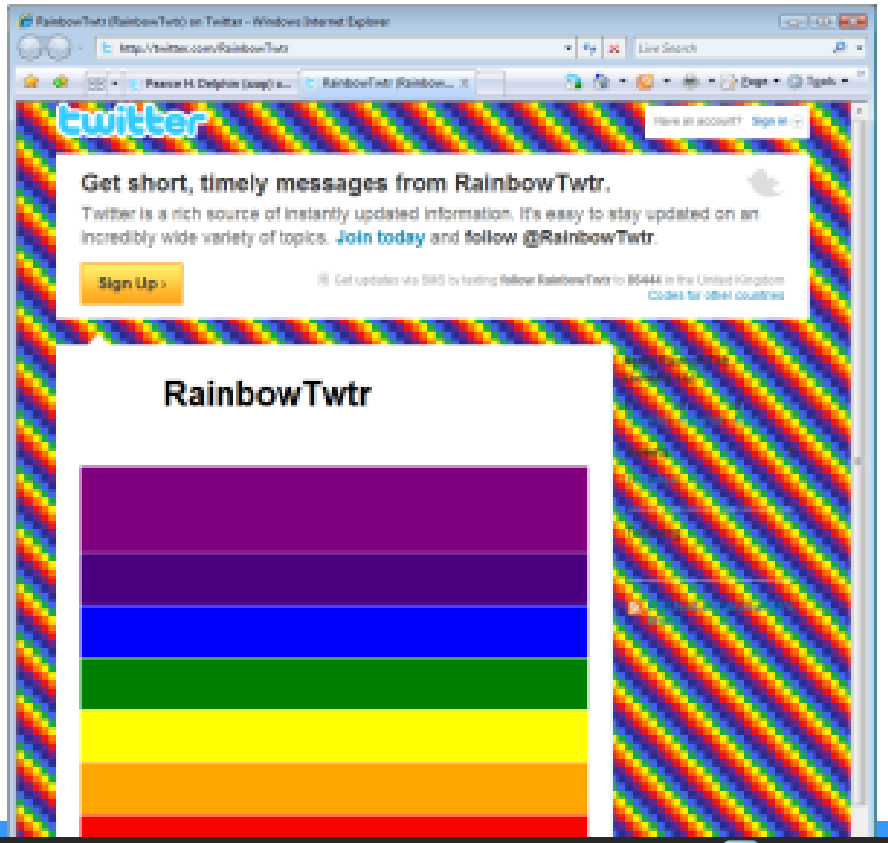
- 反射型XSS / Type-1
  - ユーザからの送信内容をそのまま表示
  - XSSフィルタ等である程度防御
  - お問い合わせフォーム、サイト内検索
- 持続型XSS / Type-2
  - 攻撃者のスクリプトがサーバ内で保持
  - 掲示板、Webメール

XSSの種類

- 反射型XSS / Type-1
  - ユーザからの送信内容をそのまま表示
  - XSSフィルタ等である程度防御
  - お問い合わせフォーム、サイト内検索
- **持続型XSS / Type-2**
  - **攻撃者のスクリプトがサーバ内で保持**
  - **掲示板、Webメール**

XSSの種類 - 持続型XSS

- 2006年6月 mixi「こんにちはこんにちは」
- 2010年9月 Twitter



XSSの種類 - 持続型XSS

- 2006年 Yahoo!メール

ここまでXSSの復習です！

XSS with HTML5 elms

HTML5の新要素によるXSS

- これまでの間違ったXSS対策
  - 危険そうな要素を検出

    `<script> <object> <iframe>`
  - onXXX、hrefなどの名称の属性を検出

    ```
    <div onmouseover=alert(1)>
    <img src=# onerror=alert(1)>
    <a href="javascript:alert(1)>
    ```
- これまで仮にこの方法で網羅できていたとしても…

HTML5の新要素によるXSS

- **HTML5で多数の要素、属性、イベントが導入**

```
<input autofocus pattern="...">
<video onplay="...">
```

HTML5の新要素によるXSS

- いわゆる「ブラックリスト」での対応に漏れ

```
<form>
 <button formaction="javascript:alert(1)">X
 </button>
//http://html5sec.org/#72
```

  - そもそもブラックリスト方式は無理がある
- 「HTML生成時にエスケープ」の原則
  - HTML5と関係なくXSSを防げる

DOM based XSS

XSSの種類

- 反射型XSS / Type-1
  - ユーザからの送信内容をそのまま表示
  - XSSフィルタ等である程度防御

```
//http://example.jp/#<script>alert(1)</script>
div.innerHTML = location.hash;
```

  - 攻撃者のスクリプトがサーバ内で保持
  - 掲示板、Webメール
- DOM based XSS / Type-0
  - JavaScriptが引き起こす
  - サーバ側のHTML生成には問題なし

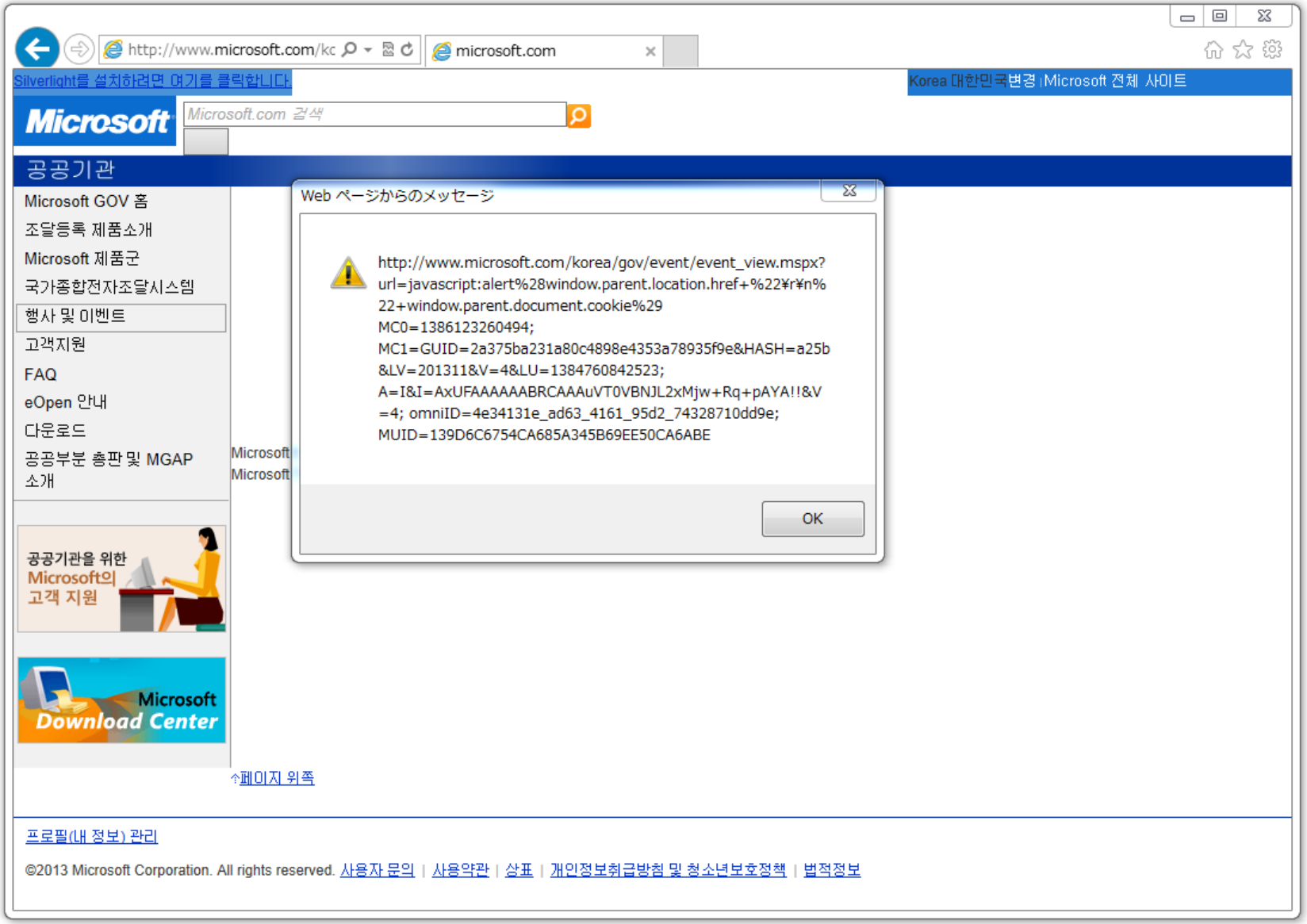
DOM based XSS

- 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

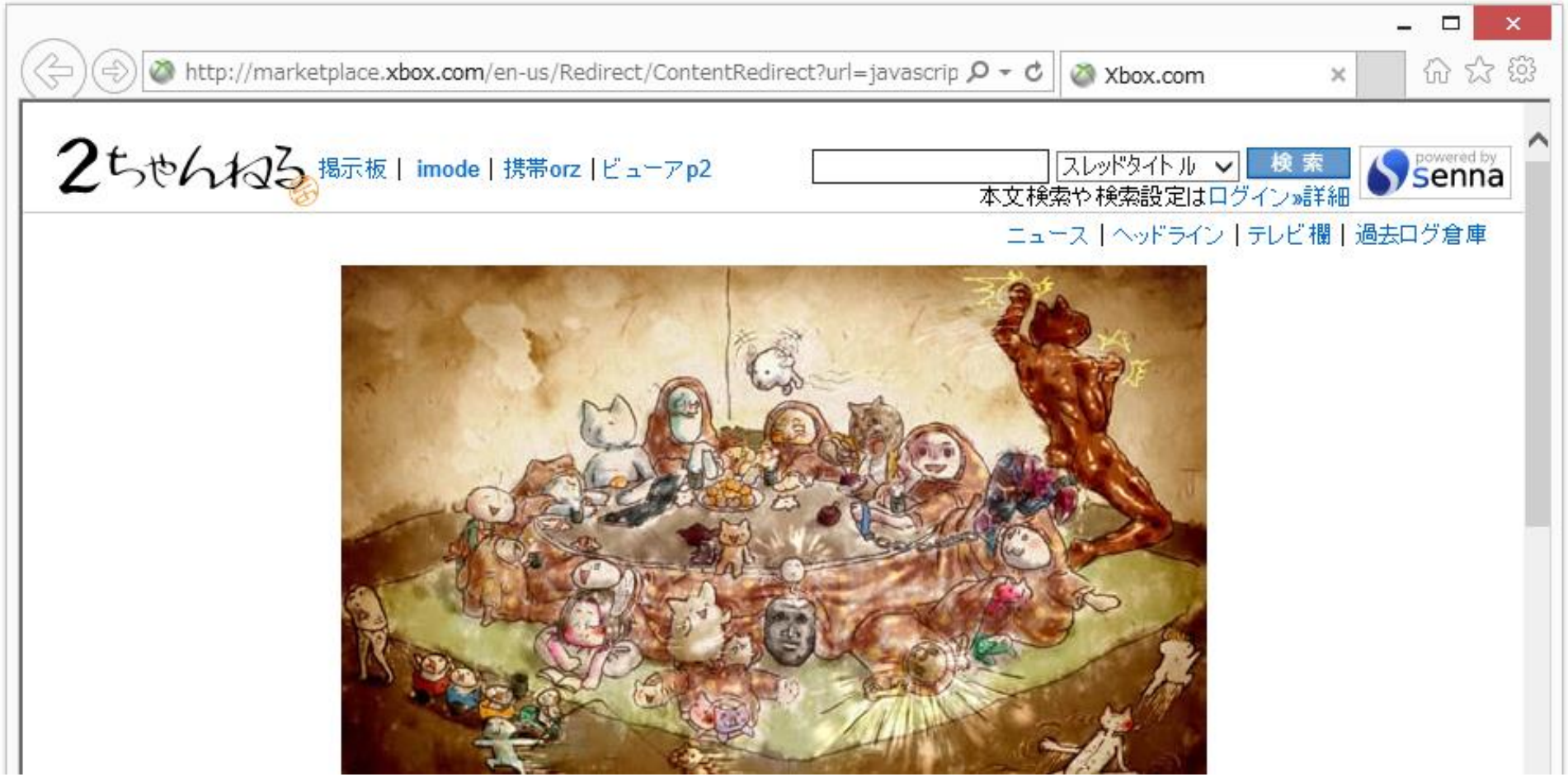
```
//http://example.jp/#<script>alert(1)</script>
div.innerHTML = location.hash.substring(1);
```

- JavaScriptの利用に合わせて増加

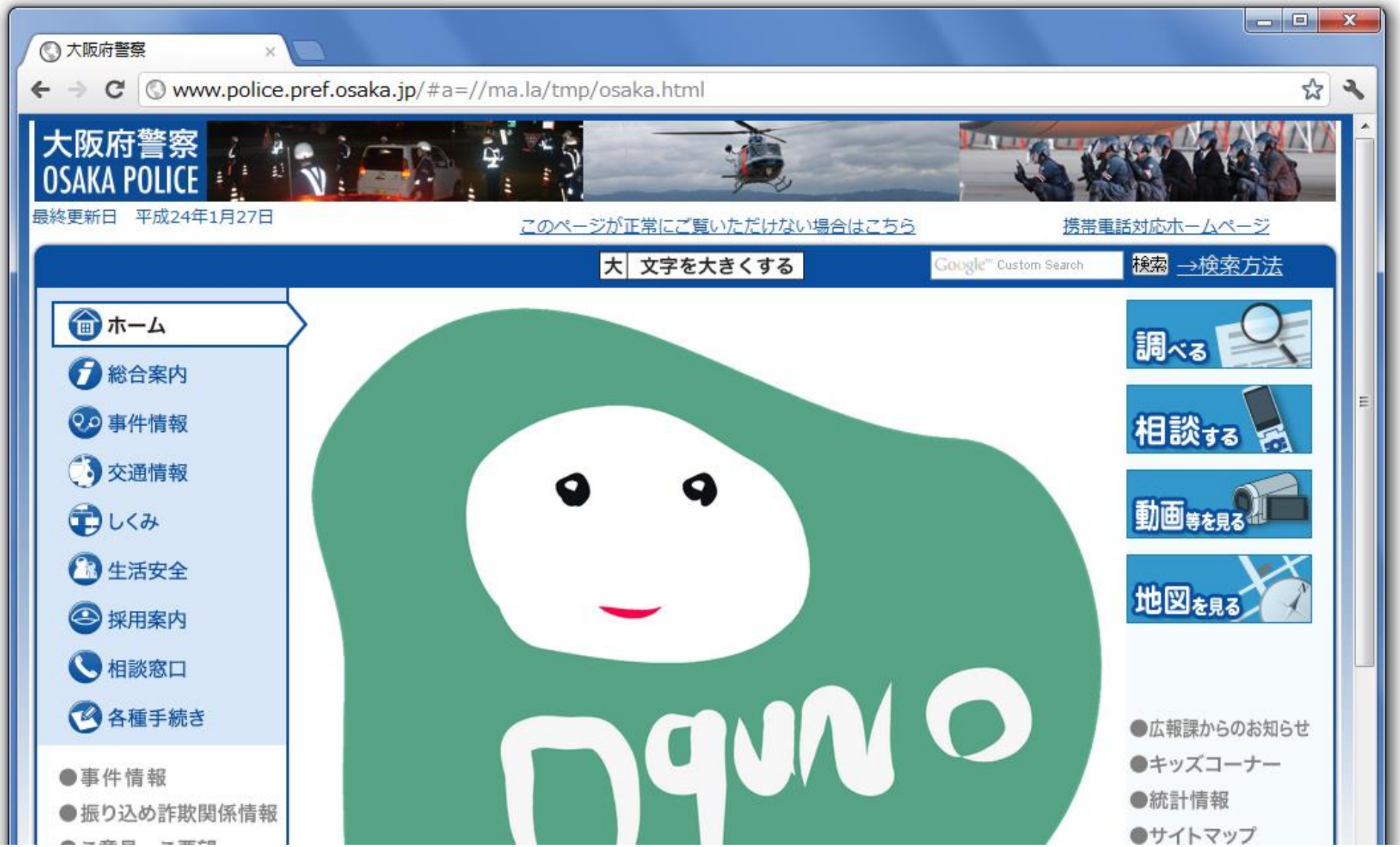
DOM based XSS



DOM based XSS



DOM based XSS



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内の実行コードはサーバ側にログが残らない

```
//http://example.jp/#<script>alert(1)</script>
div.innerHTML = location.hash.substring(1);
```

- history.pushStateでアドレスバー書き換え
  - 技術のあるユーザでもXSSに気づきにくい

DOM based XSS

- DOM based XSSは増えている
  - JavaScriptの大規模化に伴い増加
- サーバ側での対策と原則は同じ
  - HTML生成時にエスケープ
  - URL生成時はhttp(s)のみ
  - CSS backgroundImage等への代入やイベントハンドラの動的生成は避ける

DOM based XSS

- HTML生成時にエスケープ

```
div.innerHTML = s.replace( /&/g, "&amp;" )
    .replace( /</g, "&lt;" )
    .replace( />/g, "&gt;" )
    .replace( /"/g, "&quot;" )
    .replace( /'/g, "&#x27;" );
```

  - むしろtextNodeを使おう!

```
div.appendChild(
    document.createTextElement( s )
);
```

DOM based XSS

- URL生成時はhttp(s)のみ

```
// bad code
div.innerHTML = '<a href="' + url + '">' + url + '</a>';
```

```
if( url.match( /^https?:¥/¥// ) ){
    var elm = docuement.createElement( "a" );
    elm.appendChild( document.createTextNode( url ) );
    elm.setAttribute( "href", url );
    div.appendChild( elm );
}
```

DOM based XSS

- URL生成時はhttp(s)のみ
- リダイレクト時はオープンリダイレクタを発生させないよう同一ホストに制限

```
var base =
   location.protocol + "//" + location.host + "/";
if( url.substring( 0, base.length ) == base ){
    location.href = url;
}
```

DOM based XSS

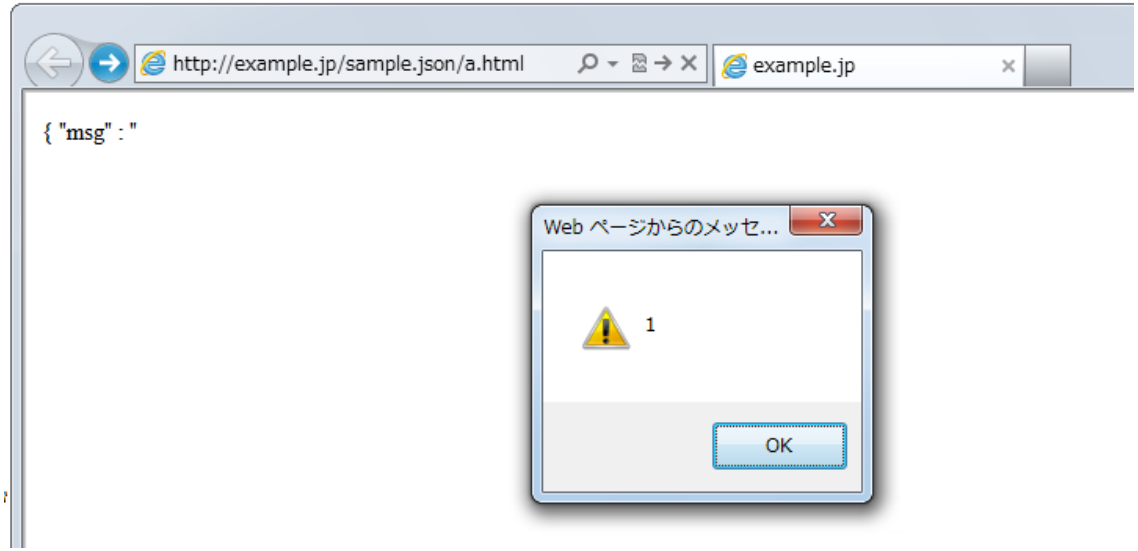
- URLの確認は実はめんどうくさい。
  - 詳細は「めんどうくさいWebセキュリティ」参照

XSS with Ajax data

XSS with Ajax data

- IEのContent-Type無視
  - HTMLではないものがHTMLに昇格してXSS

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{ "msg" : "<script>alert(1)</script>" }
```



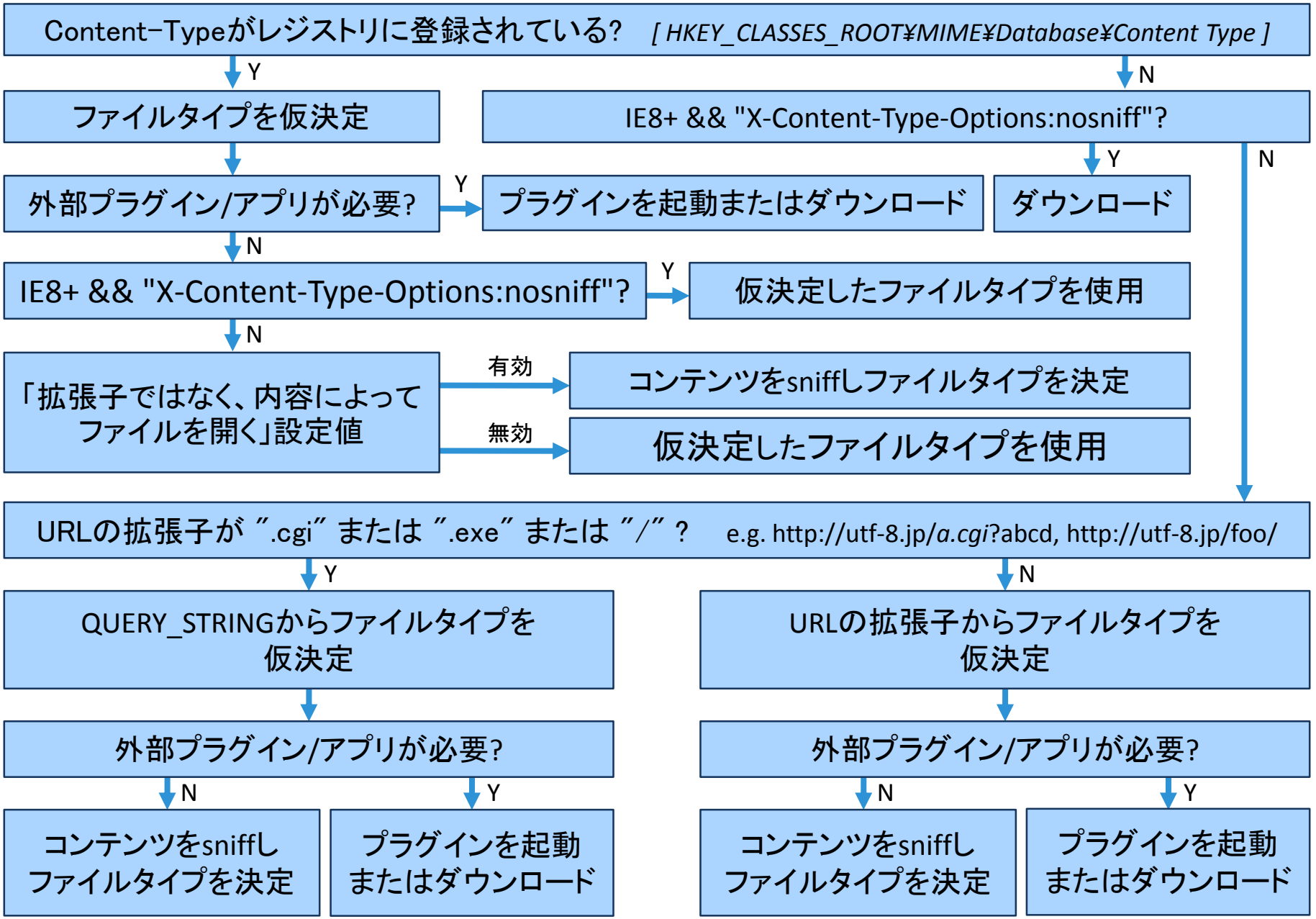
XSS with Ajax data

- IEは最終的に「ファイルタイプ」に基づいてコンテンツを処理する
- Content-Type 以外にも様々な要因からファイルタイプを決定
  - 文書化されていない複雑なメカニズム
  - 「ファイルのダウンロードダイアログで表示されるファイル名の命名規則」
    http://support.microsoft.com/kb/436153/ja
    ファイルタイプ決定のメカニズム解明に近づく唯一のドキュメント

XSS with Ajax data

- ファイルタイプの決定因子
  - "Content-Type" HTTPレスポンスヘッダ
  - "X-Content-Type-Option" HTTPレスポンスヘッダ
  - Windowsレジストリにおける関連付け
  - IEの設定:"拡張子ではなく、内容によってファイルを開く"
  - URL自身
  - コンテンツそのもの

IEにおけるファイルタイプ決定のメカニズム



※これ以外にも例外的な挙動が多数あり

Yosuke HASEGAWA http://utf-8.jp/

XSS with Ajax data

- Ajaxデータを利用したXSS
- Ajaxでやり取りされるデータを直接ブラウザ上で開いたときにXSS
  - JSON - JSON文字列内

    {"text" :"<script>..." }
  - JSONP - callback名

    http://example.com/?callback=<script>...
  - プレーンテキスト, CSV

XSS with Ajax data

- JSONならエスケープできなくはないけど

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{ "msg" : "¥u003cscript¥u003ealert(1)¥u003c/script¥u003e" }
```

  - text/plainとかtext/csvとかエスケープできない

XSS with Ajax data

- 対策
  - X-Content-Type-Optionsを付ける

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
X-Content-Type-Options: nosniff

{ "msg" : "<script>alert(1)</script>" }
```

  - 非HTMLがHTML扱いされることがなくなる

XSSまとめ

XSS beyond HTML5

- XSSの増加
  - HTML5の新要素によるXSS
  - JSコード量の増加 – DOM Based XSS
  - Ajaxデータによる XSS

XSS beyond HTML5

- 対策は従来と大きくは変わらない
  - 「HTML生成時にエスケープ」の原則
  - URL生成時はhttp(s)スキームのみ許可
  - X-Content-Type-Optionsはとにかく付けておけ
- 攻撃可能な箇所は増える
  - 新しいHTML要素、属性、イベント
  - JSコード量の増加
  - Ajax使用量の増加

セキュリティ関係のレスポンスヘッダ

セキュリティ関係のレスポンスヘッダ

- 使いこなすことでよりセキュアに
  - X-XSS-Protection
  - X-Content-Type-Options
  - X-Frame-Options
  - Content-Security-Policy
  - X-Download-Options
  - Strict-Transport-Security

セキュリティ関係のレスポンスヘッダ
X-XSS-Protection

X-XSS-Protection

- XSS保護機能の制御
  - IE,Chrome,Opera,Safariで有効
  - XSS保護機能をそのページのみ無効にする

```
X-XSS-Protection: 0
```

  - XSS保護機能を明示的に有効にする
    (デフォルト有効になっている)

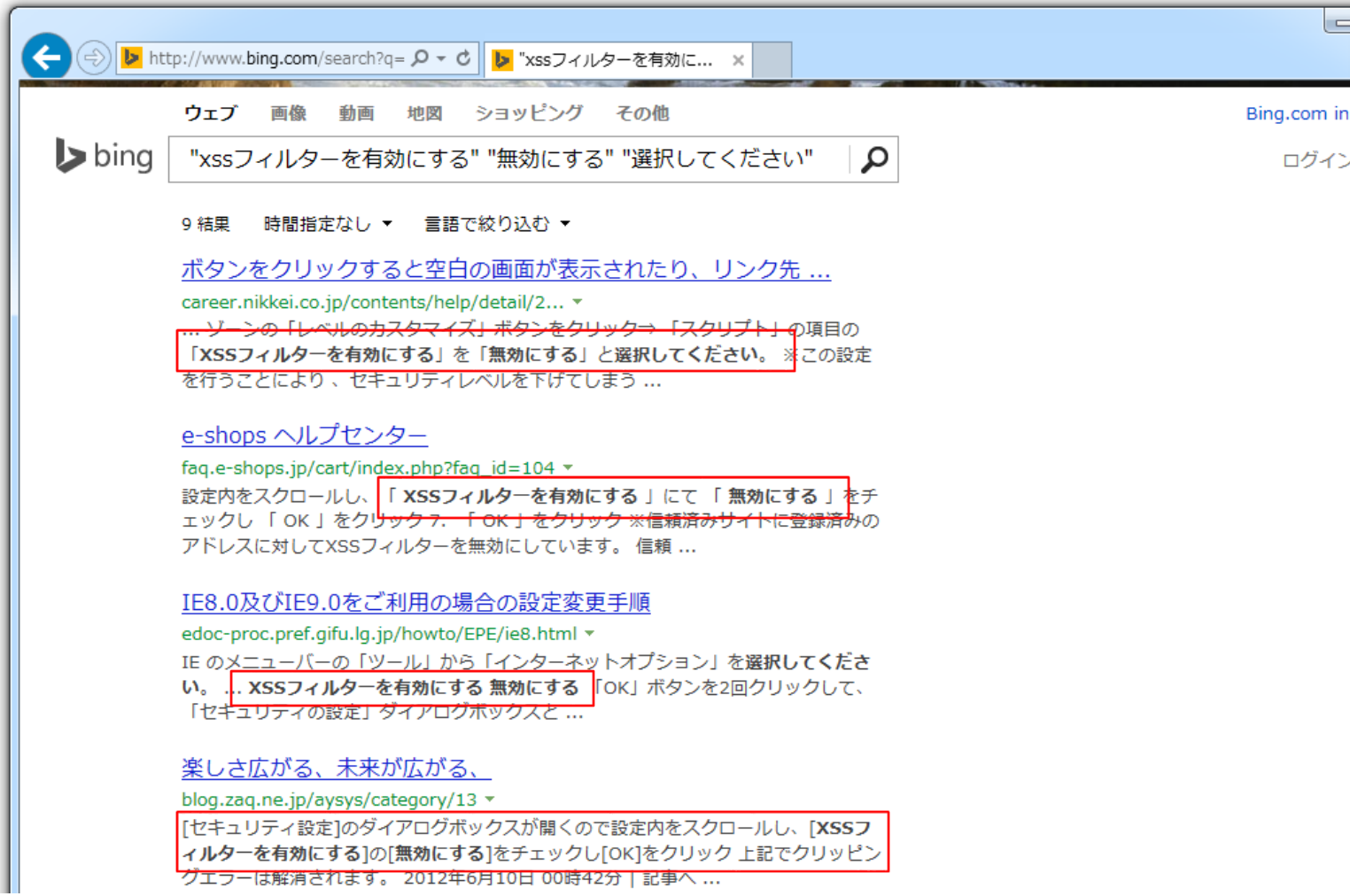
```
X-XSS-Protection: 1
```

  - XSS保護機能を有効にし、XSS検知時に空白画面を表示

```
X-XSS-Protection: 1; mode=block
```

X-XSS-Protection

- 「XSSフィルターを無効に設定」やめて！



X-XSS-Protection

- XSSフィルターの誤検知のエラーを消すには
  - 該当ページにXSSがないか慎重に検査したうえで
  - そのページのみX-XSS-Protection:0を指定
- ブラウザの設定変更を指示しないで！

セキュリティ関係のレスポンスヘッダ
X-Content-Type-Options

X-Content-Type-Options

- Content-Typeを厳格に扱う
  - 非HTMLをHTML扱いしない
    JSONやCSVによるXSSの防止

```
Content-Type: application/json; charset=utf-8
X-Content-Type-Options: nosniff

{ "message", "<script>alert(1)</script>" }
```

  - 非JSを<script src>として読み込まない
    script src経由での情報漏えい防止
    Firefox、Chromeなどでも。

```
Content-Type: application/json; charset=utf-8
X-Content-Type-Options: nosniff

[ "secret", "message", "is", "here" ]
```

X-Content-Type-Options

- 副作用はほとんどないので、全コンテンツにつけるべき
  - 稀有な副作用例：JSONP/JSONで共通処理

```
Content-Type: application/json; charset=utf-8
X-Content-Type-Options: nosniff

{ "message", "<script>alert(1)</script>" }
```

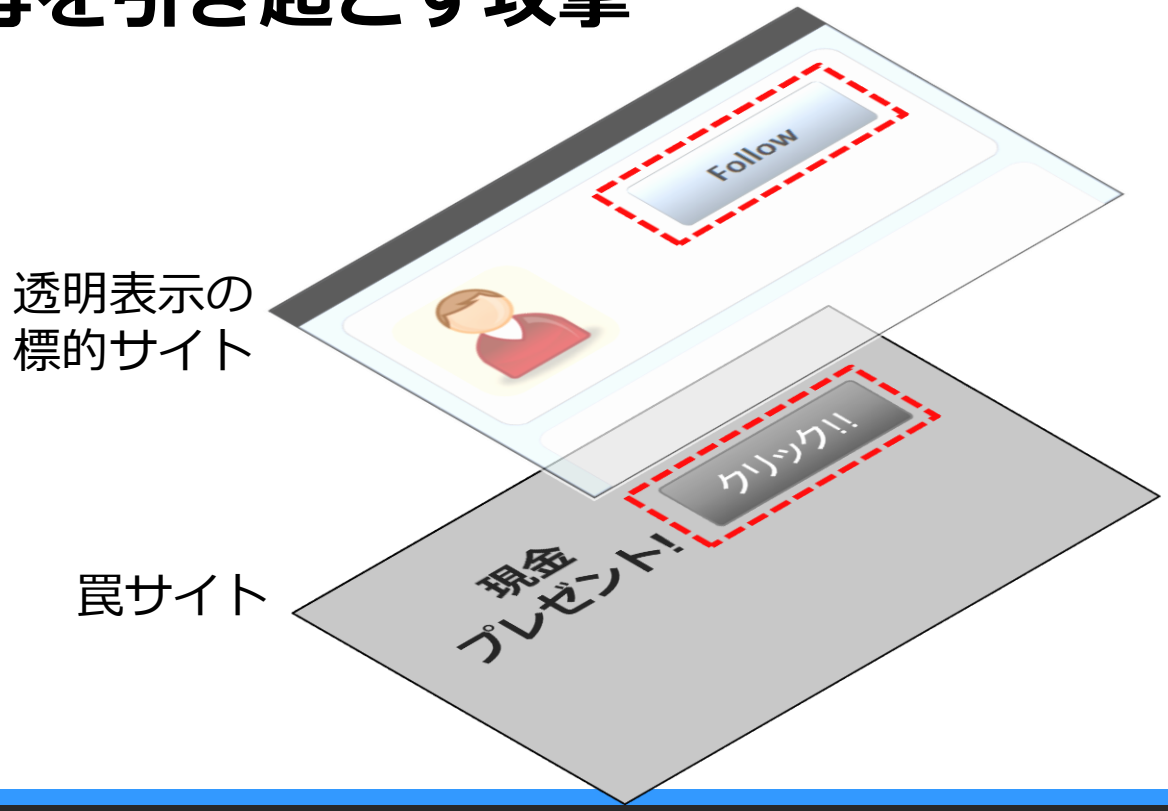
```
Content-Type: application/json; charset=utf-8
X-Content-Type-Options: nosniff

callback( { "message", "<script>alert(1)</script>" } )
```

```
<script src="api/jsonp?cb=callback"></script> …失敗する
```

セキュリティ関係のレスポンスヘッダ
X-Frame-Options

X-Frame-Options

- クリックジャッキング
  - 標的サイトを透明に重ね、意図しないクリック等を引き起こす攻撃



X-Frame-Options

- クリックジャッキング対策
  - iframe,frame等での埋め込みを禁止する
  - 全ての埋め込みを禁止

```
X-Frame-Options: DENY
```

  - 同一オリジン以外からの埋め込みを禁止

```
X-Frame-Options: SAMEORIGIN
```

  - 指定オリジン以外からの埋め込みを禁止

```
X-Frame-Options: ALLOW-FROM http://example.jp
```

X-Frame-Options

- X-Frame-Options: ALLOW-FROM

```
X-Frame-Options: ALLOW-FROM http://example.jp
```

  - http://example.jpからの埋込みのみ許可
  - ALLOW-FROMに指定できるオリジンはひとつだけ。
  - 複数のオリジンからの埋め込み許可はそのままではできない
    - Firefoxのみスペース区切りで複数オリジンの指定可能

X-Frame-Options

- ALLOW-FROMの複数オリジン対応
  - 呼出し元オリジンごとに識別子をURLに付与

```
// http://parent.example.jp/上
<iframe src="http://child.example.jp/?from=p1"></iframe>
```

```
# child.example.jp/
my $allows = { p1 => 'http://parent.example.jp' };
my $from = $allows->{ $params->{from} };
if( $from ){
    print "X-Frame-Options: ALLOW-FROM $from¥n";
}else{
    print "X-Frame-Options: DENY¥n";
}
```

セキュリティ関係のレスポンスヘッダ
Content-Security-Policy

Content-Security-Policy

- ヘッダで指定されたソースからしか画像やJSを読み込めなくする

```
HTTP/1.1 200 OK
Content-Security-Policy: default-src 'self'; image-src *;
Content-Type: text/html; charset=utf-8
```

  - Chrome拡張やFirefoxOSアプリの開発時にイラッとするアレ
  - 使いこなせばXSSも怖くないけれど、実際の運用は超たいへん

セキュリティ関係のレスポンスヘッダ
X-Download-Options

X-Download-Options

- IE8以降でダウンロード時に「開く」ボタンを非表示

```
HTTP/1.1 200 OK
Content-Disposition: attachment; filename="index.html"
X-Download-Options: noopen

<html><script>...
```

X-Download-Optionsなし

X-Download-Optionsあり

X-Download-Options

- ダウンロード時の「開く」ボタン非表示
  - 添付ファイルによる蓄積型のXSSを予防することができる
  - 安全なコンテンツをユーザーが直接開くことができなくなるので、利便性は下がる
  - 不特定多数のユーザーが添付ファイルを掲載できるWebアプリでは一考の価値あり

セキュリティ関係のレスポンスヘッダ
Strict-Transport-Security

Strict-Transport-Security

- HTTPSを強制するための指令

```
HTTP/1.1 200 OK
Strict-Transport-Security: max-age=15768000
```

```
HTTP/1.1 200 OK
Strict-Transport-Security: max-age=15768000; includeSubdomains
```

  - これ以降のHTTPへのアクセスはHTTPSに置き換わる
  - max-age は有効期間を秒数で指定
  - includeSubDomainsが指定されるとサブドメインも対象

Strict-Transport-Security

- HTTPSサイトのみがStrict-Transport-Securityヘッダを返す

```
HTTP/1.1 200 OK
Strict-Transport-Security: max-age=15768000
```

  - HTTPはすでに汚染されているかもしれないので
  - Firefox、Chromeなどは常にHTTPSで通信する "preload HSTS" のリストを持っている

Strict-Transport-Security

- Preload HSTS list
  - Firefox、Chromeは常にHTTPSで通信するサイトのリストを持っている
  - 申請すれば掲載してもらえる(常時SSL化!)
  - cybozu.com を真に常時 SSL にする話 | Cybozu Inside Out | サイボウズエンジニアのブログ

    http://developer.cybozu.co.jp/tech/?p=6096

質問タイム

Question ?

Question? 質問

hasegawa@utf-8.jp
hasegawa@netagent.co.jp

@hasegawayosuke

http://utf-8.jp/

@hasegawayosuke