Web Security and its rabbitholes

Fundamentals

What is web security all about?

SameOrigin policy

- Origin
    - scheme+host+port
- Realm
- Window

RFC 6454

5. Comparing Origins

Two origins are "the same" if, and only if, they are identical. In particular:

- o If the two origins are scheme/host/port triples, the two origins are the same if, and only if, they have identical schemes, hosts, and ports.
- o An origin that is a globally unique identifier cannot be the same as an origin that is a scheme/host/port triple.

Two URIs are same-origin if their origins are the same.

NOTE: A URI is not necessarily same-origin with itself. For example, a data URI [RFC2397] is not same-origin with itself because data URIs do not use a server-based naming authority and therefore have globally unique identifiers as origins.

Deep dive: iframe & Realm

- same origin iframe

```
window.top.globalFunction();
window.top.location.reload();
```

- cross origin iframe

```
Uncaught DOMException: Permission denied to access property "reload" on cross-origin object
```

- identity discontinuity

```
window.top.Array.prototype !== window.Array.prototype
```

Deep dive: iframe sandbox

```
<iframe sandbox="" src="...">
```

- unique origin on the document
- unique origins on the resources (scripts)

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe

The attribute values are used to **LOOSEN** the sandbox security

Engines have to do horrible things to prevent leaking information cross-origin.

May be surprising:
http://naugtur.pl/rejection-in-iframe-sandbox/

Enforcement is difficult and has lots of edge cases
https://bugs.chromium.org/p/chromium/issues/detail?id=103630

Deep dive: CORS

> “ CORS: when you need a SameOrigin Policy bypass for a feature ”

- history: using flash for cross-origin requests
- not a security mechanism
- simple requests - what HTML is capable of
- preflight for others

no-preflight requests are made anyway, just no reading the response

Headers

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Origin: https://naugtur.pl
Access-Control-Allow-Origin: ${req.headers.origin}
```

```
Access-Control-Request-Method: POST
Access-Control-Request-Headers: Authorization, Content-Type

Access-Control-Allow-Origin: https://naugtur.pl
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: Authorization, Content-Type
Access-Control-Max-Age: 7200
```

browsers have a cap on Access-Control-Max-Age

> “ Firefox caps this at 24 hours (86400 seconds). Chromium (prior to v76) caps at 10 minutes (600 seconds). Chromium (starting in v76) caps at 2 hours (7200 seconds). The default value is 5 seconds. ”

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Max-Age

Cookies and sessions

- attributes https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/06-Session_Management_Testing/02-Testing_for_Cookies_Attributes
- with HttpOnly - the only webapp storage inaccessible to XSS

Session

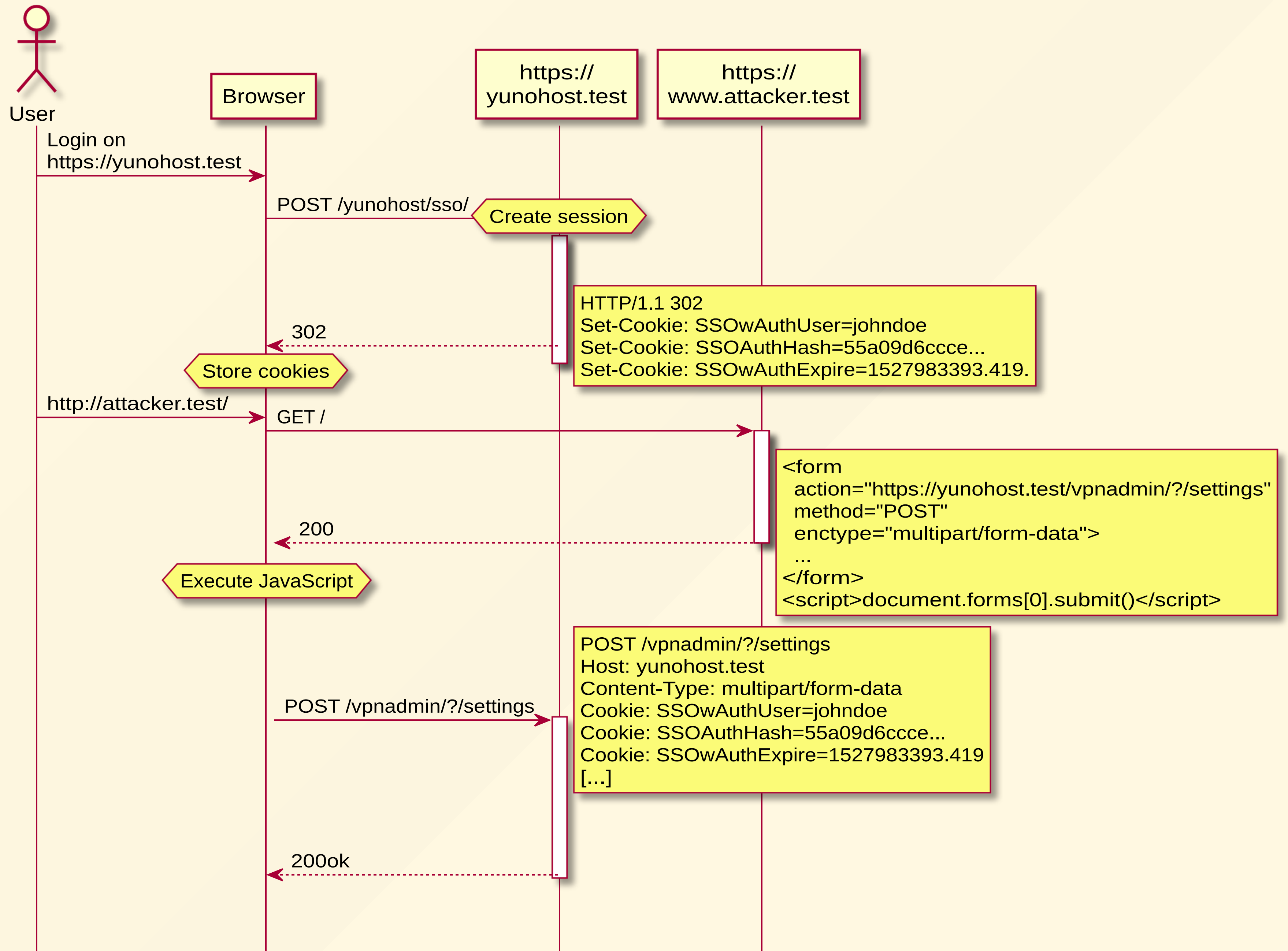
- stateful or stateless?
  - cookie session vs JWT in the app
    - stealer malware
    - xss
    - JWT exploits
  - session pinning (IP or https)
- CSRF

Historical background: websites interop

links and forms spec came before security was even needed.

Deep dive: CSRF

https://portswigger.net/web-security/csrf/lab-no-defenses



```
HTTP/1.1 302
Set-Cookie: SSOwAuthUser=johndoe
Set-Cookie: SSOAuthHash=55a09d6ccce...
Set-Cookie: SSOwAuthExpire=1527983393.419.
```

```
<form
  action="https://yunohost.test/vpnadmin/?/settings"
  method="POST"
  enctype="multipart/form-data">
  ...
</form>
<script>document.forms[0].submit()</script>
```

```
POST /vpnadmin/?/settings
Host: yunohost.test
Content-Type: multipart/form-data
Cookie: SSOwAuthUser=johndoe
Cookie: SSOAuthHash=55a09d6ccce...
Cookie: SSOwAuthExpire=1527983393.419
[...]
```

They say don't roll your own CSRF, but then again...
https://portswigger.net/daily-swig/csrf-flaw-in-csurf-npm-package-aimed-at-protecting-against-the-same-flaws
https://fortbridge.co.uk/research/a-csrf-vulnerability-in-the-popular-csurf-package/?trk=public_post_comment-text

- Double Submit Cookie Pattern is difficult to implement
- Snyk removed the post and warnings - only insecure when misused?

XSS

- Reflected XSS - where the malicious script comes from the current HTTP request.
- Stored XSS - where the malicious script comes from the website's database.
- DOM-based XSS - where the vulnerability exists in client-side code rather than server-side code.

https://portswigger.net/web-security/cross-site-scripting

Exercise

https://xss-game.appspot.com

```
<img src=a onerror=alert() >
```

spoilers

```
1' onerror=alert() a='
',alert(),'
next=javascript:alert()
data:@file/javascript;base64,YWxlcnQoKQo=
```

Mitigations

- Filtering
  - find-replace denylist based will never work
  - allowlist is better
- DomPurify
  - Nothing is perfect - mb's parsers exploit
- CSP - assume XSS ulns exist and focus on stopping them

CSP and how to roll out

- core knowledge
  https://content-security-policy.com/
  https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP
- level 1 2 3
  - level 3 support is incomplete

Basic directives

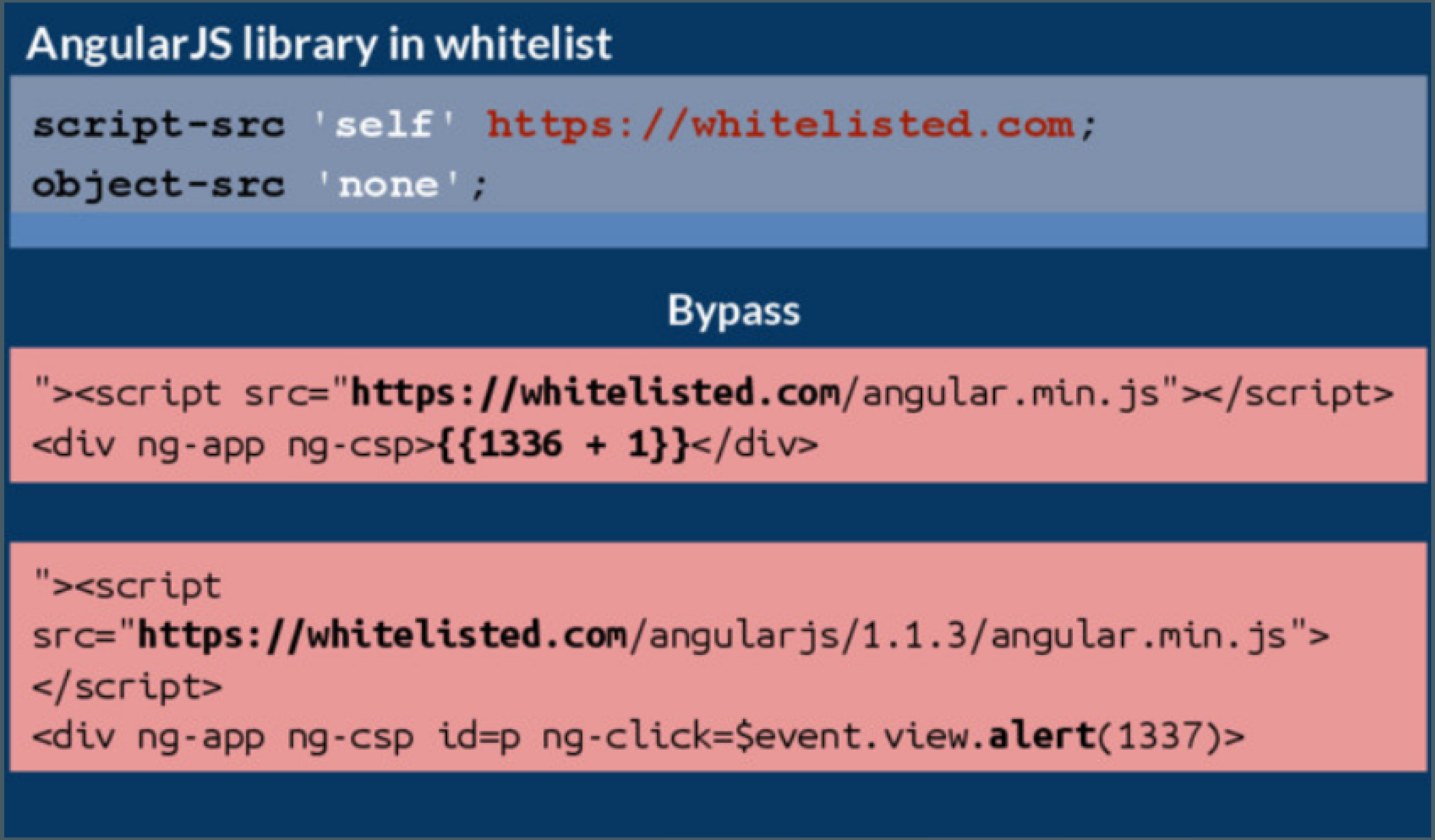
default-src
script-src
*-src

sandbox

frame-ancestors

base-uri

XSS CSP bypass



AngularJS library in whitelist

```
script-src 'self' https://whitelisted.com;
object-src 'none';
```

Bypass

```
"><script src="https://whitelisted.com/angular.min.js"></script>
<div ng-app ng-csp>{{1336 + 1}}</div>
```

```
"><script
src="https://whitelisted.com/angularjs/1.1.3/angular.min.js">
</script>
<div ng-app ng-csp id=p ng-click=$event.view.alert(1337)>
```

BYPASSING CSP [5/5]

Path relaxation

Path relaxation due to open redirect in whitelist

```
script-src https://whitelisted.com/totally/secure.js https://site.with.redirect.com;
object-src 'none';
```

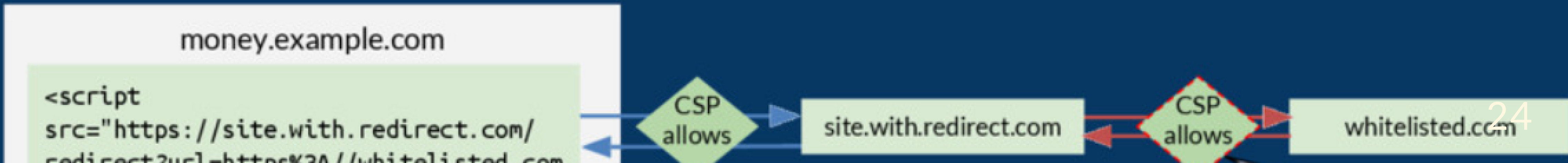
Path is ignored after redirect!

Bypass

```
~~">'><script src="https://whitelisted.com/jsonp?callback=alert">~~
```

```
">'><script src="https://site.with.redirect.com/redirect?url=https%3A//whitelisted.com/jsonp%2Fcallback%3Dalert">
```

Spec: "To avoid leaking path information cross-origin (as discussed in Homakov's Using Content-Security-Policy for Evil) the matching algorithm ignores path component of a source expression if the resource loaded is the result of a redirect."

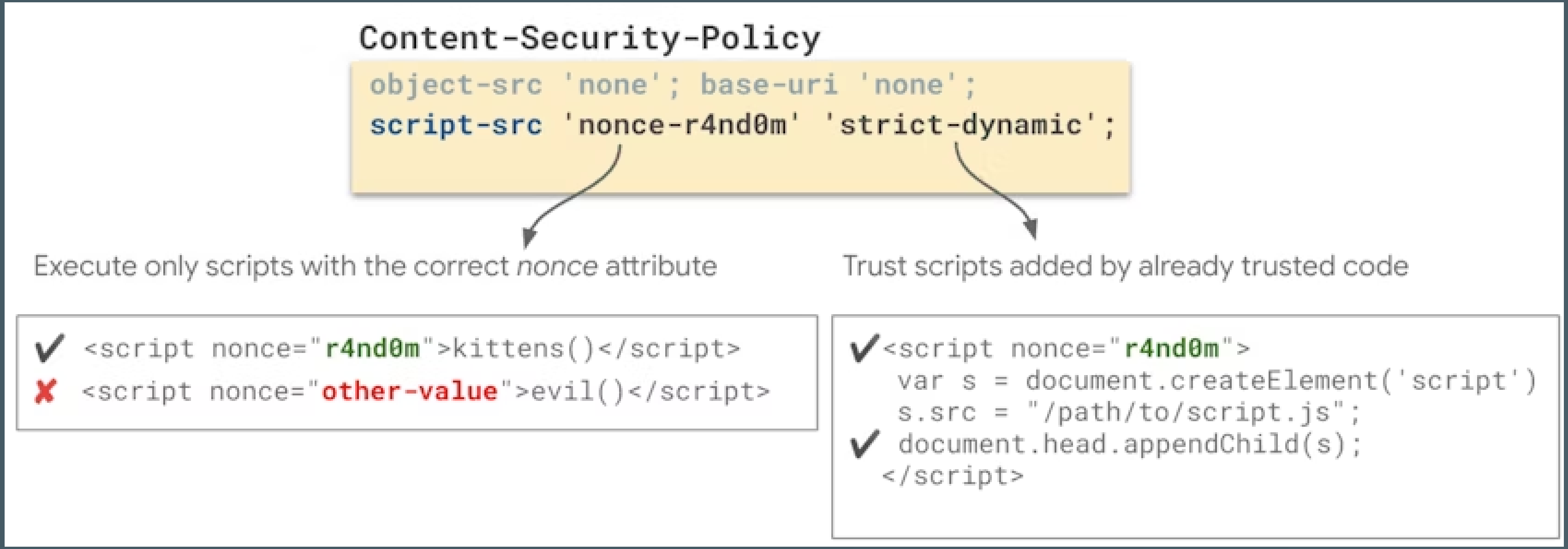


Whole deck
https://speakerdeck.com/lweichselbaum/csp-is-dead-long-live-strict-csp-deepsec-2016?slide=16

Strict CSP

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

https://web.dev/strict-csp/



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

Execute only scripts with the correct *nonce* attribute

```
✓ <script nonce="r4nd0m">kittens()</script>
✗ <script nonce="other-value">evil()</script>
```

Trust scripts added by already trusted code

```
✓<script nonce="r4nd0m">
   var s = document.createElement('script')
   s.src = "/path/to/script.js";
✓  document.head.appendChild(s);
  </script>
```

Allowing scripts

- Nonce-based CSP
- Hash-based CSP

Exercise

https://github.com/naugtur/CSP-exercise

https://portswigger.net/web-security/cross-site-scripting/contexts/client-side-template-injection/lab-angular-sandbox-escape-and-csp

Rollout

- careful with features
- careful with reporting sink
  https://github.com/naugtur/csp-report-lite

iteratie rollout algo

```
1. add your desired CSP to the app, `report-only`
2. open the app in the browser
3. observe what breaks, fix the app or loosen the policy
4. run e2e tests
5. observe what breaks, fix the app or loosen the policy
6. roll out to test enironments/stagings for a week
7. observe what breaks, fix the app or loosen the policy
8. roll out to users (or just 1-5% of them)
9. observe what breaks, fix the app or loosen the policy
10. full roll-out
11. observe what breaks, fix the app or loosen the policy
12. When no longer getting reports, remoe `report-only`
```

Deep dive: report-to vs report-uri

report-uri works.

report-to - only supported by chromium and doesn't seem to work.

- must be https, apparently
- Report-To header already deprecated and doesn't seem to work
- Reporting-Endpoints doesn't work either when I tried it

DOH! https://bugs.chromium.org/p/chromium/issues/detail?id=1152867

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/report-to

Deep dvie: trusted types

https://web.dev/trusted-types/

`trusted-types` - configure policy limitations

`require-trusted-types-for 'script'`

`TrustedHtml`

Only chromium browsers.

bad

```
el.innerHTML = '<img src=xyz.jpg>';
```

good

```
el.textContent = '';
const img = document.createElement('img');
img.src = 'xyz.jpg';
el.appendChild(img);
```

https://developer.mozilla.org/en-US/docs/Web/API/TrustedHTML

https://github.com/cure53/DOMPurify#what-about-dompurify-and-trusted-types

More reading

https://david-gilbertson.medium.com/im-harvesting-credit-card-numbers-and-passwords-from-your-site-here-s-how-9a8cb347c5b5

https://portswigger.net/research/ambushed-by-angularjs-a-hidden-csp-bypass-in-piwik-pro

Supply chain

- https://security.snyk.io/
- npm maudit etc.
- https://socket.dev
- LavaMoat

LavaMoat

https://naugtur.pl/pres3/lava/index.html

Deep dive: LavaMoat

https://github.com/naugtur/js-training-examples/tree/master/lavamoat/preparation.md