PHPカンファレンス北海道

徳丸本に載っていない
Webアプリケーションセキュリティ

2012年4月21日

徳丸　浩

本日お話しする内容

- キャッシュからの情報漏洩に注意
- クリックジャッキング入門
- Ajaxセキュリティ入門
- ドリランド カード増殖祭りはこうしておこった…かも?

徳丸浩の自己紹介

- 経歴
  - 1985年 京セラ株式会社入社
  - 1995年 京セラコミュニケーションシステム株式会社(KCCS)に出向・転籍
  - 2008年 KCCS退職、HASHコンサルティング株式会社設立
- 経験したこと
  - 京セラ入社当時はCAD、計算幾何学、数値シミュレーションなどを担当
  - その後、企業向けパッケージソフトの企画・開発・事業化を担当
  - 1999年から、携帯電話向けインフラ、プラットフォームの企画・開発を担当
    Webアプリケーションのセキュリティ問題に直面、研究、社内展開、寄稿などを開始
  - 2004年にKCCS社内ベンチャーとしてWebアプリケーションセキュリティ事業を立ち上げ
- その他
  - 1990年にPascalコンパイラをCabezonを開発、オープンソースで公開
    「大学時代のPascal演習がCabezonでした」という方にお目にかかることも
- 現在
  - HASHコンサルティング株式会社　代表　http://www.hash-c.co.jp/
  - 京セラコミュニケーションシステム株式会社　技術顧問　http://www.kccs.co.jp/security/
  - 独立行政法人情報処理推進機構　非常勤研究員　http://www.ipa.go.jp/security/

本を書きました

2011年3月5日初版第1刷
2011年12月13日 初版第5刷

キャッシュからの情報漏洩に注意

Webアプリケーションとキャッシュ

- フレームワークのキャッシュ機能
- リバースプロキシ
- フォワードプロキシ
- ブラウザのキャッシュ機能

- キャッシュは、Webアプリケーションの負荷を軽減し、応答を高速化するために用いられる
- サイト運営側が設置するキャッシュと、閲覧側（ISP等を含む）が設置するものがある

キャッシュからの情報漏洩とは?

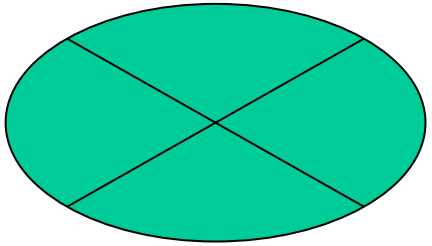
- いわゆる「別人問題」
- 秘密情報がキャッシュされて、それを別人が見る状況
  - 最初のアクセスは正当な権限あり
    →表示内容がキャッシュされるに
  - 次のアクセスは権限のないユーザ
    →同一URLを参照すると、キャッシュされた情報を閲覧してしまう

- 続きはデモで

Demoの構成

ブラウザ

PROXY
squid



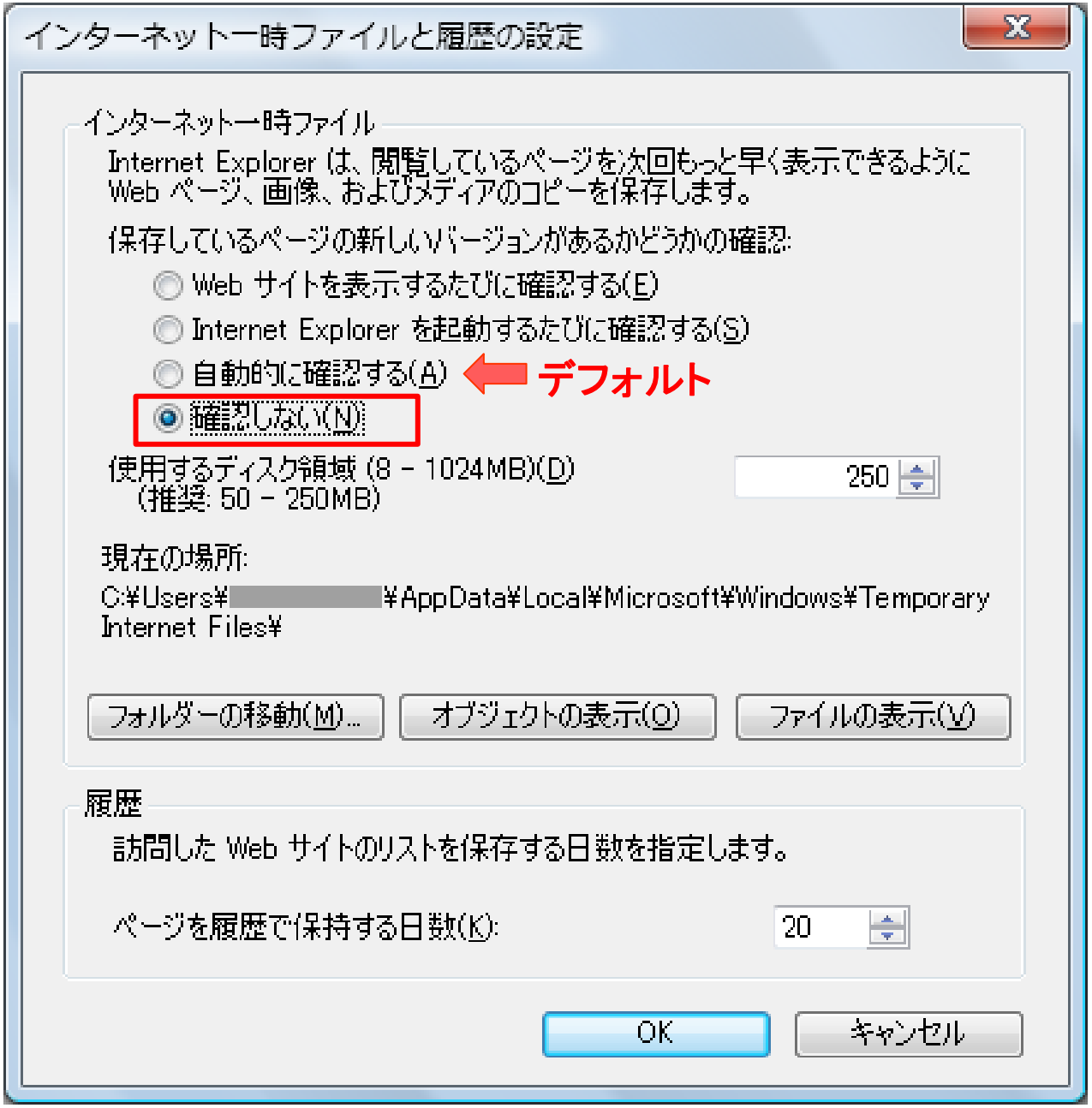
OpenPNE 3.4.12.1
PHP/MySQL
Ubuntu 10.04LTS

```
refresh_pattern ^ftp:           1440    20%     10080
refresh_pattern ^gopher:        1440    0%      1440
refresh_pattern -i (/cgi-bin/|¥?) 0     0%      0
refresh_pattern (Release|Package(.gz)*)$        0       20%     2880
refresh_pattern . 1440 20% 4320 override-lastmod
```

※デモ環境では、Webサーバーと Proxyサーバーは同一VM上で稼働

Demo

ブラウザキャッシュのデモ



そもそもこれは脆弱性なのか?

- PROXY設定の問題ではないのか?
  - PROXY設定の問題とは言い切れない
  - squidの標準的な機能の範囲である
  - PROXY設定の基準が明確にあるわけではない
- RFC2616 13.4 Response Cacheability
  - (原文)A response received with a status code of 200, 203, 206, 300, 301 or 410 MAY be stored by a cache and used in reply to a subsequent request, subject to the expiration mechanism, unless a cache-control directive prohibits caching.
  - (日本語訳)200, 203, 206, 300, 301, 410 のステータスコードと共に受信されたレスポンスは、cache-control 指示子がキャッシングを禁止していなければ、キャッシュによって保存され、以降のリクエストへの応答に使用され、期限メカニズムに従う事ができる。

http://tools.ietf.org/html/rfc2616#section-13.4
http://www.studyinghttp.net/cgi-bin/rfc.cgi?2616#Sec13.1.3 より引用

ブラウザのキャッシュはどうか

- ブラウザのキャッシュも同様だが
- そもそも、ブラウザを共有することが問題
- Windows95,98,MEの時代ならともかく、2012年の時点では、端末は共有しても、ユーザは別にすべし。そうすれば問題は発生しない
- つまり、「はてなブックマークでのデモ事例」は許容可能
- とはいえ、Webサイト提供者としては対策することが望ましい

対策

- HTTPレスポンスヘッダで「キャッシュを禁止するヘッダ」を応答することが根本対策
- セキュアプログラミング講座には…
  1. Cache-Control: private
     Webサーバから返されるコンテンツがただ一人のユーザのためのものであることを示す。このコンテンツは、複数のユーザが共有されるキャッシュに記録されるべきではないことを表している。ただし、これは、一人のユーザのみが利用するキャッシュ（ブラウザのキャッシュ等）への記録を禁じるものではない。Cache-Control: private のみが指定されている場合、何らかのキャッシュへの記録が行われるおそれがある。
  2. Cache-Control: no-store
     このヘッダは、Webサーバから返されてくるコンテンツをキャッシュに記録するな、という指示である。
  3. Cache-Control: no-cache
     一見「キャッシュを使うな」のように見えるこのヘッダが実際に意味するところは少々ニュアンスが異なる。このヘッダの意味は、いちどキャッシュに記録されたコンテンツは、現在でも有効か否かを本来のWebサーバに問い合わせて確認がとれない限り再利用してはならない、という意味である。
  4. Cache-Control: must-revalidate
     このヘッダは、キャッシュに記録されているコンテンツが現在も有効であるか否かをWebサーバに必ず問い合わせよ、という指示である。

http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/405.html より引用

対策（続き）

- HTTP/1.0対策として
  - Pragma: no-cache
  - 非標準の方法であり、確実なものではない
- その他の手法
  - 「現場の知恵」としてURL（クエリ文字列）にランダムな数字を付加することが行われる
    - 例: http://example.jp/private.php?rand=8314329428479210
  - HTTPSにする
    - 通常のPROXYはHTTPSのメッセージをキャッシュしない（できない）
    - リバースPROXY、ブラウザはキャッシュする可能性があるので過信しないこと
  - テストする
    - 別ユーザ or 非ログイン状態で、同じURLをアクセスする

乱数をURLにつける例

クリックジャッキング入門

クリックジャッキング攻撃

- ターゲットの画面をiframe上で「透明に」表示する
- その下にダミーの画面を表示させる。ユーザにはダミーの画面が透けて見える
- 利用者がダミーの画面上のボタンを押すと、実際には全面のターゲット画面のボタンが押される

本当の画面（前面、透明）

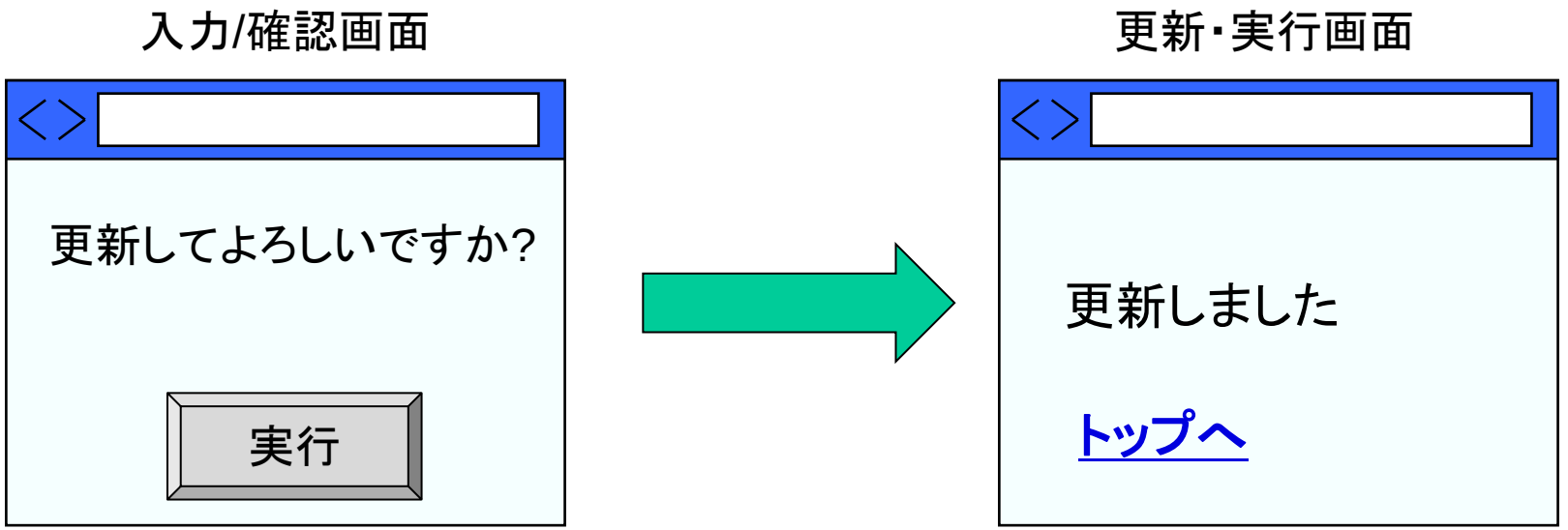
ダミーの画面（後面）

- 続きはデモで

クリックジャッキングの対策

- クリックジャッキングの影響はクロスサイト・リクエストフォージェリ(CSRF)と同等
  - ユーザの意識とは無関係に、ユーザの権限で操作が行われる
- クリックジャッキングされると困るページには、X-FRAME-OPTIONSヘッダを指定する（徳丸本P63）
  - frame/iframeを禁止して良い場合
    ```
    header('X-FRAME-OPTIONS', 'DENY');
    ```
  - frame/iframeを禁止できないが単一ホストの場合
    ```
    header('X-FRAME-OPTIONS', 'SAMEORIGIN');
    ```
- CSRF対策のトークン発行しているページが対象となる
- メタ要素によるX-FRAME-OPTIONS指定は無効です。
  徳丸本第3刷までの記述は間違いです(_ _)

CSRF対策との関係



- トークン埋め込み
- X-FRAME-OPTIONS ヘッダ出力

- トークン確認

トークン埋め込みしている画面に、X-FRAME-OPTIONS ヘッダを出力する（全ての画面で出力しても良い）

Ajaxセキュリティ入門

Ajaxはあたりまえになったけど、Ajaxのセキュリティは当たり前になってない

たとえば入門書の問題

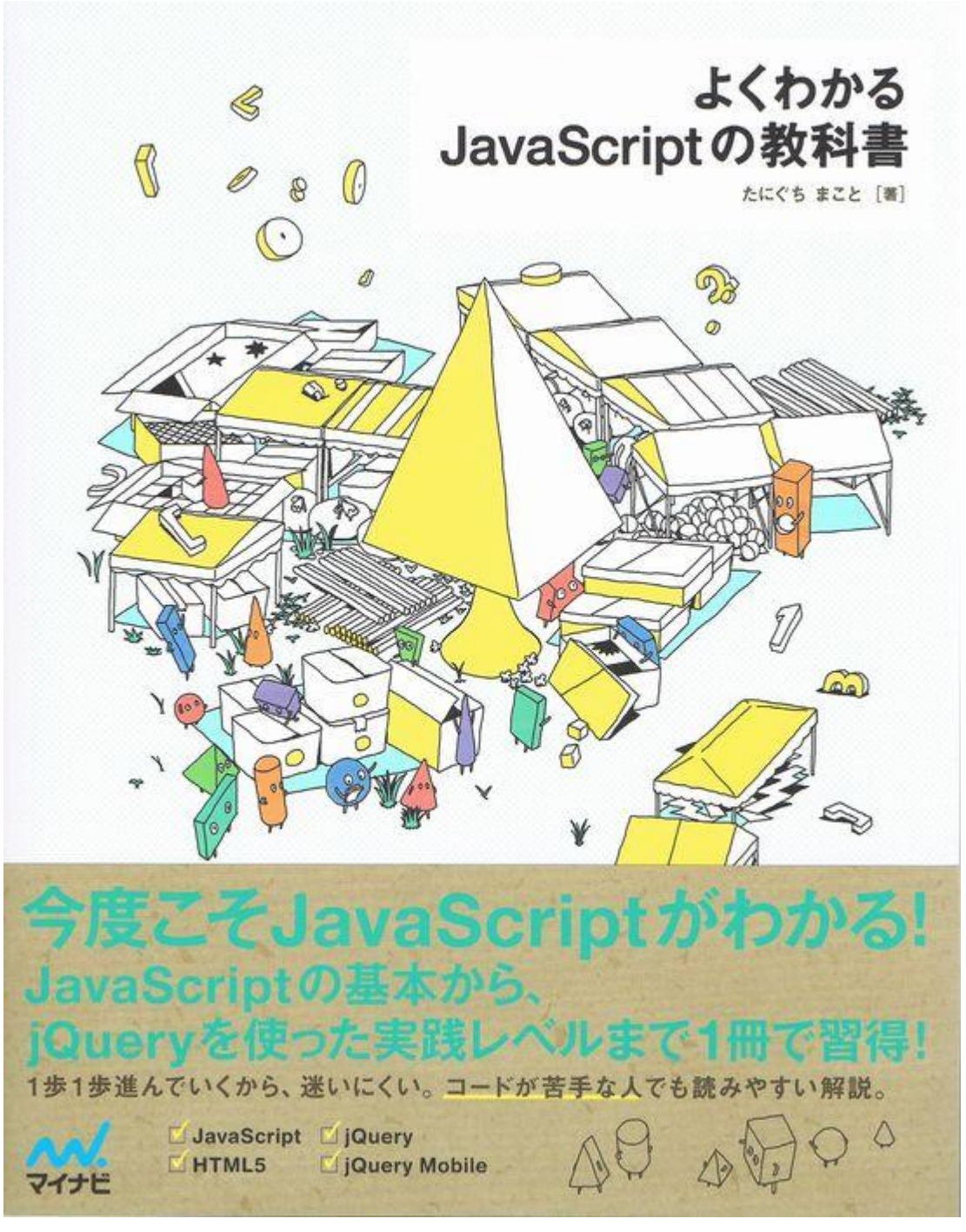
とある入門書のサンプル

```
01 ...
02 $.getJSON('images.json', null, function(json) {
03     var ul = $('<ul>');
04     for (var i=0; i<json.length; i++) {
05         ul.append('<li><a href="' + json[i].image.replace('.jpg',
   '_big.jpg') + '"><img src="' + json[i].image + '" width="100"></
   a><br><p>' + json[i].caption + '</p></li>');
06     }
07     $('body').append(ul);
08 });
09 });
10 ...
```

```
<p>' + json[i].caption + '</p>
```


よくわかるJavaScriptの教科書、たにぐちまこと著、2012 P228より引用

この本です


よくわかる
JavaScriptの教科書
たにぐち まこと［著］
今度こそJavaScriptがわかる!
JavaScriptの基本から、
jQueryを使った実践レベルまで1冊で習得!
1歩1歩進んでいくから、迷いにくい。コードが苦手な人でも読みやすい解説。
JavaScript
jQuery
HTML5
jQuery Mobile
マイナビ

動かしてみる

```
1  <html>↓
2  <head><meta http-equiv="Content-Type" content="text/html;charset=UTF-8">↓
3  <script src="js/jquery.js"></script>↓
4  <script>↓
5  $(function() {↓
6    $.getJSON('images.json', null, function(json) {↓
7      var ul = $('<ul>');↓
8      for (var i = 0; i < json.length; i++) {↓
9        ul.append('<li><a href="' + json[i].image.replace('.jpg', '_big.jpg') + '"><img src="'↓
10         + json[i].image + '"width="100"></a><br><p>' + json[i].caption + '</p></li>');↓
11     }↓
12     $('#slideshow').html(ul);↓
13   });↓
14 });↓
15 </script>↓
16 <head>↓
17 <body>↓
18 <div id="slideshow"></div>↓
19 </body>↓
20 </html>↓
21 [EOF]
```

```
[↓
{↓
  "image": "img/1.jpg",↓
  "image_big": "img/1_big.jpg",↓
  "caption": "キャプション1"↓
}↓
]↓
```

結果

JSON作成をPHPに

```php
<?php
 $image = 'img/1.jpg';
 $image_big = 'img/1_big.jpg';
 $caption  = 'キャプション1';
 $a = json_encode(array(array("image" => $image,
              "image_big" => $image_big,
              "caption" => $caption)));
 header('Content-Type: application/json; charset=utf8');
 echo $a;
```

生成されるJSON

```
[{"image":"img¥/1.jpg","image_big":"i
mg¥/1_big.jpg","caption":"¥u30ad¥u30e
3¥u30d7¥u30b7¥u30e7¥u30f31"}]
```

結果

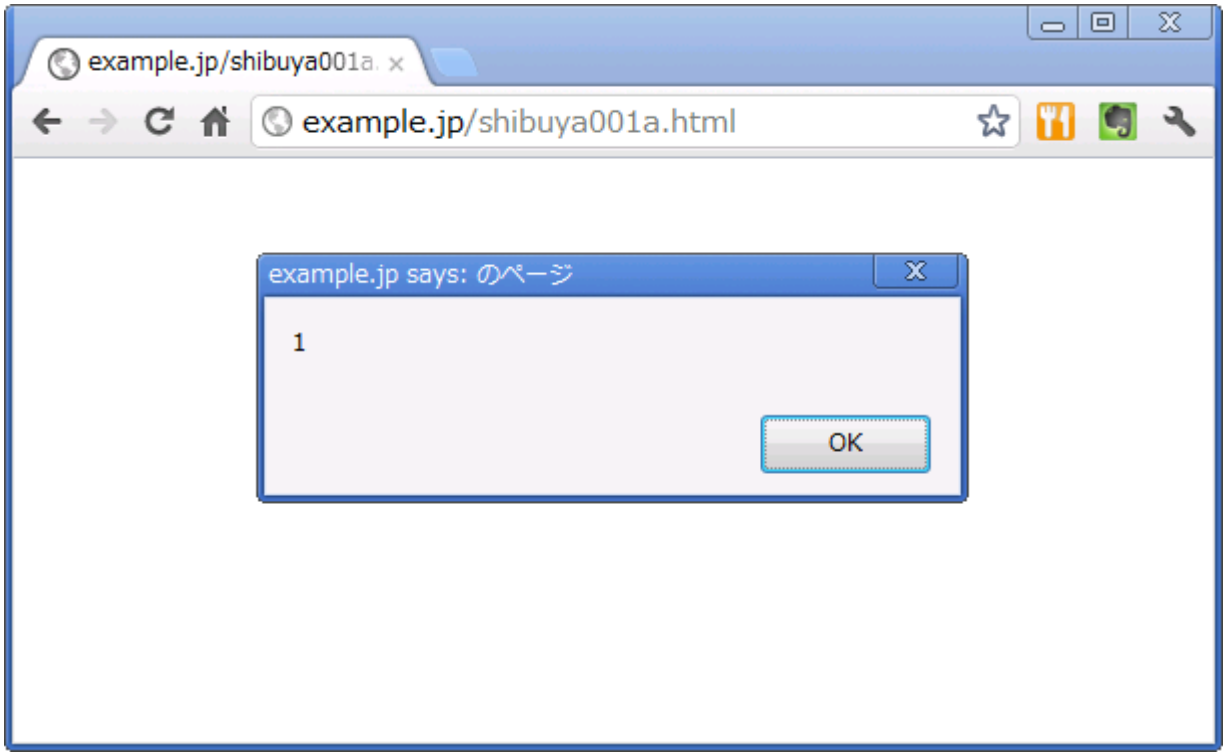
データにJavaScriptを入れてみる

```
$image = 'img/1.jpg';↓
$image_big = 'img/1_big.jpg';↓
$caption  = 'キャプション1<script>alert(1)</script>';↓
$a = json_encode(array(array("image" => $image,↓
            "image_big" => $image_big,↓
            "caption" => $caption)));↓
header('Content-Type: application/json; charset=utf8');↓
echo $a;↓
```

生成されるJSON

```
[{"image":"img¥/1.jpg","image_big":"i
mg¥/1_big.jpg","caption":"¥u30ad¥u30e
3¥u30d7¥u30b7¥u30e7¥u30f31<script>ale
rt(1)<¥/script>"}]
```

結果



脆弱性の原因と対策

- htmlメソッドによりHTMLテキストを設定しているにも関わらず、データをHTMLエスケープしていない
- ただし、元文献は固定テキストなので、必ずしも脆弱性とは言えない
- HTMLエスケープするタイミングは、以下の候補がある
  - ブラウザでレンダリングする箇所
  - JSONを組み立てる段階で、あらかじめHTMLエスケープしておく

次の話題

evalインジェクション

JSON解釈をevalでやってみる

```
1  <html>↓
2  <head><meta http-equiv="Content-Type" content="text/html;charset=UTF-8">↓
3  <script src="js/jquery.js"></script>↓
4  <script>↓
5  $(function() {↓
6    $.get('json2.php', null, function(text) {↓
7      var json = eval(text); // 文字列として読み込んでevalする↓
8      var ul = $('<ul>');↓
9      for (var i = 0; i < json.length; i++) {↓
10       ul.append('<li><a href="' + json[i].image.replace('.jpg', '_big.jpg') + '"><img src="'↓
11         + json[i].image + '"width="100"></a><br><p>' + json[i].caption + '</p></li>');↓
12     }↓
13     $('#slideshow').html(ul);↓
14   });↓
15 });↓
16 </script>↓
17 <head>↓
18 <body>↓
19 <div id="slideshow"></div>↓
20 </body>↓
21 </html>↓
```

JSON作成も自前で

```
<?php
  $image = 'img/1.jpg';
  $image_big = 'img/1_big.jpg';
  $caption  = 'キャプション2';
  $a = '[{"image":"' . $image . '","image_big":"' .
    $image_big . '","caption":"' . $caption . '"}]';
  echo $a;
```

生成されるJSON

```
[{"image":"img/1.jpg","image_big":"img/1_b
ig.jpg","caption":"キャプション2 "}]
```

結果

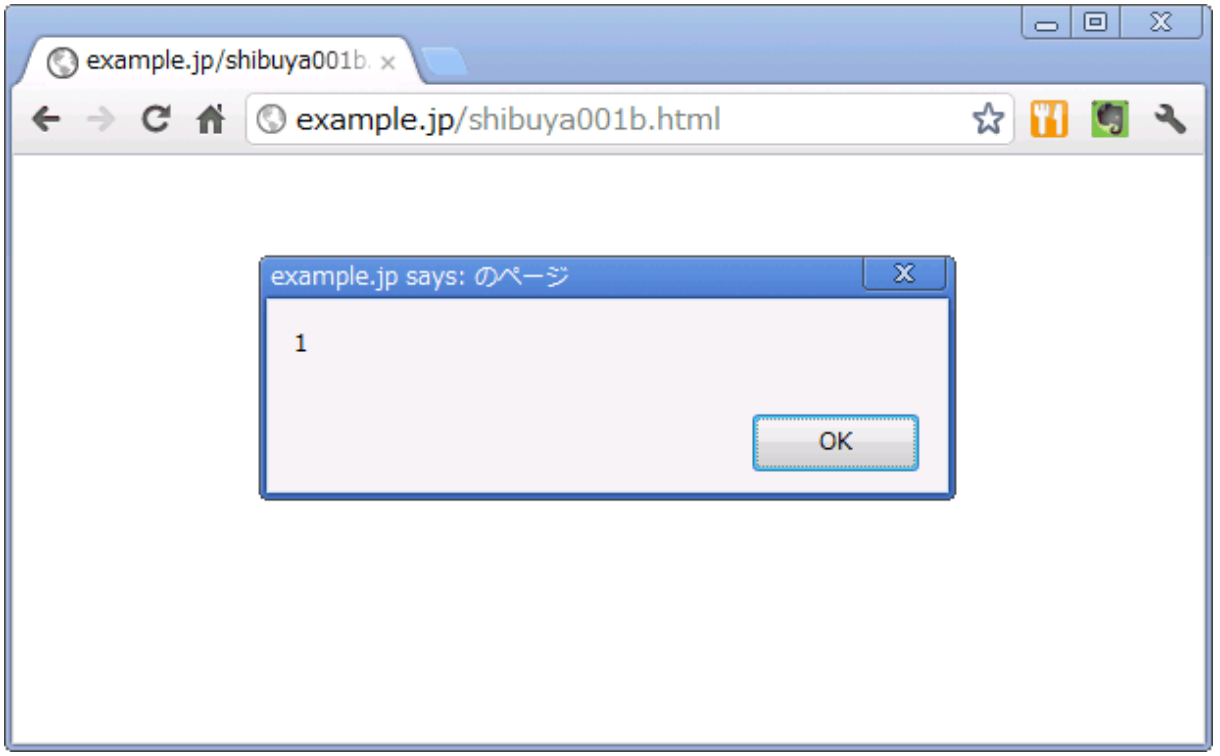
データにJavaScriptを入れてみる

```
<?php
  $image = 'img/1.jpg';
  $image_big = 'img/1_big.jpg';
  $caption  = 'キャプション2x"+alert("1")+"';
  $a = '[{"image":"' . $image . '","image_big":"' .
    $image_big . '","caption":"' . $caption . '"}]';
  echo $a;
```

生成されるJSON

```
[{"image":"img/1.jpg","image_big":
"img/1_big.jpg","caption":
"キャプション2x"+alert("1")+""}]
```

結果



脆弱性の原因と対策

- JSON/JavaScriptとしての適切なエスケープを怠っていることが根本原因
- 根本単位策：JSON生成を自前でしないで、信頼できるライブラリを用いる
- 保険的対策（強く推奨）:evalでJSONを解釈しない。

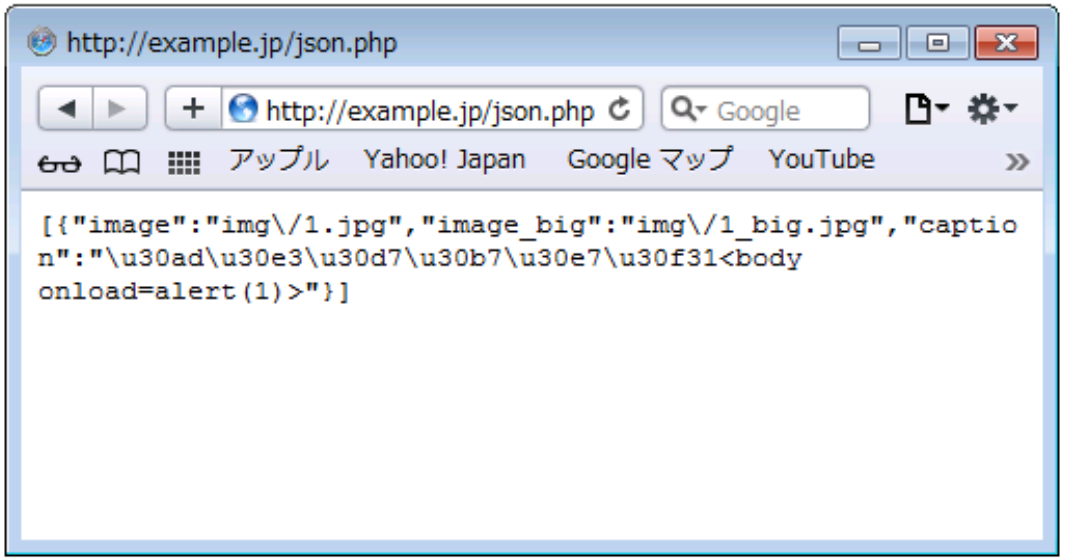
次の話題

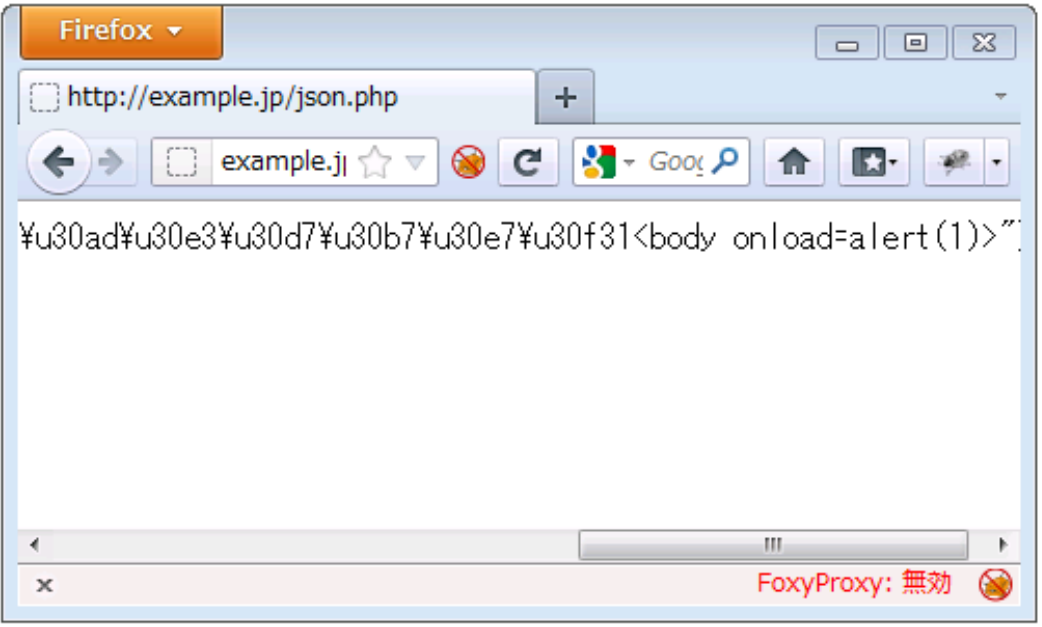
json.phpを直接ブラウズしてみる

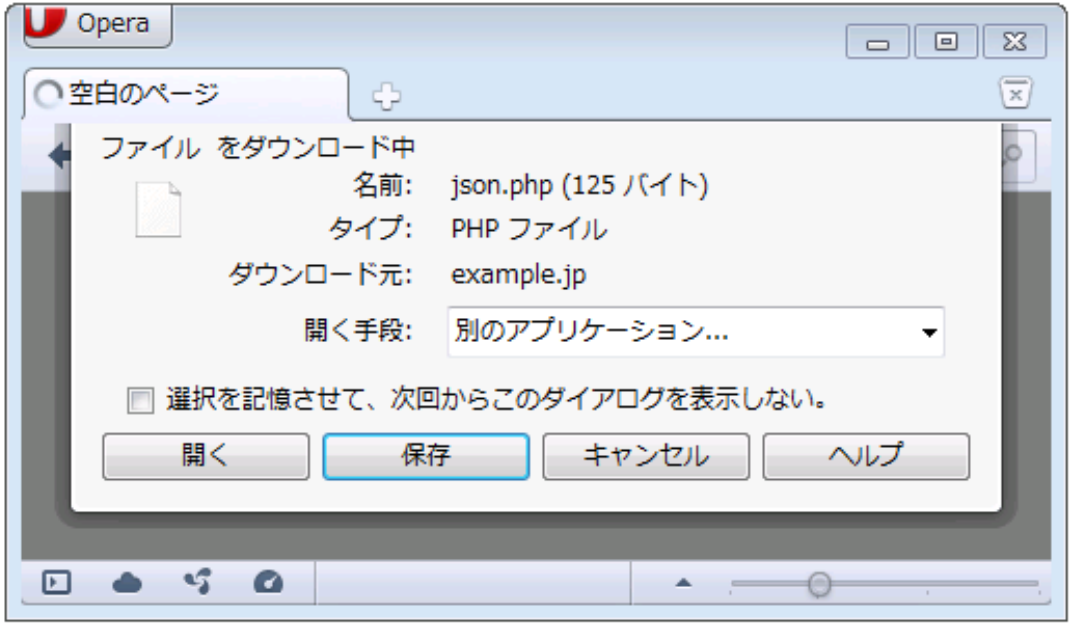
データを少しいじりましょう

```
<?php↓
 $image = 'img/1.jpg';↓
 $image_big = 'img/1_big.jpg';↓
 $caption  = 'キャプション1<body onload=alert(1)>';↓
 $a = json_encode(array(array("image" => $image,↓
              "image_big" => $image_big,↓
              "caption" => $caption)));↓
 header('Content-Type: application/json; charset=utf8');↓
```

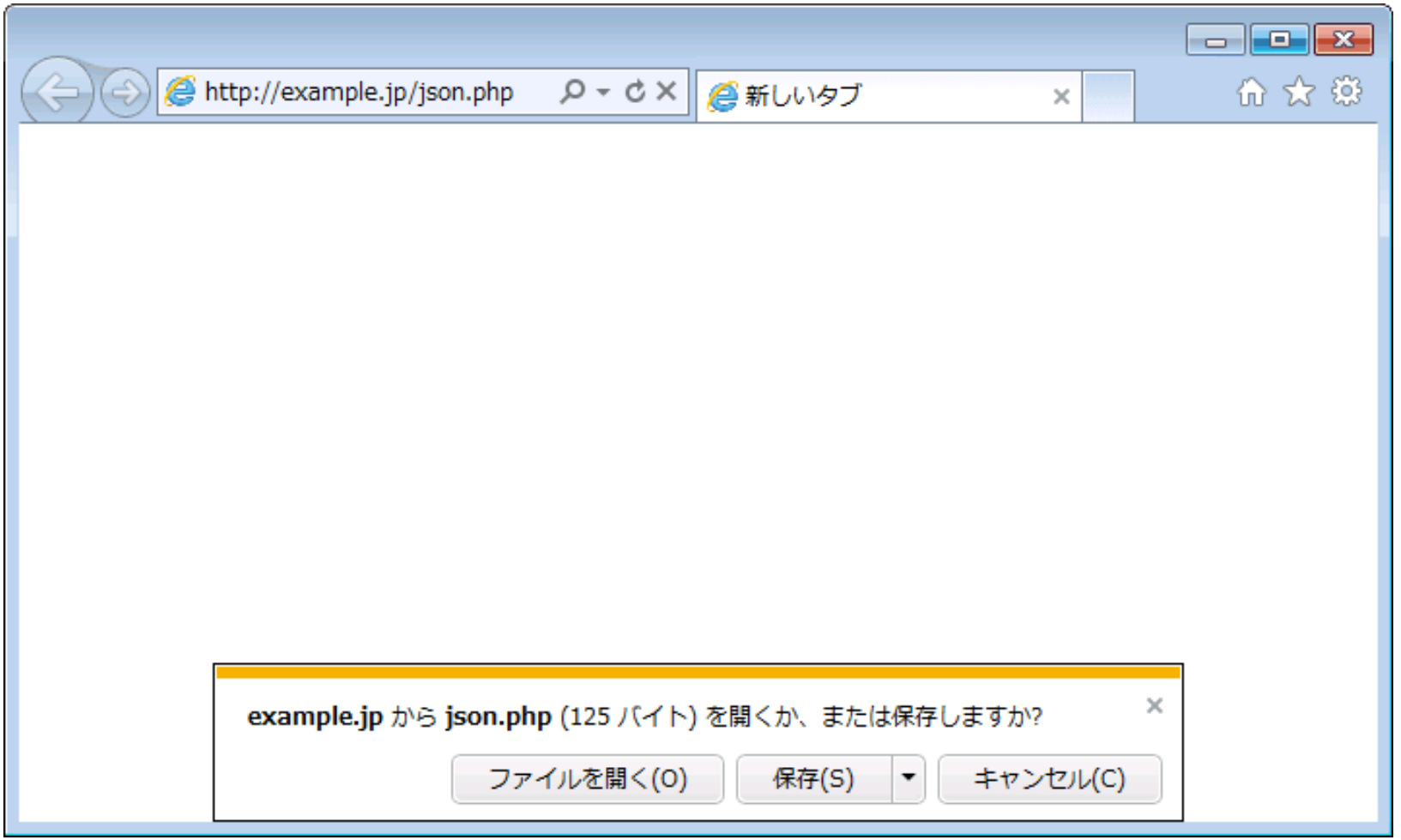
さまざまなブラウザでの結果







で、IE9は?



大丈夫なのか

でも、だめ

■[SECURITY] X-Content-Type-Options: nosniff つかわないやつは死ねばいいのに!

2011-01-06: IE8ということを追記 & ちょっと間違いを修正。

あけましておめでとうございます。

年明け早々ですが、Internet Explorerの話題です。IEはご存じの通り、Content-Type だけでなくコンテンツの内容なども sniff することでファイルタイプを決定しているため、画像ファイルやテキストファイルをHTMLと判定してしまい、クロスサイトスクリプティングが発生することが昔からたびたび報告されていました[*1]。現在は幾分マシになったとはいえ、IEのファイルタイプの判定アルゴリズムは非常に難解であり、現在でも状況によってはWebサイト運営者のまったく意図していないかたちでのXSSが発生する可能性があったりします。そういうわけで、IEがコンテンツを sniff してHTML以外のものをHTML扱いしてしまうことを防ぐために、動的にコンテンツを生成している場合には、とにかく**あらゆるコンテンツのレスポンスヘッダに `X-Content-Type-Options: nosniff` を付与**するようにしましょう。少なくとも、IE8以降ではHTML以外のものがHTML扱いされてXSSが発生する、ということはなくなります[*2]。これを付与することによる副作用は

- 間違った Content-Type を吐きだした場合にIE上でも化けて表示される
- レスポンスヘッダの分だけ、ちょっとだけレスポンスのサイズが大きくなる

くらいです。前者はそもそも間違えたContent-Typeを吐いているという状況がおかしいので、Webアプリケーション側が修正すべきで、後者についてはたかだか33バイトなので我慢してください。

X-Content-Type-Options: nosniff を使っていない場合に起こり得るXSSのシナリオとしては、サーバ側ではPDFを動的に生成(あるいはユーザからアップロード可能)となっていたが、被害者のWindowsにはPDF readerがインストールされていないので Content-Type: application/pdf は未知であり、HTMLと判定されてXSS が発生、などが考えられます。

まあ、死ぬべきは IE だというのはまっとうな意見ですね →
https://twitter.com/#!/kazuho/status/22828878628126720

http://d.hatena.ne.jp/hasegawayosuke/20110106/p1 より引用

■[SECURITY] X-Content-Type-Options: nosniff つかわないやつは死ねばいいのに! 500 users タンポポをのせる

146

2011-01-06: IE8ということを追記 & ちょっと間違いを修正。

あけま

年明け

sniff

サイ

ファ

ない

HTM

ンス

がHT

-
-

くらい

修正

X-Content-Type-Options: nosniff を使っていない場合に起こり得るXSSのシナリオとしては、ユーザ入力からPDFを動的に生成(あるいはユーザからアップロード可能)となっていたが、被害者のWindowsにはPDF readerがインストールされていないので Content-Type: application/pdf は未知であり、HTMLと判定されてXSS が発生、などが考えられます。

まあ、死ぬべきは IE だというのはまっとうな意見ですね →
https://twitter.com/#!/kazuho/status/22828878628126720

http://d.hatena.ne.jp/hasegawayosuke/20110106/p1 より引用

X-Content-Type-Options: nosniff を入れてみる

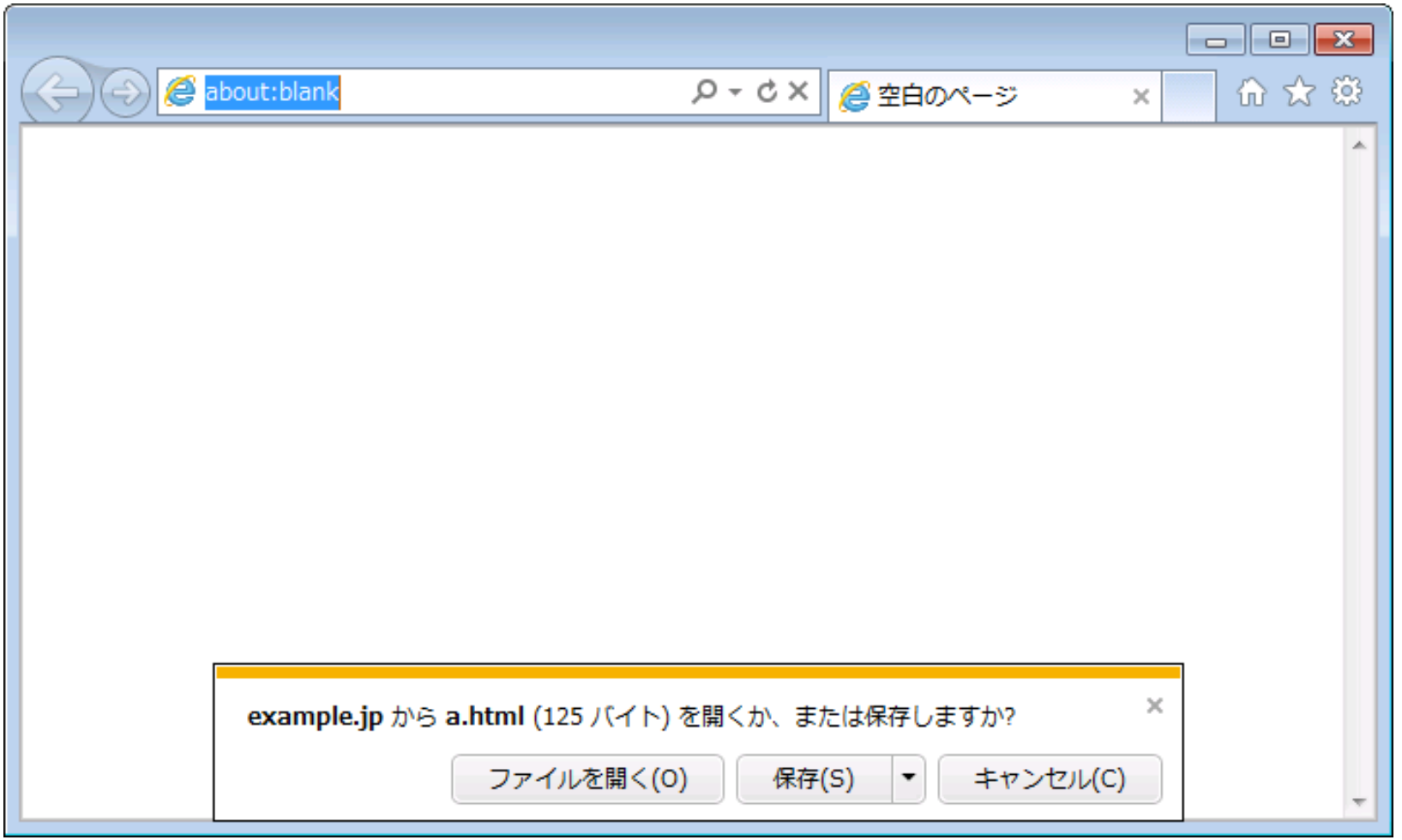
```
<?php↓
 $image = 'img/1.jpg';↓
 $image_big = 'img/1_big.jpg';↓
 $caption  = 'キャプション1<body onload=alert(1)>';↓
 $a = json_encode(array(array("image" => $image,↓
              "image_big" => $image_big,↓
              "caption" => $caption)));↓
 header('Content-Type: application/json; charset=utf8');↓
 header('X-Content-Type-Options: nosniff');↓
 echo $a;↓
```

生成されるJSON

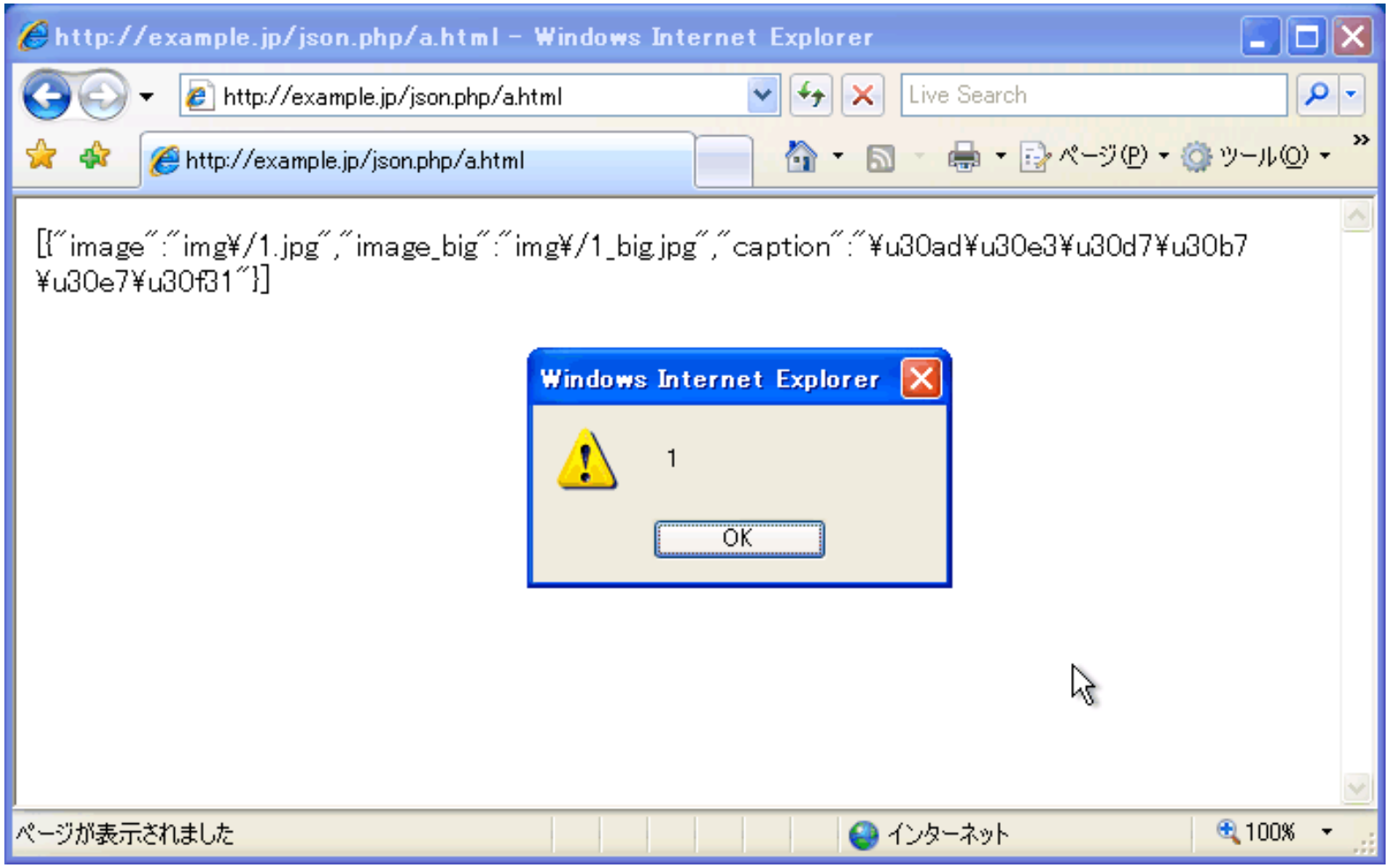
```
X-Content-Type-Options: nosniff
Content-Length: 125
Content-Type: application/json; charset=utf8

[{"image":"img¥/1.jpg","image_big":"img¥/1_b
ig.jpg","caption":"¥u30ad¥u30e3¥u30d7¥u30b7¥
u30e7¥u30f31<body onload=alert(1)>"}]
```

IE9だとOK(IE8以上)



IE7だとだめ



Kazuho Oku
@kazuho

フォロー

(IEが) 死ねばいいのに！ / X-Content-Type-Options: nosniff つかわないやつは死ねばいいのに! - 葉っぱ日記 http://htn.to/whSXLN

34 リツイート　**20** お気に入り

2011年1月6日 - 10:36 Hatenaから · このツイートをサイトに埋め込む

返信　リツイート　お気に入りに登録

対策

- 必須対策（全ブラウザ共通）
  - レスポンスがブラウザによりtext/htmlと解釈されないようにする
    X-Content-Type-Options: nosniff
    Content-Type: application/json; charset=utf8

- IE6,7対策
  - IE7以前をサポートしない
  - ブラウザからの直接リクエストを受け付けない
  - 「<」、「>」などもエスケープする

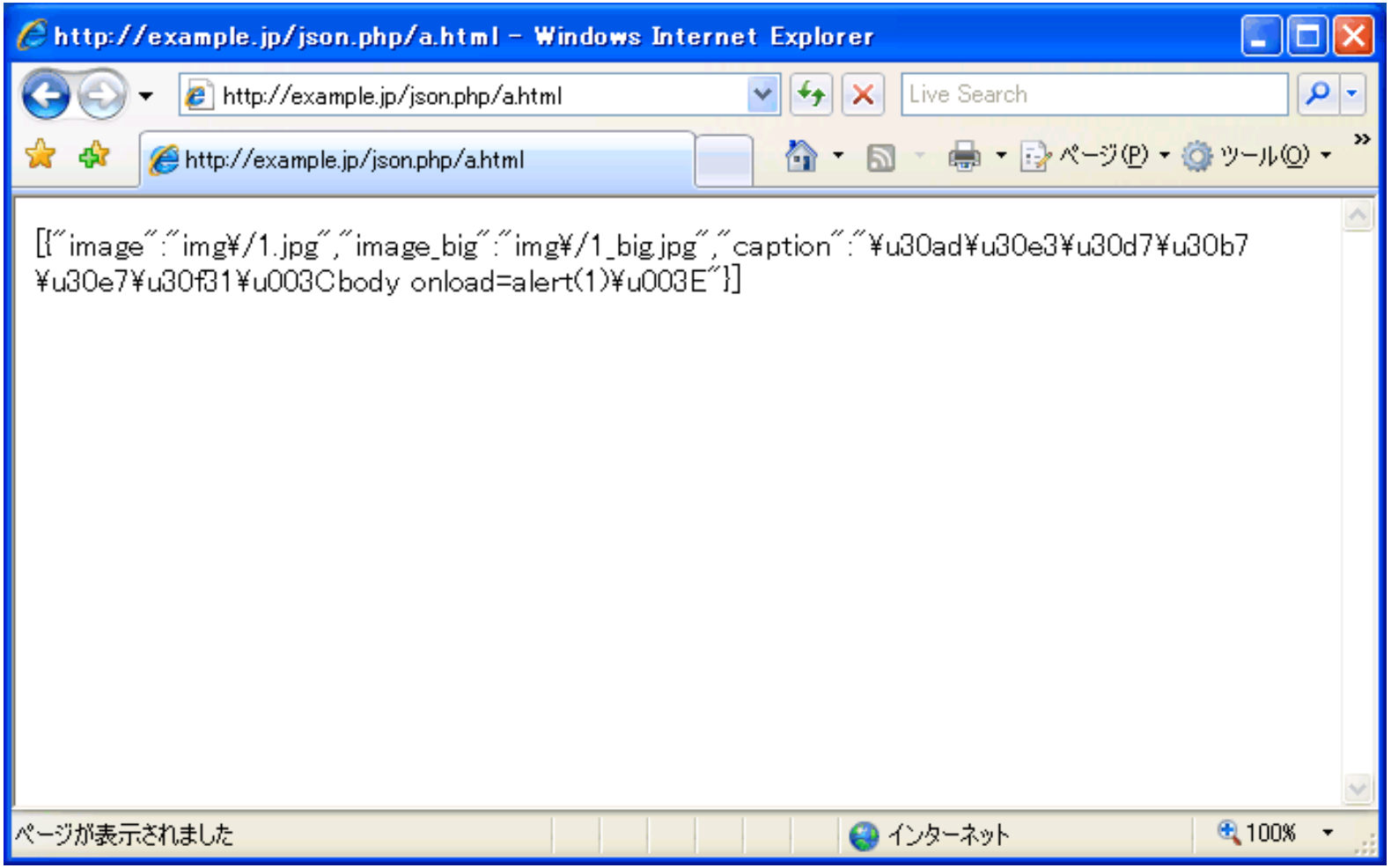
Json_encodeのパラメータを追加

```
<?php↓
 $image = 'img/1.jpg';↓
 $image_big = 'img/1_big.jpg';↓
 $caption  = 'キャプション1<body onload=alert(1)>';↓
 $a = json_encode(array(array("image" => $image,↓
                "image_big" => $image_big,↓
                "caption" => $caption)),↓
            JSON_HEX_TAG | JSON_HEX_APOS | JSON_HEX_QUOT);↓
 header('Content-Type: application/json; charset=utf8');↓
 header('X-Content-Type-Options: nosniff');↓
 echo $a;↓
```

生成されるJSON

```
[{"image":"img¥/1.jpg","image_big":"img¥/1_bi
g.jpg","caption":"¥u30ad¥u30e3¥u30d7¥u30b7¥u3
0e7¥u30f31¥u003Cbody onload=alert(1)¥u003E"}]
```

今度は大丈夫



次の話題はJSONハイジャック

Firefox3などで発現。現在の主要ブラウザでは対策されている

JSONハイジャックとは

- JSONを罠サイトからscript要素で読み出す
- 利用者のブラウザからは、正規のCookieが送信されるので、認証ずみの状態でJSONが取得できる
- なんらかの手法で、このJSONの中味を読むのがJSONハイジャック

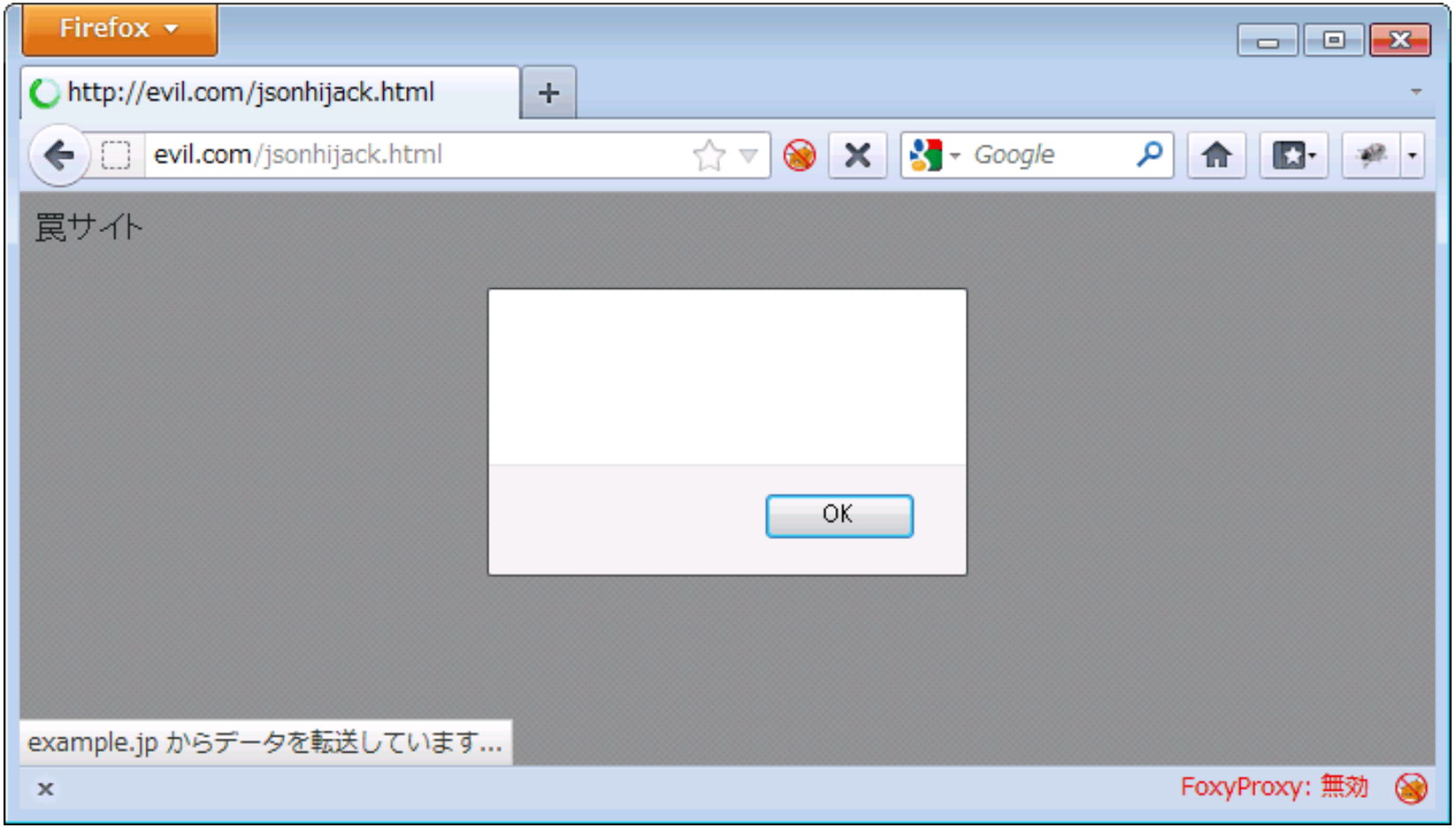
```
これは罠サイト
<script>
  ここにJSONを読み出す仕掛けを置く
// script要素で読み出したJSON
[{"name":"Yamada", "mail":"yama@example.jp"}]
</script>
```

サンプルスクリプト（罠）

```
1  <body onload="alert(x)">↓
2  罠サイト↓
3  <script>↓
4  var x = "";↓
5  Object.prototype.__defineSetter__("image",  function(v) {↓
6     x += 'image = ' + v + "¥n";↓
7   });↓
8  Object.prototype.__defineSetter__("caption", function(v) {↓
9     x += 'caption = ' + v + "¥n";↓
10  });↓
11 </script>↓
12 <script src="http://example.jp/json.php"></script>↓
13 </body>↓
```

結果

Firefox11.0だと問題なし



Androidの標準ブラウザでハイジャック成功

Xperia ARC(SO-01C)
Android 2.3.4 でキャプチャ

Galaxy Nexus
Android 4.0.3では取得できず

対策

- script要素からのリクエストにはレスポンスを返さないようにする
  - XMLHttpRequestからのリクエストに特別なヘッダを入れておき、JSON提供側でチェックするなど
    Jqueryの以下のヘッダを使っても良いかも
    X-Requested-With: XMLHttpRequest
  - POSTにのみ応答（個人的には好みません）
- JSONデータを、JavaScriptとして実行できない形にする
  - 1行目に for(;;) ; を置く（個人的には好みません）
  - JSONデータをJavaScriptとして実行できない形にする（同上）
- JSONハイジャック可能なブラウザを使わない（利用者側でとれる対策）

まとめ

- HTML5の話題が盛り上がる中、HTML4のAjaxのセキュリティが当たり前になっていない
- 分かっている人と分かってない人の二極化が進んでいる?
- どうすればよいか、よく考えよう
- CSRF等もふつーに発生するので、ふつーに対策すること

ドリランド カード増殖祭りはこうしておこった…かも?

https://help.gree.jp/app/answers/detail/a_id/3231 より引用

こんな感じだった?

必要なもの 携帯もしくはPCを二台 gleeアカウント二つ
二台の機器でそれぞれのgleeアカウントでログインしてドリランド起動後お互いでトレードさせる
トレード(受け取りはしない)が終わったら片方の機器はログアウトして二つの機器のアカウントを同じにする
それぞれトレード品受け取り画面にして受け取るボタン同時押し

カード毎にID振って無いのかよ。。。。
馬鹿じゃねぇの？
>>277
ちゃんとふってあるけど、トレード時にトレードされる側とする側でちゃんとアイテムIDが変わる仕様だったw

http://jin115.com/archives/51849925.html より引用

こんな感じだった?

- DBからトレード元のカードのデータを読み取る
- 新しいIDをつけてトレード先に保存する
- トレード元のデータを削除する

http://bakera.jp/ebi/topic/4722 より引用

作ってみた

```
// トレードするカードのidと新オーナーのidがパラメータ
$item_id = (int)$_GET['item_id'];
$newowner = (int)$_GET['newowner'];
// カード情報を取得
$stmt = $pdo->query(
  "SELECT * FROM cards WHERE item_id=$item_id");
$row = $stmt->fetch(PDO::FETCH_ASSOC);
$kind = $row['kind'];
$owner_id = $row['owner_id'];
// トレード後のカードを作成
$stmt = $pdo->exec("INSERT INTO cards VALUES (NULL, $kind,".
  "$newowner, $item_id, $owner_id, NOW())");
// 元のカードを削除
$stmt = $pdo->exec(
  "DELETE FROM cards WHERE item_id= $item_id ");
```

※デモスクリプトはプレースホルダをちゃんと使っています!

DEMO

中ではこうなった

時刻	リクエストA	リクエストB	DB
0			
1	SELECT * FROM cards …		52 | 1 | – | –
2		SELECT * FROM cards …	
3	INSERT INTO cards …		52 | 1 | – | – 53 | 2 | 52 | 1
4		INSERT INTO cards …	52 | 1 | – | – 53 | 2 | 52 | 1 54 | 3 | 52 | 1
5	DELETE FROM cards …		53 | 2 | 52 | 1
6		DELETE FROM cards …	54 | 3 | 52 | 1

対策

- トランザクションを使う
  - PDOの場合は、begeinTransaction ～ commit / rollbackの間
- ロック（排他制御）
  - SELECT … FOR UPDATE など
- 上記の両方が必須
  - トランザクションを使うだけでは排他制御にならない場合が多い
    厳密にはDB依存 および モードに依存
- 「クリティカルセクション」に注意
  - このスクリプトの場合は、SELECT ～ DELETE FROMまではクリティカルセクションであり、同一カードIDについては並行動作してはいけない
- ちゃんとRDBを基礎から勉強しよう

トランザクションと行ロックで対策

```
try {
 $pdo->beginTransaction();
 $item_id = (int)$_GET['item_id'];
 $stmt = $pdo->query(
   "SELECT * FROM cards WHERE item_id=$item_id FOR UPDATE");
 $row = $stmt->fetch(PDO::FETCH_ASSOC);
 if (! $row) throw new DataNotFoundException();
 $kind = $row['kind'];
 $owner_id = $row['owner_id'];
 $stmt = $pdo->exec("INSERT INTO cards VALUES (NULL, $kind,".
  "$newowner, $item_id, $owner_id, NOW())");
 $stmt = $pdo->exec(
  "DELETE FROM cards WHERE item_id=$item_id");
 $pdo->commit();
} catch (DataNotFoundException $e) {
 $pdo->rollback();
} catch (Exception $e) {
 // ……
```

※上記は概要でありデモスクリプトはプレースホルダをちゃんと使っています!

中ではこうなる

時刻	リクエストA	リクエストB	DB
0			52 | 1 | – | –
1	beginTransaction		
2		beginTransaction	
3	SELECT * FROM cards …		
4		SELECT * FROM cards …	
5	INSERT INTO cards …	行ロックにより待ち状態 … …	52 | 1 | – | – 53 | 2 | 52 | 1
6	DELETE FROM cards …		53 | 2 | 52 | 1
7	commit	ロック解除されSELECTが実行されるが、0件ヒット	
8		rollback	

トランザクションを難しくする要因

- NoSQL
- Memcached
- データベースのパーティショニング、シャーディング
- データベースのレプリケーション
- LDAP、ファイル等トランザクション非対応のストレージ
- …

- 完全なロールバックが難しい場合でも、排他制御だけなら比較的容易に実装できる場合がある

まとめ

- 徳丸本に載っていない脆弱性をいくつか紹介しました
- 「徳丸本に載っていることだけやっていればおk」ではない
- プログラミングの基礎（並行処理、トランザクション、etc）を勉強しよう
- そうは言っても、まずは「徳丸本」のマスターから始めよう（宣伝）

Thank you!