HACK #55 コンテンツセキュリティポリシー

Firefox 4以降では新時代のセキュリティポリシー機能が実装されています。

HTMLでは<script>、<iframe>、<img>などの要素で外部サイトのコンテンツ埋め込みが許可されています。JavaScriptはインラインスクリプトはもちろん、onclickなどのイベント属性でも簡単に実行できます。

これは便利ではありますが、Ajaxなどに対する同一生成元ポリシーしか制限がないことで、クロスサイトスクリプティング（Cross Site Scripting：XSS）などの攻撃が容易になっているという側面もあります。そこで、Firefox 4からは新時代のセキュリティポリシーとして、**コンテンツセキュリティポリシー**（Content Security Policy：CSP）が導入されました†。

CSPの基本的な使い方

CSPのポリシーはX-Content-Security-Policy HTTPレスポンスヘッダで指定します。ブラウザはこのヘッダ付きで送信されてきたページに対してのみCSPによるセキュリティ制限を適用します。

```
X-Content-Security-Policy:ポリシー
```

CSPのポリシーでは、対象とするファイルの種類を指定する「ディレクティブ」と、許可されるドメインやプロトコル、ポートなどを指定する「ソース」のペアを記述します。ディレクティブはセミコロンで区切って、いくつでも記述できます。

```
ディレクティブ ソース [;ディレクティブ ソース]+
```

それでは基本的なCSPヘッダの例を見てみましょう。次のポリシーでは、すべてのコンテンツについてページと同じドメインからのみ読み込みます。

```
X-Content-Security-Policy: default-src 'self'
```

default-srcディレクティブではその名のとおり、他のディレクティブで指定されなかったファイルを読み込むソースを指定します。'self'キーワードは現在のページと同一ドメイン、同一ポートを表します。

---

† 本稿執筆時点でCSPはW3C非公式草案段階の仕様であり、今後変更の可能性がありますが、ここではFirefox 4での実装と、2011年6月時点で最新のCSP仕様（https://dvcs.w3.org/hg/content-security-policy/raw-file/tip/csp-specification.dev.html）を元に解説します††。

†† 本稿執筆時点でWebKitでの実装もすでに進められており、Twitterなどでも採用されています。遠からずセキュリティポリシーの新常識となると筆者は予想・期待しています。

このディレクティブは仕様策定当初allowという名称でした。Firefox 5からはallowとdefault-srcが同様に扱われますが、Firefox 4ではallowディレクティブを指定する必要があります。他のブラウザとの互換性のためにはCSSのベンダー接頭辞同様に新旧両方を併記してください。サポートされないディレクティブは単に無視されます。

```
X-Content-Security-Policy: default-src 'self'; allow 'self'
```

次のポリシーでは、ページと同じドメインとdynamis.jpおよびそのサブドメインのコンテンツだけを読み込みます。

```
X-Content-Security-Policy: default-src 'self' *.dynamis.jp
```

img-src、object-src、script-srcなどのディレクティブでは、特定種別のファイルを読み込むドメインを指定します。

次の例（実際には1行）では、画像は任意のドメインから、Flashなどのプラグインコンテンツはstatic.slideshare.netから、スクリプトファイルはsafescript.jpから、その他のファイルはページと同じドメインからのみ読み込みます。

```
X-Content-Security-Policy: default-src 'self'; img-src *;
                           object-src static.slideshare.net;
                           script-src safescript.jp
```

X-Content-Security-Policyヘッダは同じHTTPレスポンスの中で複数含めることもできます。同時に複数のセキュリティポリシーが送信された場合、すべてのポリシーで許可されているコンテンツだけが読み込まれます。

例えば次の2つのヘッダが送信された場合、ブラウザはスクリプトと<audio>、<video>のメディアファイルは自身のサイトからのみ、他のファイルは任意のサイトから読み込みます。

```
X-Content-Security-Policy: default-src *; script-src 'self'
X-Content-Security-Policy: default-src *; script-src 'self';
                           media-src 'self'
```

これは例えばサイト全体に適用するセキュリティポリシーをApacheのhttpd.confで次のように常に出力させているが、一部のページではさらに制限を加えたいというような場合に便利です。

```
Header always append X-Content-Security-Policy \
                     "default-src *; script-src 'self'"
```

銀行などセキュリティ第一のサイトでは、サイトの内外にかかわらずすべてのコンテンツの読み込みをHTTPS接続に限定したくなります。このような場合はドメイン名で

なくプロトコルとポートを制限することもできます。

```
X-Content-Security-Policy: default-src https://*:443
```

このようなポリシーを使用してもHSTS（詳しくは**[Hack #56]**を参照）が不要になるわけではないことに注意してください。CSPは個別のページに読み込むコンテンツを制限するもので、新しいページの読み込みをHTTPSに限定するものではないからです。

デフォルトセキュリティポリシー

Webアプリケーションにおいて最も有名なセキュリティ問題でありながら、10年以上も抜本的な対策が導入されてこなかった、クロスサイトスクリプティングを解決することがCSP最大の目的です。

そのためCSPでは、HTML中に画像などを`data:` URLで埋め込んだり、スクリプトやスタイルをインラインで記述することを一切禁止します。それらはすべて外部ファイルに分離して、ポリシーで明示的に許可したサイトから読み込む必要があります。

このように制限することで、例え攻撃者がHTMLの中に自由な文字列を埋め込むことができても、サーバのファイルを書き換えることができない限り、任意のスクリプトやコンテンツを読み込ませることはできなくなります。

それでは具体的に、デフォルトポリシーで無効化されている機能を確認し、それにどう対応すべきか確認していきましょう。

data: URL

`data:` URLはデフォルトポリシーで禁止されます。先ほど最後の例ではコンテンツを読み込むソースとして443番ポートの`https:`というプロトコルを指定していましたが、同様にプロトコル`data:`を明示的に許可対象として指定することで、`data:` URLが使えるようになります。

例えば、画像ファイルだけは`data:` URLで埋め込む許可をするには次のようなポリシーになるでしょう。

```
X-Content-Security-Policy: default-src 'self'; img-src data:
```

インラインスクリプトは実行されない

次のようなところでHTML中に直接スクリプトを記述することは禁止されます。デフォルトポリシーでは`<script src="...">`要素で読み込む外部スクリプトファイルに限られます†。

† Firefox 4ではまだ制限されていませんが、仕様によるとスクリプトファイルの`Content-Type`は`application/javascript`や`application/json`に制限されます。

- javascript: URL
- onclickなどのイベント属性
- <script>要素内のインラインスクリプト

イベント属性については属性として記述するのではなく、addEventListener()でイベントリスナを登録するか、スクリプトからイベントハンドラ属性を設定してください。

```
element.addEventListener("click", someFunction, false);
element.onclick = someFunction;
```

インラインスクリプトの制限を解除するには、script-srcディレクティブで'unsafe-inline'キーワードを指定します。ただし、このキーワードは最近仕様に追加されたもので、執筆時点のFirefoxではまだサポートしていません†。

```
X-Content-Security-Policy: default-src 'self';
                           script-src 'self' 'unsafe-inline';
```

文字列はスクリプトとして評価できない

外部スクリプトであっても、eval()など文字列をスクリプトとして評価する処理は禁止されます††。

- eval()
- new Function()コンストラクタ
- 文字列引数でのsetTimeout()
- 文字列引数でのsetInterval()

文字列をスクリプトとして評価できない制限を解除するには、script-srcディレクティブで'unsafe-eval'キーワードを指定します。ただし、執筆時点のFirefoxではまだ'unsafe-eval'キーワードはサポートされていません。

```
X-Content-Security-Policy: default-src 'self';
                           script-src 'self' 'unsafe-eval';
```

インラインスタイルは適用されない

スクリプトだけでなくCSSスタイルについてもインラインで書かれたものは適用されません。デフォルトポリシーでは<link rel="stylesheet" href="...">要素

† 執筆時点でもまだCSPのデフォルトセキュリティポリシーに関しては議論が続いており、将来変更される可能性があります。

†† Firefox 4の時点ではCSPによりeval()が禁止されている場合でもWorker内では実行できてしまいます（https://bugzilla.mozilla.org/show_bug.cgi?id=609748）。

で読み込む外部スタイルファイルに限られます。

ただし、インラインスタイルの制限は最近仕様に追加されたものであり、執筆時点のFirefoxではまだこの制限は実装されておりません。

- <style>要素
- style属性

インラインスタイルについてもインラインスクリプトと同様に'unsafe-inline'キーワードで制限解除できるようになる予定です。

```
X-Content-Security-Policy: default-src 'self';
                           style-src 'self' 'unsafe-inline';
```

CSPのディレクティブ

執筆時点の最新仕様で定義されている主なCSPディレクティブと、ソースに指定できるキーワードを**表55-1**と**表55-2**にまとめました。

これらのディレクティブは直接HTTPヘッダに出力する以外に、policy-uriディレクティブを使ってポリシーファイルを読み込ませることもできます。なお、ポリシーファイル中のコメント行はFirefox 8時点ではまだサポートされていません。

各ディレクティブについてさらに詳しくは仕様書やMDNのページを参照してください。

- https://dvcs.w3.org/hg/content-security-policy/raw-file/tip/csp-specification.dev.html#directives
- https://developer.mozilla.org/en/Security/CSP/CSP_policy_directives

表55-1 ポリシーディレクティブ一覧

ディレクティブ名	ディレクティブの説明
default-src	特定ファイル用のディレクティブで指定されなかったファイルの読み込みを許可するソースを指定する
script-src	<script>要素で読み込み許可する、外部スクリプトファイルのソースを指定する
object-src	<object>、<embed>、<applet>要素で読み込み許可する、プラグインなどの外部オブジェクトファイルのソースを指定する
img-src	<img>、<link rel="icon">、<link rel="shortcut icon">要素、CSSのurl()やimage()プロパティで読み込み許可する、画像ファイルのソースを指定する。data: URLで画像を埋め込む場合はimg-src data:を指定する
media-src	<audio>、<video>要素で読み込み許可する、メディアファイルのソースを指定する

ディレクティブ名	ディレクティブの説明
style-src	<link rel="stylesheet">要素で読み込み許可する、外部スタイルシートファイルのソースを指定する
frame-src	<iframe>要素で読み込み許可する、HTMLページのソースを指定する
font-src	CSSの@font-face規則で読み込み許可する、ダウンロードフォンのソースを指定する
xhr-src	XMLHttpRequestで読み込み許可する、外部オブジェクトのソースを指定する
frame-encestors	<frame>、<iframe>要素で現在のページを埋め込むことを許可する、祖先フレームページのソースを指定する*。
policy-uri	ポリシーをHTTPレスポンスヘッダに直接記述せず、指定したURL**からダウンロードする。指定可能なURLはページと同じドメインに限られ、そのContent-Typeはtext/x-content-security-policyとして送信される必要がある
report-uri	ポリシー違反が生じたときにブラウザがレポートを送信するURL**を指定する。指定可能なURLはページと同じドメインに限られる
options	CSPの処理モデルを変更するオプションを指定する。ただし執筆時点の最新仕様では指定するオプションが定義されていない***

* frame-encestorsディレクティブはクリックジャッキング対策の機能であり、CSRF対策には有効でないことに注意してください。ブラウザ側で現在ページの読み込みを禁止するものであり、サーバがリクエストを受け付けないようになるものではないからです。

** policy-uriやreport-uriのURLではリダイレクトは禁止されます。Firefox 4～8の時点では302のリダイレクトを許可してしまっていますが、将来修正予定です。また、Firefox 4ではこれらのアクセスがWebコンソールに表示されないので注意してください。

*** 古い仕様ではoptionsでインラインスクリプト制限などを解除するオプションが定義されていました。

表55-2　ソースに指定するキーワード一覧

キーワード名	キーワードの説明
'self'	CSPで保護する対象である現ページと同じドメインを表す
'none'	どのドメインのホストにもマッチしない
'unsafe-inline'	script-srcやstyle-srcディレクティブで指定すると、インラインスクリプトやインラインスタイルの使用を許可できる
'unsafe-eval'	script-srcディレクティブで指定すると、eval()などの使用を許可できる

ポリシー違反レポートを受け取る

CSPで指定したセキュリティポリシーに違反したコンテンツは、ブラウザに読み込まれないだけでなく、違反の内容を記述したレポート（violation report）を指定のURLに送信させることができます。

違反レポートは開発時のデバッグに便利なだけでなく、本番運用のサイトにおいても攻撃の検出やセキュリティホールの素早い特定に役立ちます。セキュリティホールを突いてスクリプトインジェクションが行われたがCSPによりユーザーを保護できた場合などに、ブラウザから指定のURLへとレポートが送信されサイト管理者がすぐに知ることができるからです。

違反レポートには該当ページとそのリクエストヘッダ、ブロックされたURLや違反対象となったポリシーなどをJSON形式で記述したものになります。

例えばで次のポリシーを設定したページで外部ドメインの画像ファイルを読み込もうとした場合、

```
X-Content-Security-Policy: allow 'none'; img-src 'self'
```

送信されるレポートは次のようなものになります。

```
{
  "csp-report":
    {
      "request": "GET http://index.html HTTP/1.1",
      "request-headers": "Host: example.jp
        User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:2.0)
Gecko/20100101 Firefox/4.0
        Accept: text/html,application/xhtml+xml,application/
xml;q=0.9,*/*;q=0.8
        Accept-Language: ja,en-us;q=0.7,en;q=0.3
        Accept-Encoding: gzip,deflate
        Accept-Charset: Shift_JIS,utf-8;q=0.7,*;q=0.7
        Keep-Alive: 115
        Connection: keep-alive",
      "blocked-uri": "http://evilsite.jp/lisa_et_gaspard.png",
      "violated-directive": "img-src 'self'",
      "original-policy": "allow 'none'; img-src 'self'"
    }
}
```

ポリシー違反のレポートは受け取るが、実際にブラウザでコンテンツをブロックしないようブラウザに指示することもできます。既存のサイトに適したポリシーを決めたり、開発途中の段階ではX-Content-Security-Policyの代わりにX-Content-Security-Policy-Report-Onlyヘッダにポリシーを記述してください。

```
X-Content-Security-Policy-Report-Only: allow 'none'; img-src 'self'
```

既存サイトをCSP対応に移行させる

CSPではインラインスクリプトやeval()関数の使用を禁止するなど、従来に比べると大幅にXSSに対して安全になると同時に、これらの処理を利用したコードが動作しなくなります。

既存のサイトをCSPに対応させる場合、基本的にはX-Content-Security-Policy-Report-Onlyでポシリーを設定した状態でサイトをブラウズして、Firefoxのエラーコンソールに表示されるエラーや、report-uriに送信される違反レポートを見ながらコードを修正していくことになります。

しかし、ゼロからポリシーを記述するのは手間がかかります。そのためFirefoxのCSP対応を行ったBrandon Sterneはベースとなるポリシーを提案するブックマークレットを公開しています。

> http://brandon.sternefamily.net/posts/2011/01/update-to-csp-bookmarklet/

このブックマークレットでは、現在のページに読み込まれている外部ファイルやインラインスクリプトを検出し、そのページ用のセキュリティポリシーと、ポリシー違反のインラインスクリプト一覧を出力します（**図55-1**）。

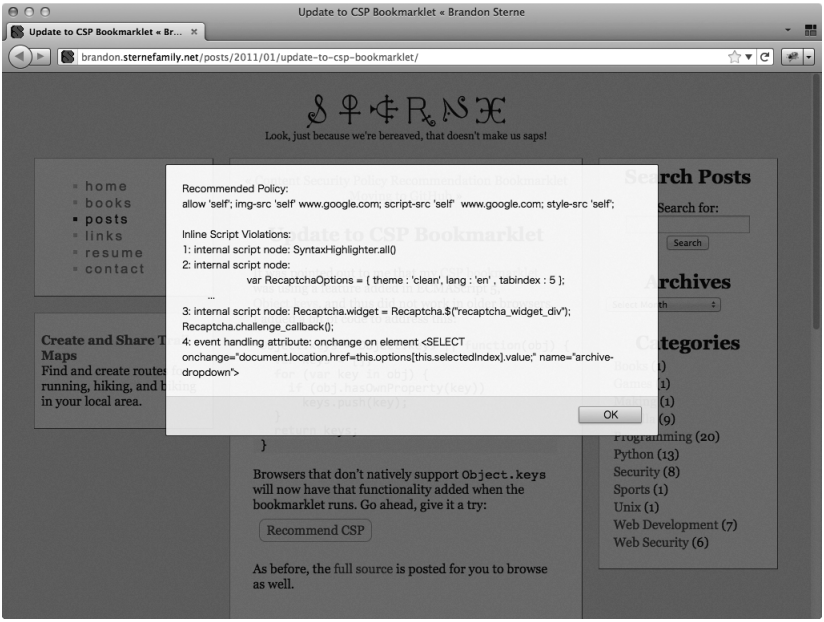

図55-1　CSPブックマークレット使用例

CSP対応させたいサイトでまずはこのブックマークレットを使い、提案されたポリシーをベースとして調整していくとよいでしょう。

また、ライブラリについては古いバージョンがCSPに対応していない場合が見受けられます。例えばjQueryについては1.4.3以前のバージョンでは読み込み時に`eval()`テストが実行されるためCSP違反エラーが発生します。この問題が修正されている1.4.4以降のバージョンを使うようにしてください。

既存のシステムをCSPに対応させる

WordPressなどの主要なCMSやフレームワークには、CSP対応するためのプラグインなどが公開されているものがあります。代表的なものの配布元を紹介しますので、必要に応じて使ってみてください。

- WordPress: http://wordpress.org/extend/plugins/content-security-policy/
- Django: https://github.com/mozilla/django-csp
- Drupal: http://drupal.org/project/content_security_policy

— Tomoya Asai

HACK #56 イマドキのセキュリティ機能を活用する

安全なサイトを作るため、これだけは知っておきたいイマドキのセキュリティ機能について解説します。

Webは性善説のもとで生まれたシステムです。HTTPの通信内容は平文のまま中継され、HTMLページにおいても<img>、<iframe>、<script>などの要素では別サイトのコンテンツもページに埋め込めます。

もちろん新しいWeb標準技術では常にセキュリティが確保できるよう設計されます。SSLによる暗号化通信も可能ですし、セッション情報を保存するCookieの読み取りは他のドメインからはできず、XMLHttpRequestなどには同一生成元ポリシーによって制限されています。

しかしそれでもセッションハイジャック、クリックジャッキング、クロスサイトスクリプティングなどの問題がなくなることはありません。本稿では、それらのセキュリティ問題を解決するために導入された、比較的新しいセキュリティ機能を解説します。

HTTP Strict Transport-Securityによるセッションハイジャック対策（Firefox 4〜）

Webサイトでユーザー認証を行いセッションを維持する場合には通常Cookieが使われます。しかし、通信がHTTPによる平文で行われていると、容易にセッションを乗っ取られてしまいます。特にPublic WiFiでは他のユーザーの通信を簡単に傍受できる場合が多く、深刻な問題となります†。

攻撃者による盗聴を防止するには、常にHTTPSで暗号化通信を行う必要があります。これを実現するのがHSTS（HTTP Strict Transport-Security）と呼ばれるHTTPS

† 初心者でも簡単にセッションハイジャックできるツール（http://codebutler.github.com/firesheep/）まで公開されています。