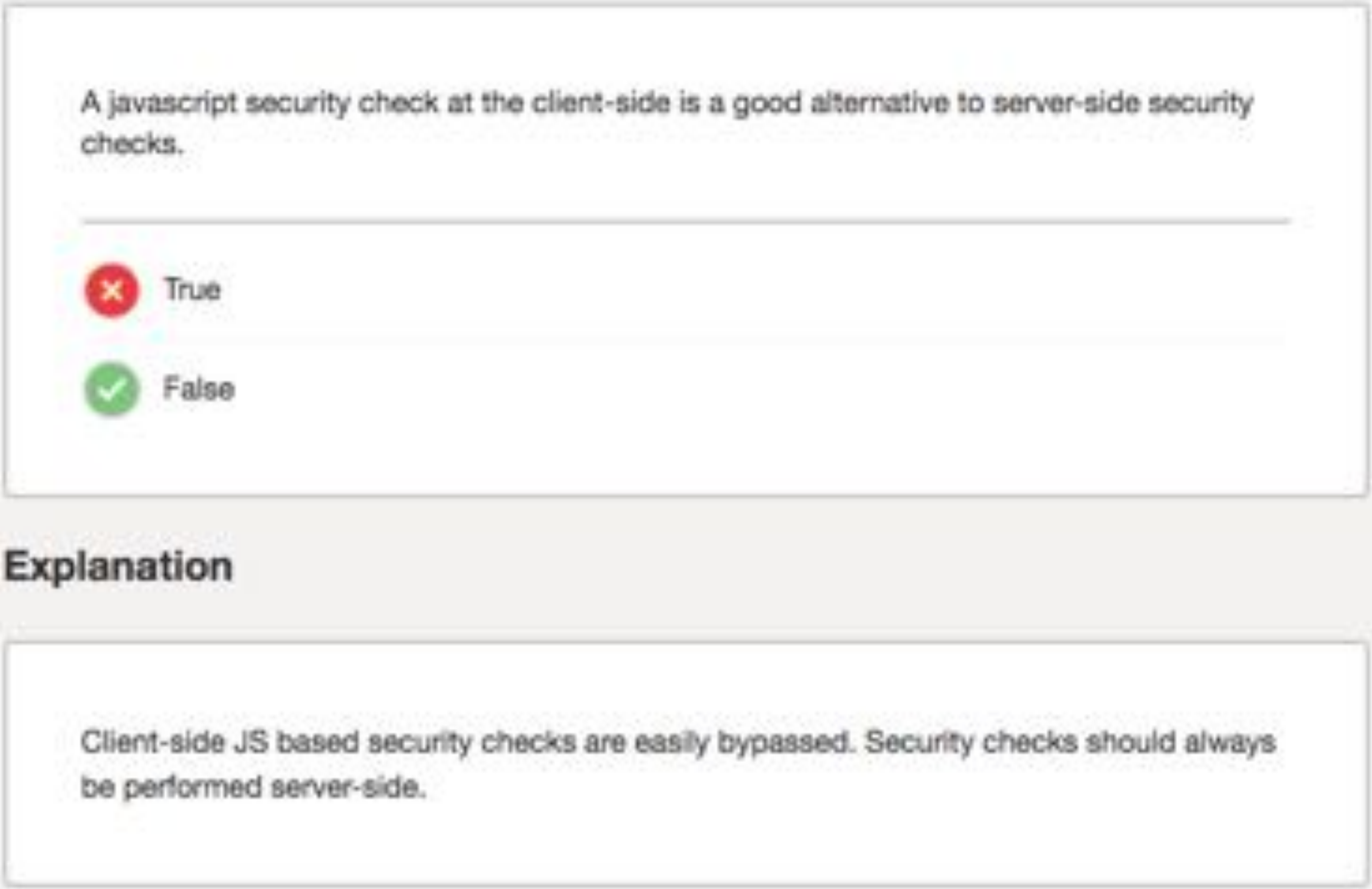
The rise and fall of client-side security policies

Philippe De Ryck

SecAppDev 2017

https://www.websec.be

@PhilippeDeRyck

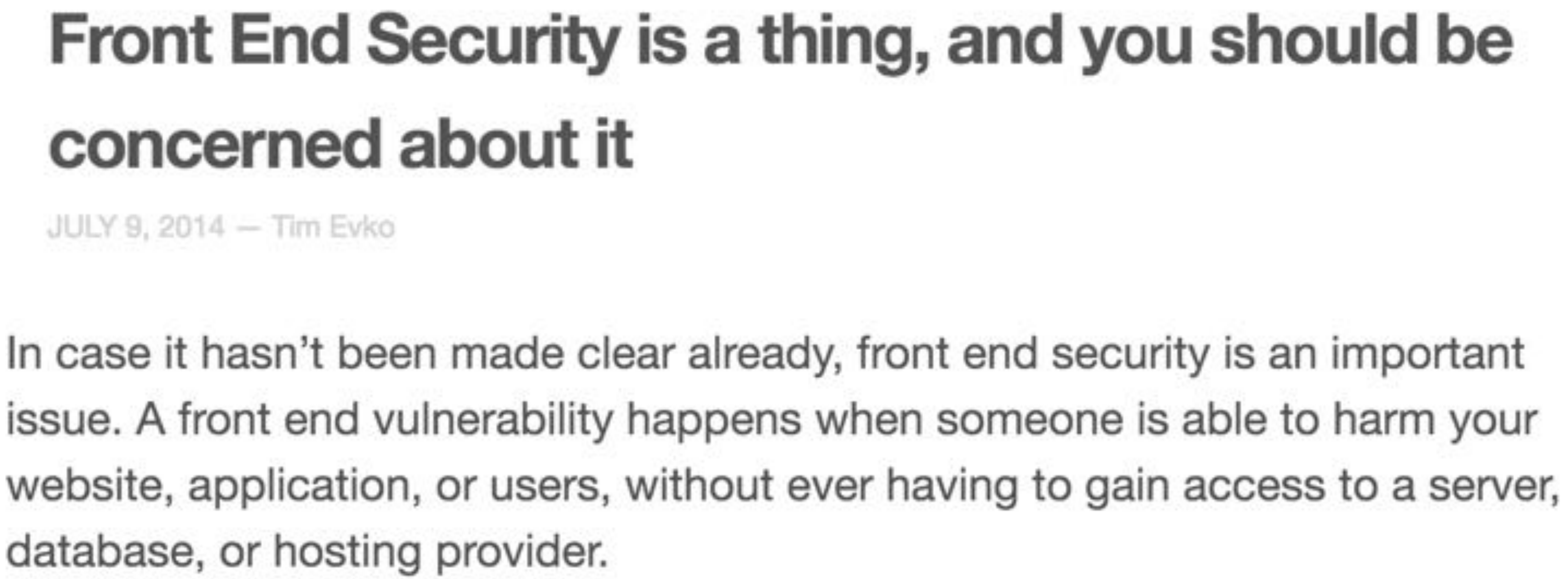


A javascript security check at the client-side is a good alternative to server-side security checks.

- ✗ True
- ✓ False

Explanation

Client-side JS based security checks are easily bypassed. Security checks should always be performed server-side.



Front End Security is a thing, and you should be concerned about it

JULY 9, 2014 — Tim Evko

In case it hasn't been made clear already, front end security is an important issue. A front end vulnerability happens when someone is able to harm your website, application, or users, without ever having to gain access to a server, database, or hosting provider.

https://web-design-weekly.com/2014/07/09/front-end-security-thing-concerned/

NEW CLIENT-SIDE SECURITY TECHNOLOGIES ARE PREVALENT

HTTP Strict Transport Security

X-Content-Type-Options

X-FRAME-OPTIONS

X-XSS-Protection

Content Security Policy

HTTP Public Key Pinning

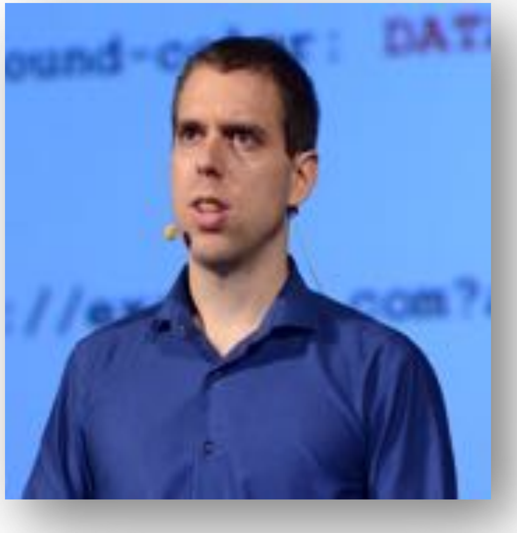
Sandbox attribute

Subresource Integrity

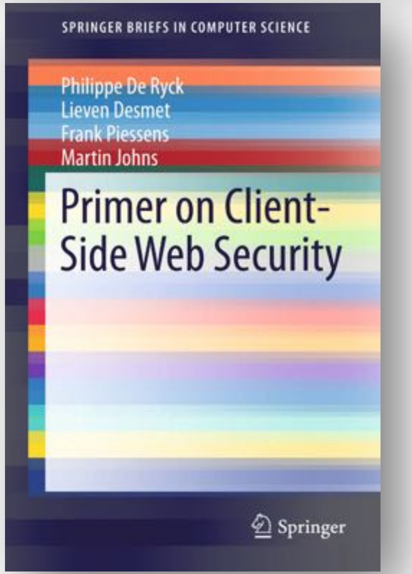
YES, LET'S DO SECURITY!

WHERE DO YOU GET STARTED?

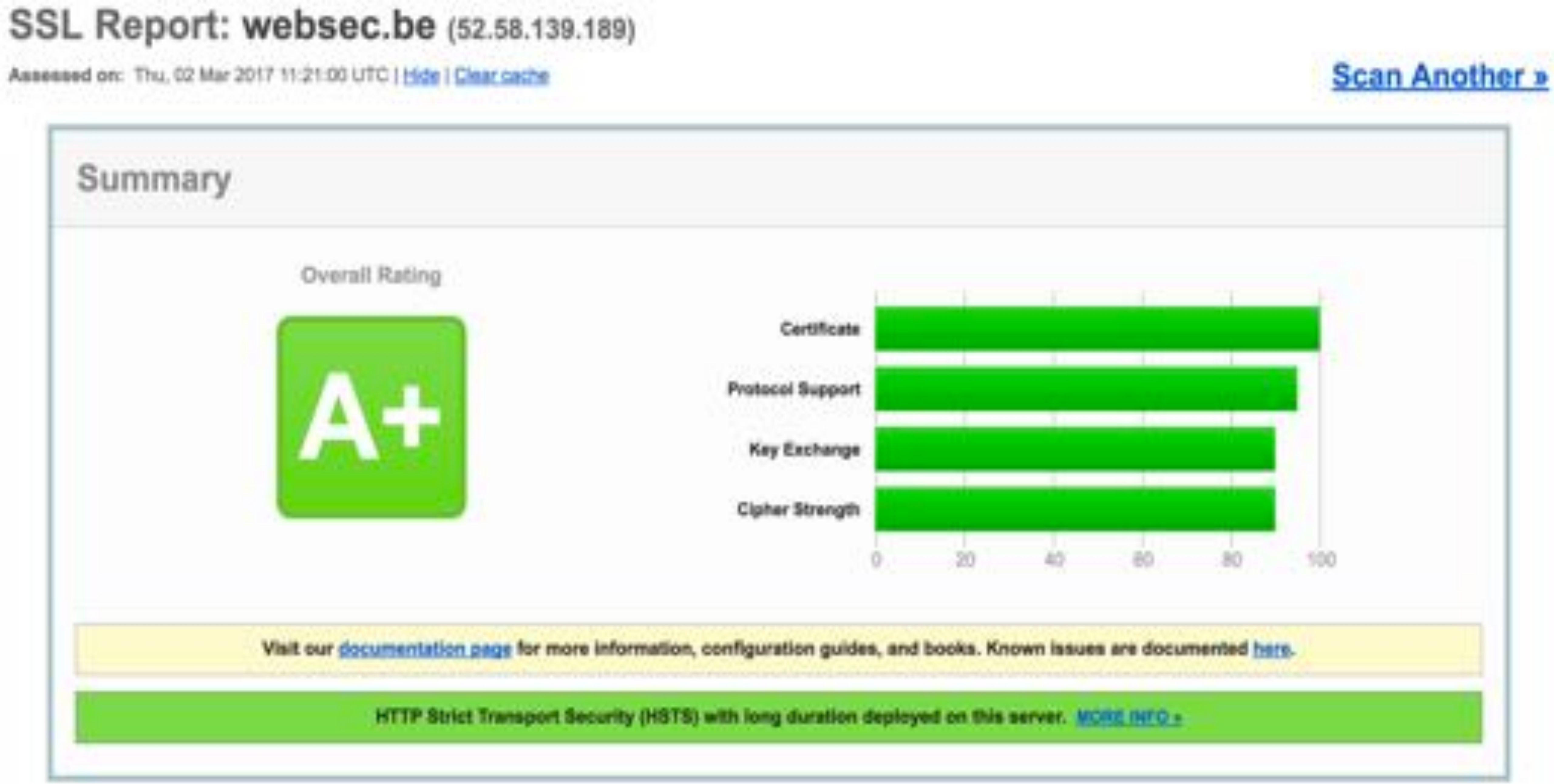
ABOUT ME – PHILIPPE DE RYCK



- My goal is to help you build secure web applications
  - Hosted and customized in-house training
  - Specialized security assessments of critical systems
  - Threat landscape analysis and prioritization of security efforts
  - More information and resources on **https://www.websec.be**


SPRINGER BRIEFS IN COMPUTER SCIENCE
Philippe De Ryck
Lieven Desmet
Frank Piessens
Martin Johns
Primer on Client-Side Web Security
Springer

- My security expertise is broad, with a focus on Web Security
  - PhD in client-side web security
  - Main author of the *Primer on client-side web security*



SSL Report: websec.be (52.58.139.189)

Assessed on: Thu, 02 Mar 2017 11:21:00 UTC | Hide | Clear cache

Scan Another »

Summary

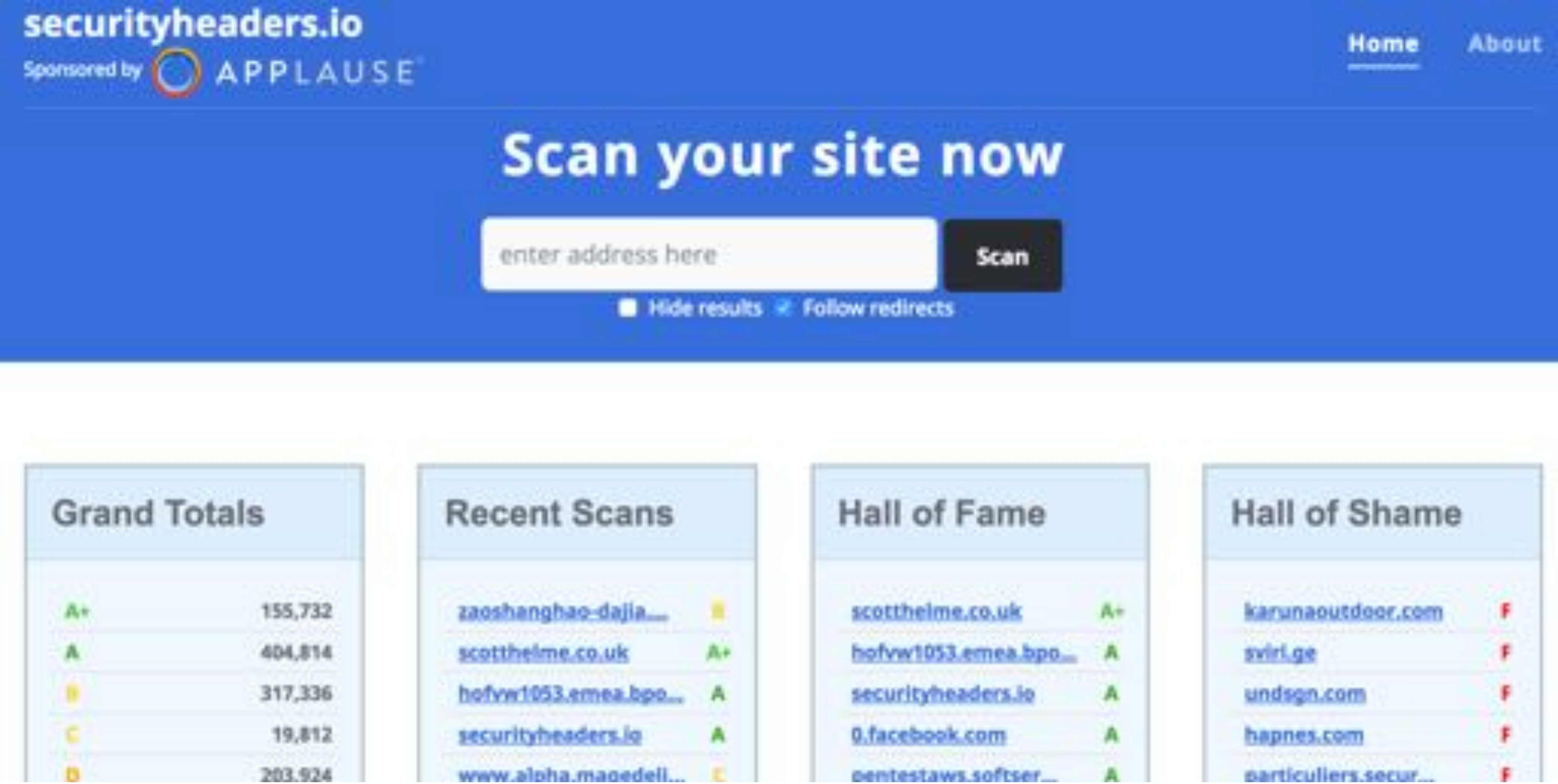
Overall Rating

A+

Visit our documentation page for more information, configuration guides, and books. Known issues are documented here.

HTTP Strict Transport Security (HSTS) with long duration deployed on this server. MORE INFO »

https://www.ssllabs.com/ssltest/analyze.html?d=websec.be



securityheaders.io

Sponsored by APPLAUSE

Scan your site now

□ Hide results ☑ Follow redirects

Grand Totals

A+	155,732
A	404,814
B	317,336
C	19,812
D	203,924

Recent Scans

zaoshanghao-dajia...	B
scotthelme.co.uk	A+
hofvw1053.emea.bpo...	A
securityheaders.io	A
www.alpha.magedeli...	C

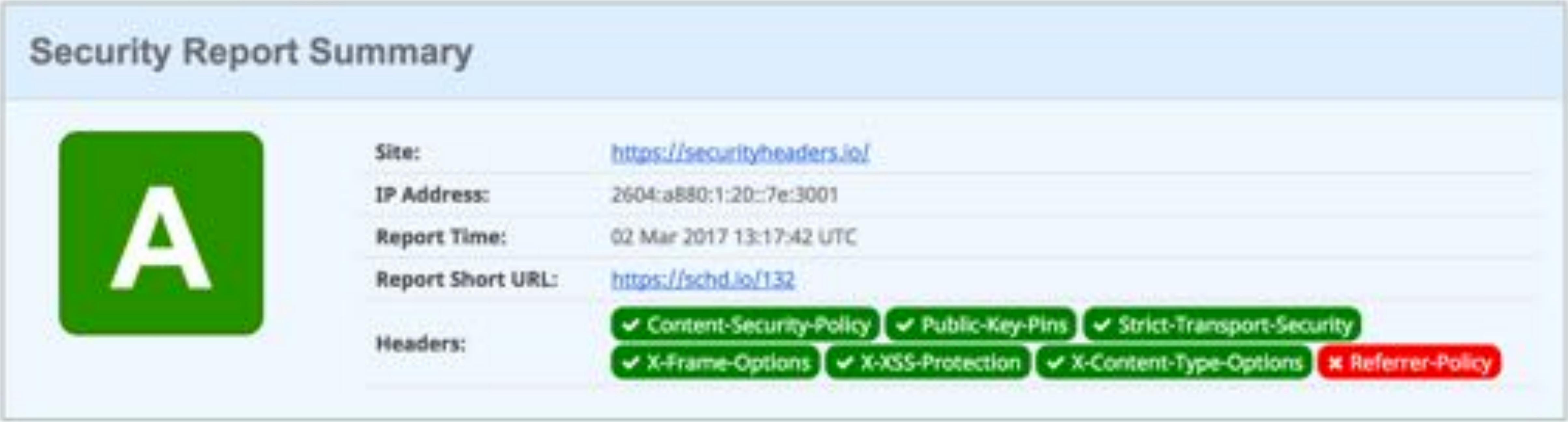
Hall of Fame

scotthelme.co.uk	A+
hofvw1053.emea.bpo...	A
securityheaders.io	A
0.facebook.com	A
pentestaws.softser...	A

Hall of Shame

karunaoutdoor.com	F
sviri.ge	F
undsgn.com	F
hapnes.com	F
particuliers.secur...	F

https://securityheaders.io/



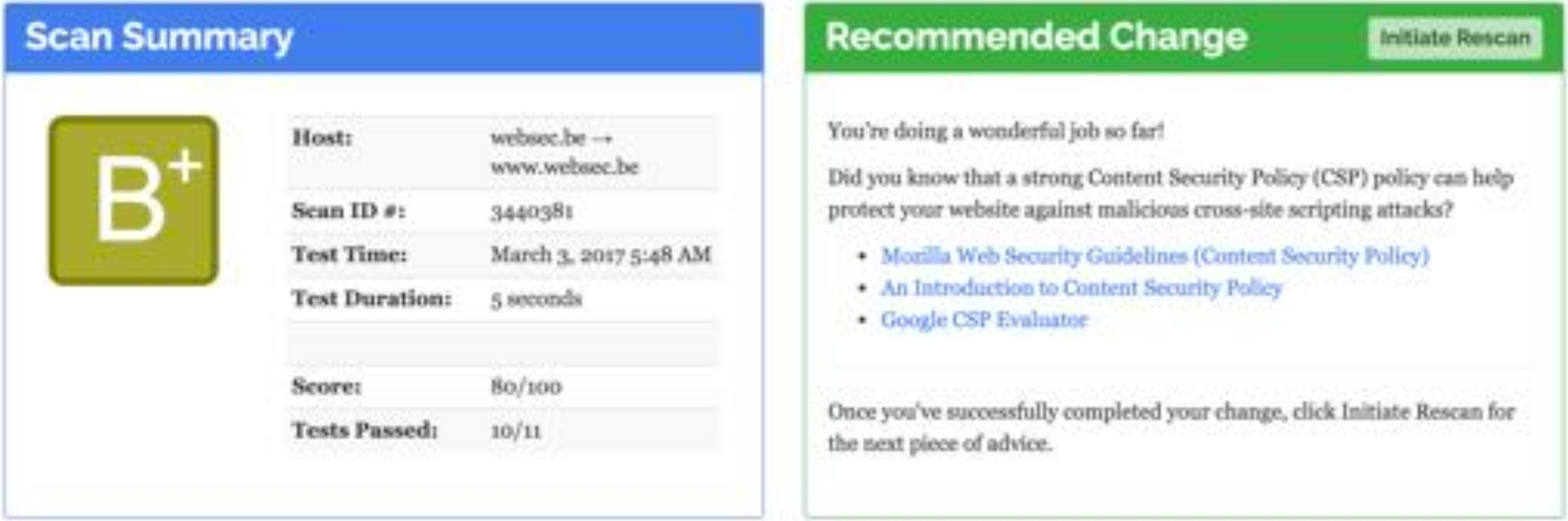
Security Report Summary

A

Site:	https://securityheaders.io/
IP Address:	2604:a880:1:20::7e:3001
Report Time:	02 Mar 2017 13:17:42 UTC
Report Short URL:	https://schd.io/132
Headers:	✓ Content-Security-Policy ✓ Public-Key-Pins ✓ Strict-Transport-Security ✓ X-Frame-Options ✓ X-XSS-Protection ✓ X-Content-Type-Options ✗ Referrer-Policy

https://securityheaders.io/?q=https%3A%2F%2Fsecurityheaders.io%2F

OBSERVATORY BY MOZILLA



Scan Summary

B+

Host:	websec.be → www.websec.be
Scan ID #:	3440381
Test Time:	March 3, 2017 5:48 AM
Test Duration:	5 seconds
Score:	80/100
Tests Passed:	10/11

Recommended Change

You're doing a wonderful job so far!

Did you know that a strong Content Security Policy (CSP) policy can help protect your website against malicious cross-site scripting attacks?

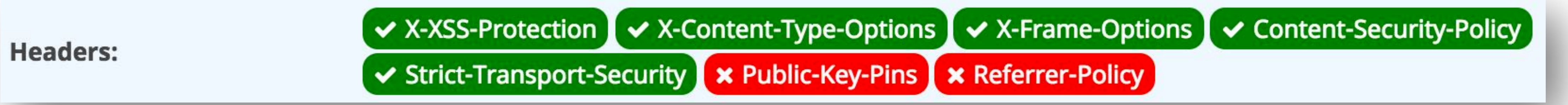
- Mozilla Web Security Guidelines (Content Security Policy)
- An Introduction to Content Security Policy
- Google CSP Evaluator

Once you've successfully completed your change, click Initiate Rescan for the next piece of advice.

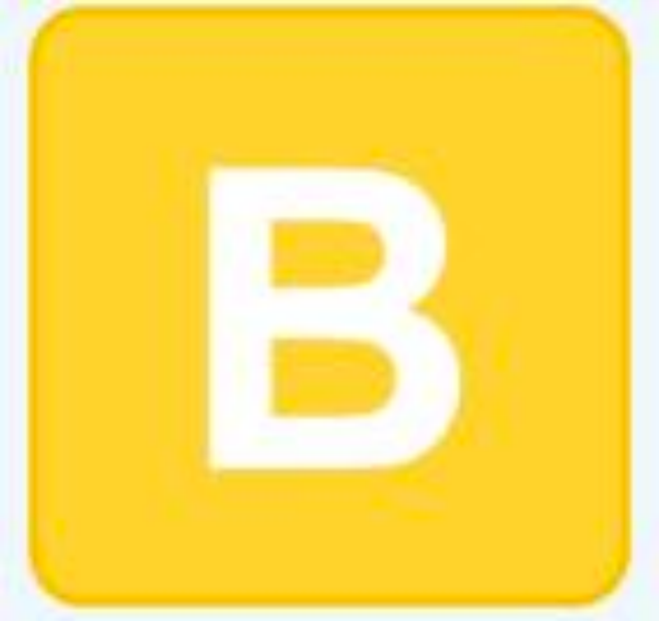
https://observatory.mozilla.org/

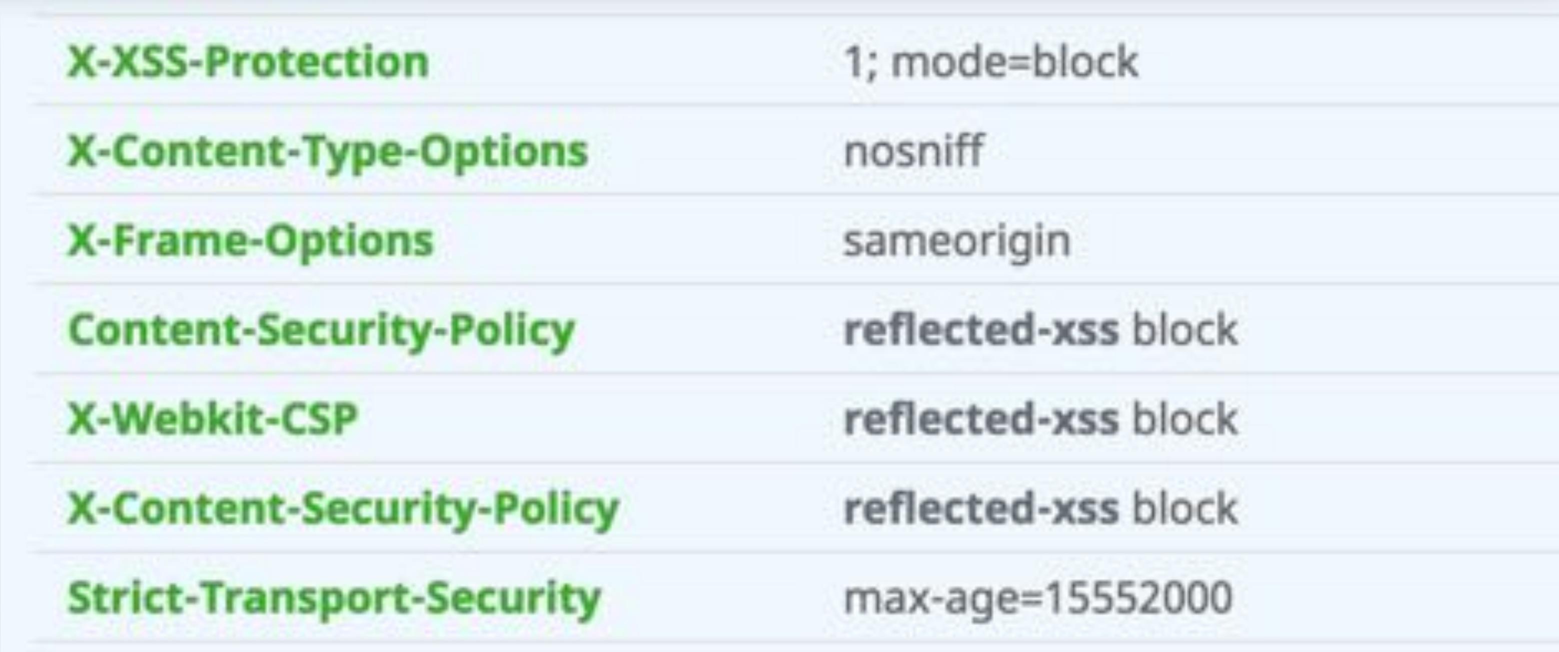
PROS / CONS OF SECURITY SCANNERS

- Security scanners play an important role in awareness
  - Grade-based evaluation is a strong motivator to improve your security



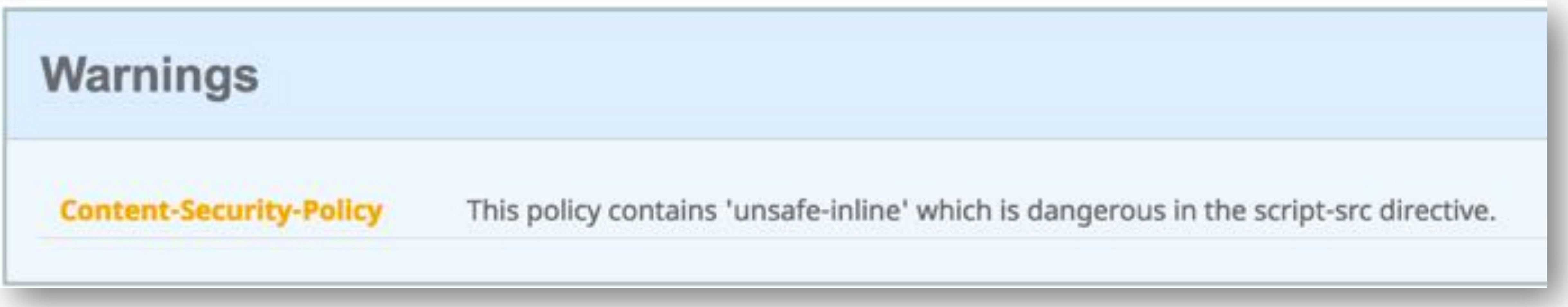
Headers: ✓ X-XSS-Protection ✓ X-Content-Type-Options ✓ X-Frame-Options ✓ Content-Security-Policy ✓ Strict-Transport-Security ✗ Public-Key-Pins ✗ Referrer-Policy





X-XSS-Protection	1; mode=block
X-Content-Type-Options	nosniff
X-Frame-Options	sameorigin
Content-Security-Policy	**reflected-xss** block
X-Webkit-CSP	**reflected-xss** block
X-Content-Security-Policy	**reflected-xss** block
Strict-Transport-Security	max-age=15552000

PROS / CONS OF SECURITY SCANNERS



Warnings

Content-Security-Policy This policy contains 'unsafe-inline' which is dangerous in the script-src directive.

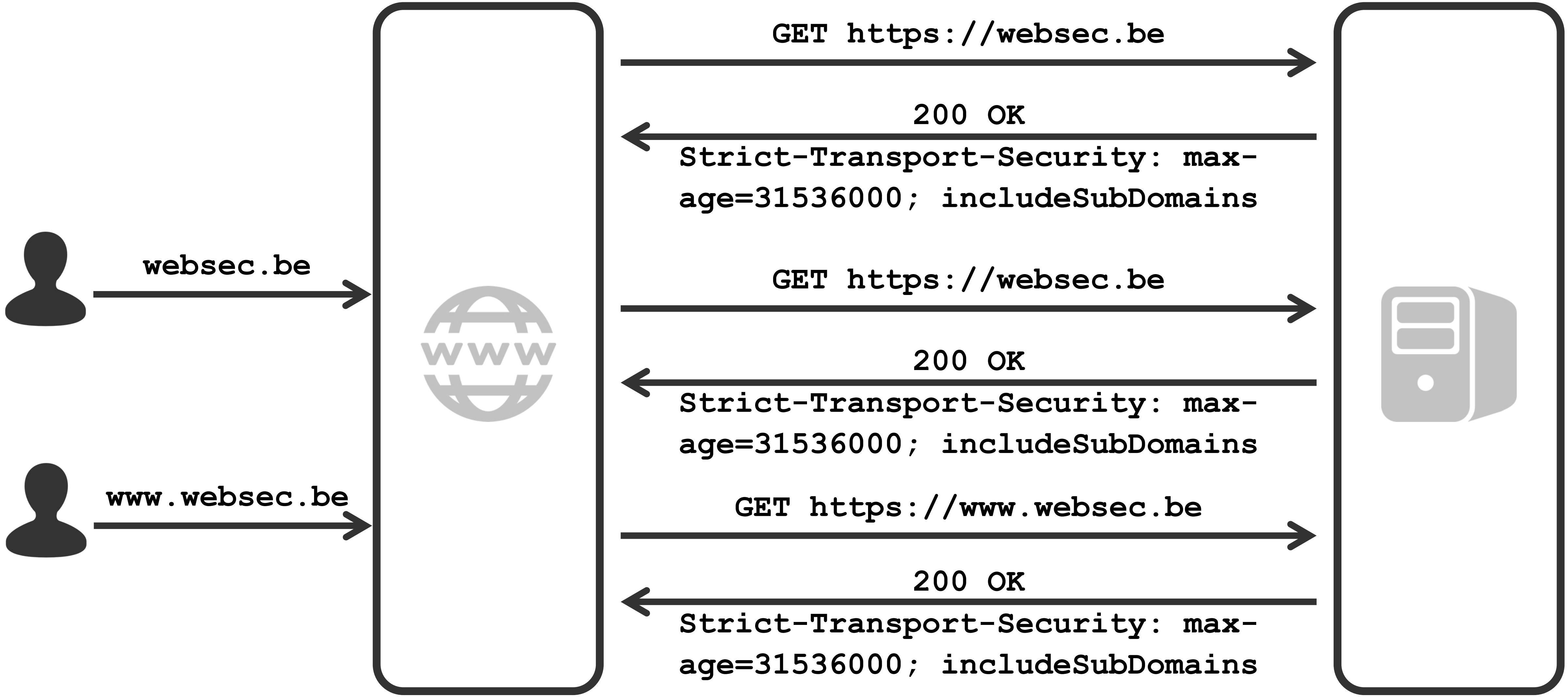
PROS / CONS OF SECURITY SCANNERS

- Security scanners play an important role in awareness
  - – Grade-based evaluation is a strong motivator to improve your security
- Fundamentally, this raises a lot of questions
  - – How do you know you understood the security measure correctly?
  - – How do you know your configuration is secure?
  - – How do you know you covered it all?
  - – And if you don’t get an A, what do you focus on first?
- The real answer comes down to knowledge
  - – Understand the security technology, and make sure it fits within your context

HTTP Strict Transport Security

Case study 1

HSTS TRANSFORMS HTTP INTO HTTPS



GET https://websec.be

200 OK
Strict-Transport-Security: max-age=31536000; includeSubDomains

websec.be

GET https://websec.be

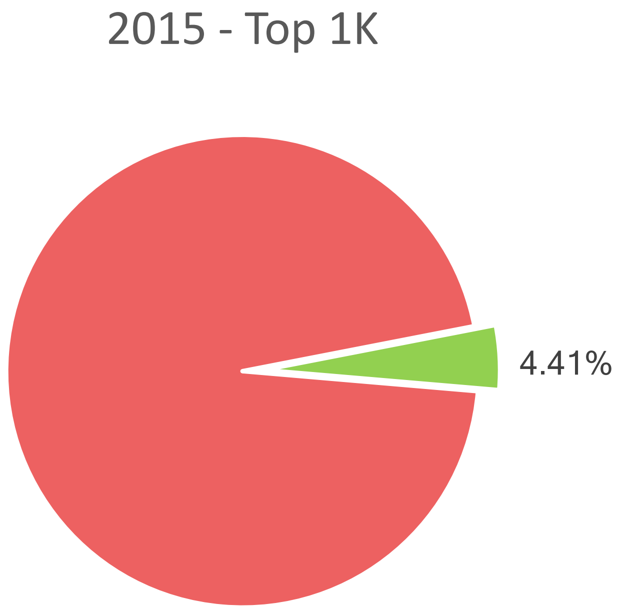
200 OK
Strict-Transport-Security: max-age=31536000; includeSubDomains

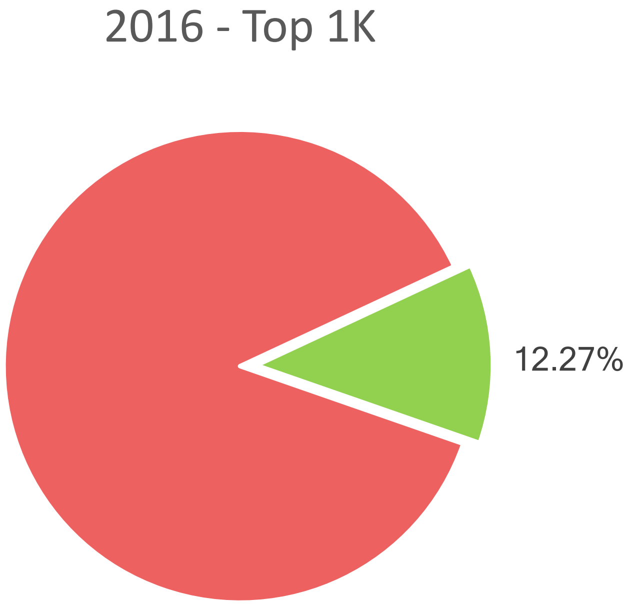
www.websec.be

GET https://www.websec.be

200 OK
Strict-Transport-Security: max-age=31536000; includeSubDomains

HSTS USAGE STATISTICS

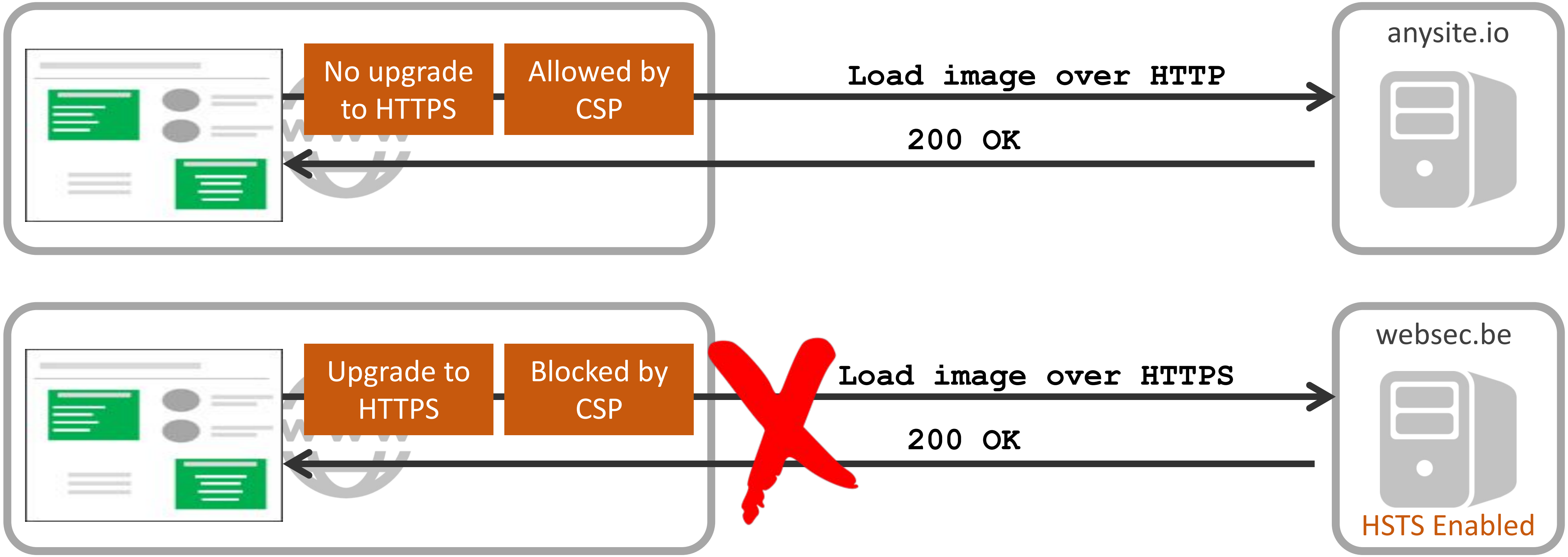




https://www.owasp.org/index.php/OWASP_Secure_Headers_Project#tab=Stats

History sniffing with HSTS and CSP

```
<img src="http://…" onerror="detectTiming()"/>
```



No upgrade to HTTPS

Allowed by CSP

Load image over HTTP

200 OK

anysite.io

Upgrade to HTTPS

Blocked by CSP

Load image over HTTPS

200 OK

websec.be

HSTS Enabled

```
Content-Security-Policy: img-src http:
```

History sniffing with HSTS and CSP

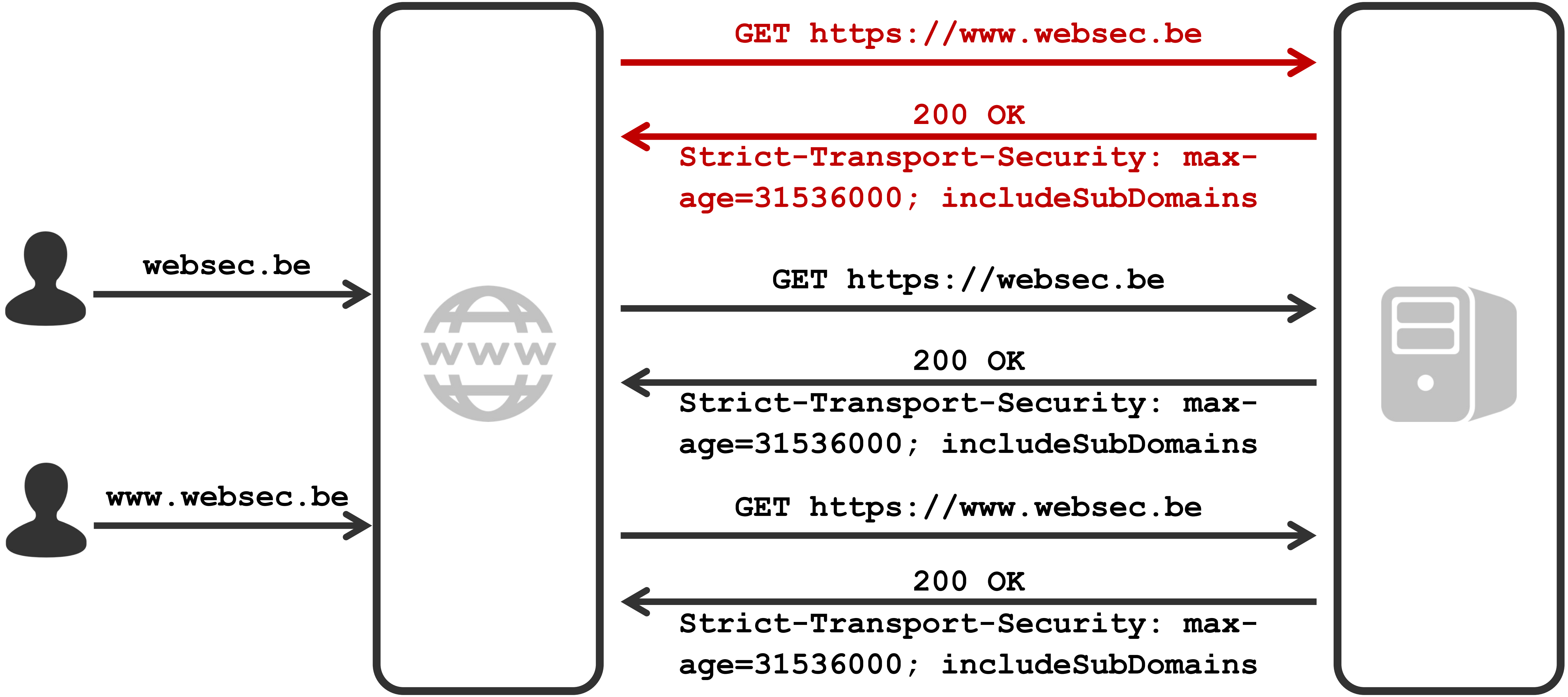
- Sites that deploy HSTS redirect HTTP to HTTPS
  - The browser will load HTTP resources over HTTPS

- Sniffly is a timing tool that loads an image over HTTP, while blocking it with CSP
  - Based on timing, it determines whether your browser knew the site or not

- Attacks like this are somewhat inherent to what HSTS does
  - Yet, this specific attack has been prevented by modifying the CSP spec
  - CSP no longer allows you to lock yourself in to use only insecure resources
    - `http:` is essentially treated as `http: https:`

https://github.com/diracdeltas/sniffly

SECURING THE FIRST CONNECTION OVER HTTPS …



websec.be

www.websec.be

GET https://www.websec.be

200 OK
Strict-Transport-Security: max-age=31536000; includeSubDomains

GET https://websec.be

200 OK
Strict-Transport-Security: max-age=31536000; includeSubDomains

GET https://www.websec.be

200 OK
Strict-Transport-Security: max-age=31536000; includeSubDomains

PRELOAD COPY/PASTING

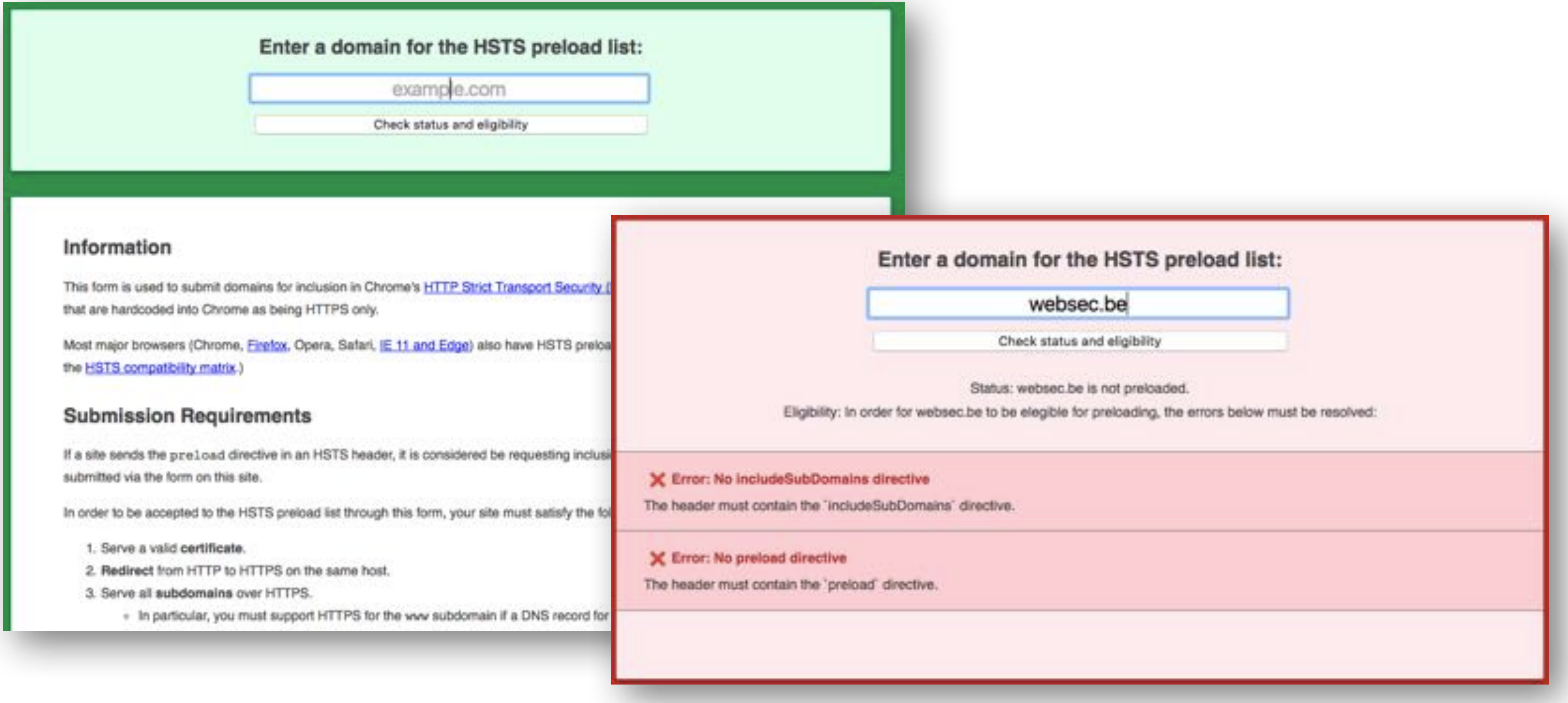
- HTS sites can opt-in to be preloaded in the browser
  - This requires explicit consent by adding the preload flag to the header

```
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
```

- It turns out that many sites give this consent, without being on the list
  - Theoretically, this allows anyone to put them on the list
  - Once on the list, it’s HTTPS or nothing

- The preload site actually performs some sanity checks before adding you
  - So this prevents rampant abuse of this kind of feature

Preloading HSTS into the browser



Enter a domain for the HSTS preload list:

Information

This form is used to submit domains for inclusion in Chrome's HTTP Strict Transport Security (
that are hardcoded into Chrome as being HTTPS only.

Most major browsers (Chrome, Firefox, Opera, Safari, IE 11 and Edge) also have HSTS preloa
the HSTS compatibility matrix.)

Submission Requirements

If a site sends the `preload` directive in an HSTS header, it is considered be requesting inclusi
submitted via the form on this site.

In order to be accepted to the HSTS preload list through this form, your site must satisfy the fo

1. Serve a valid **certificate**.
2. **Redirect** from HTTP to HTTPS on the same host.
3. Serve all **subdomains** over HTTPS.
   - In particular, you must support HTTPS for the www subdomain if a DNS record for

Enter a domain for the HSTS preload list:

websec.be

Status: websec.be is not preloaded.

Eligibility: In order for websec.be to be elegible for preloading, the errors below must be resolved:

✗ **Error: No includeSubDomains directive**

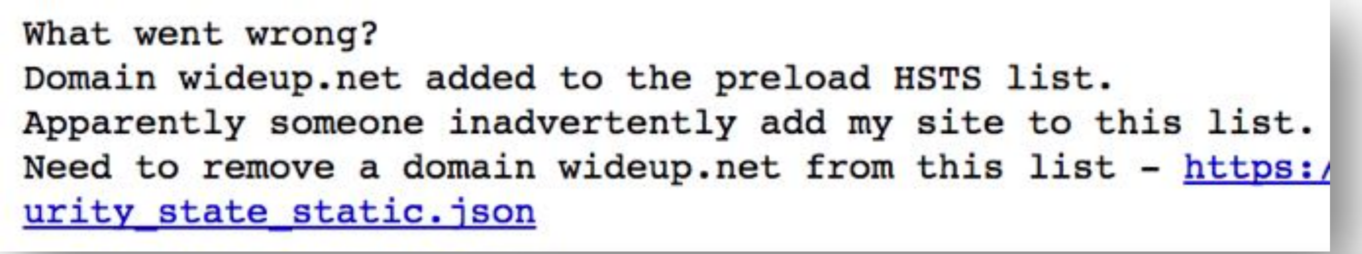
The header must contain the `includeSubDomains` directive.

✗ **Error: No preload directive**

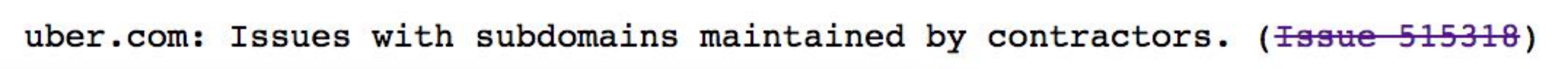
The header must contain the `preload` directive.

https://hstspreload.appspot.com/?

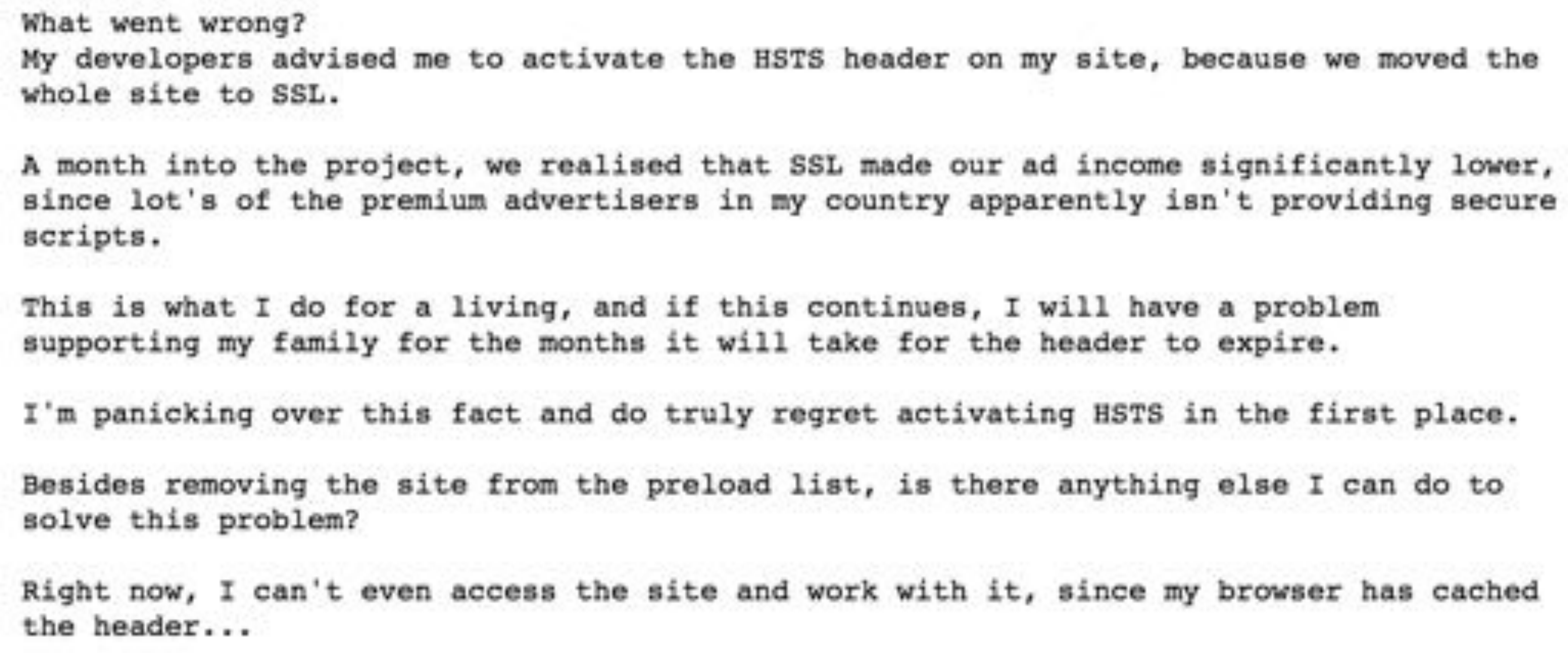
PRELOAD FOR BETTER OR FOR WORSE



What went wrong?
Domain wideup.net added to the preload HSTS list.
Apparently someone inadvertently add my site to this list.
Need to remove a domain wideup.net from this list - https:/
urity_state_static.json



uber.com: Issues with subdomains maintained by contractors. (~~Issue 515318~~)



What went wrong?
My developers advised me to activate the HSTS header on my site, because we moved the whole site to SSL.

A month into the project, we realised that SSL made our ad income significantly lower, since lot's of the premium advertisers in my country apparently isn't providing secure scripts.

This is what I do for a living, and if this continues, I will have a problem supporting my family for the months it will take for the header to expire.

I'm panicking over this fact and do truly regret activating HSTS in the first place.

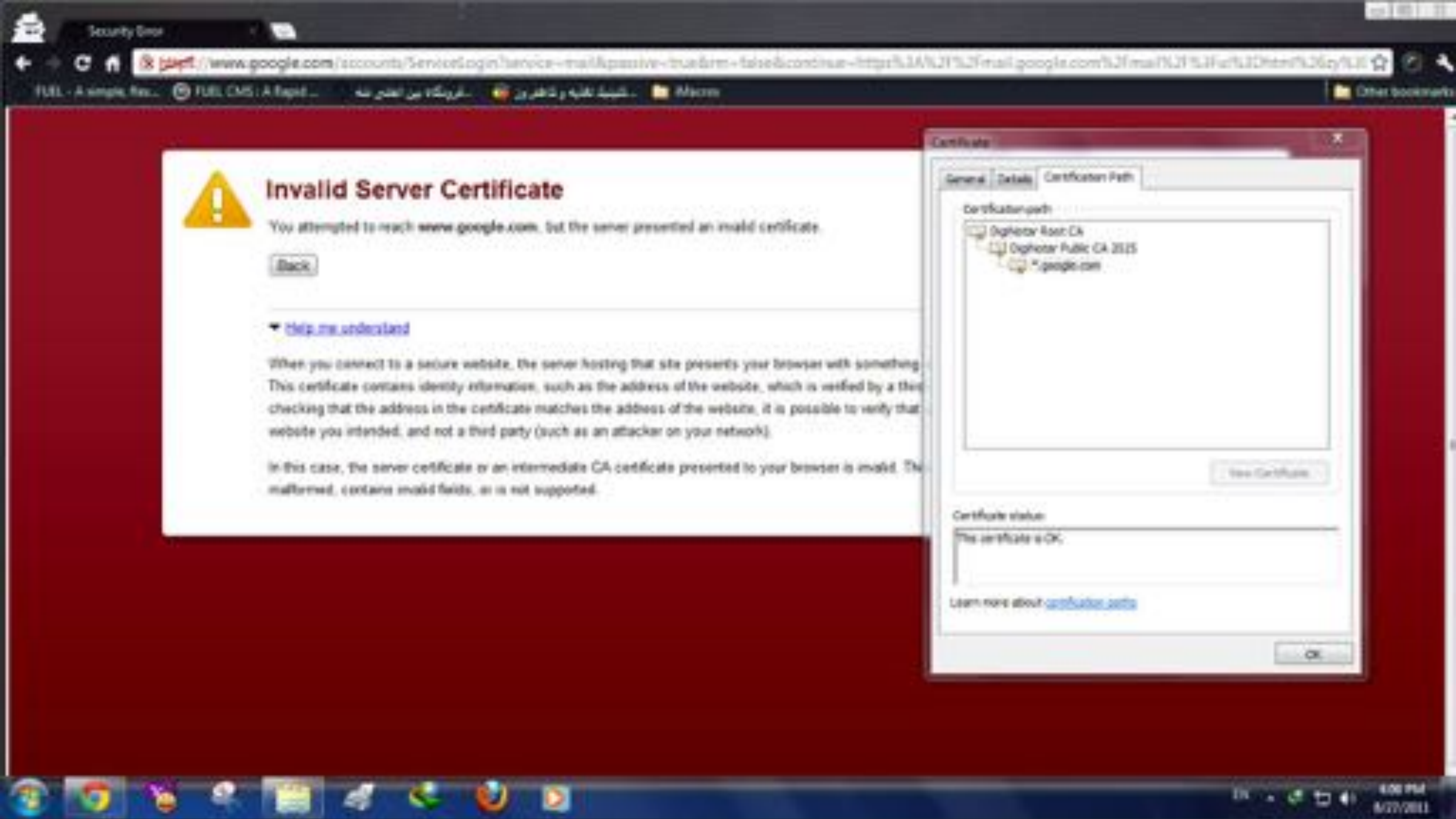
Besides removing the site from the preload list, is there anything else I can do to solve this problem?

Right now, I can't even access the site and work with it, since my browser has cached the header...

https://bugs.chromium.org/p/chromium/issues/detail?id=527947

HTTP Public Key Pinning

Case study 2



Invalid Server Certificate

You attempted to reach **www.google.com**, but the server presented an invalid certificate.

When you connect to a secure website, the server hosting that site presents your browser with something
This certificate contains identity information, such as the address of the website, which is verified by a thi
checking that the address in the certificate matches the address of the website, it is possible to verify that
website you intended, and not a third party (such as an attacker on your network).

In this case, the server certificate or an intermediate CA certificate presented to your browser is invalid. Th
malformed, contains invalid fields, or is not supported.

This certificate is OK.

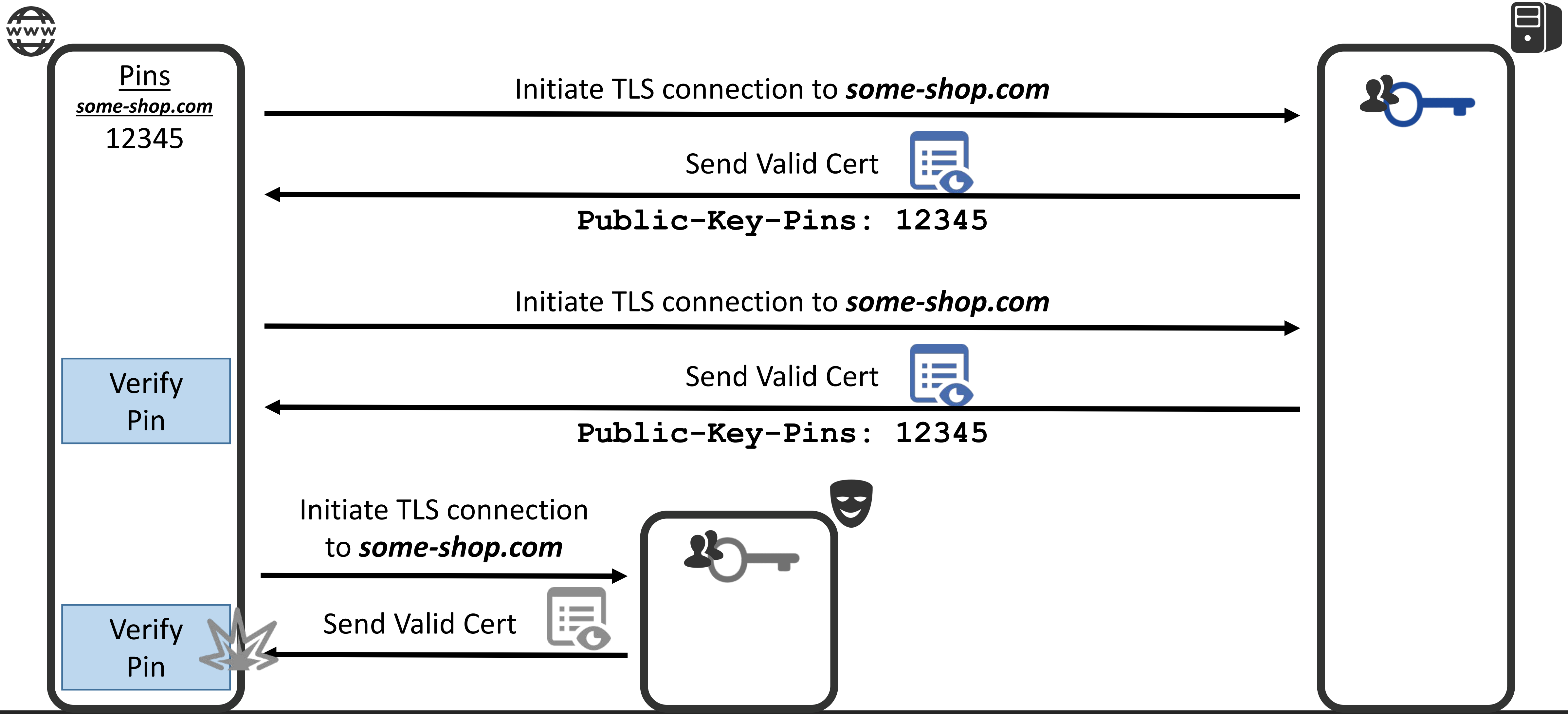
HTTP Public-Key Pinning (HPKP)

- HPKP is a server-driven, browser-enforced security policy
  - Instructs the browser to only accept a pinned public key
  - Intended to be used in combination with HSTS

- Pins associate a hostname with a cryptographic identity
  - Can be on certificate level, CA level, …
  - Trade-off between specificity and resilience

```
Public-Key-Pins: max-age=3000;
   pin-sha256="d6qzRu9zOECb90Uez27xWltNsj0e1Md7GkYYkVoZWmM=";
   pin-sha256="E9CZ9INDbd+2eRQozYqqbQ2yXLVKB9+xcprMF+44U1g="
```

HTTP Public-Key Pinning (HPKP)



Pins
some-shop.com
12345

Initiate TLS connection to ***some-shop.com***

Send Valid Cert

`Public-Key-Pins: 12345`

Initiate TLS connection to ***some-shop.com***

Verify Pin

Send Valid Cert

`Public-Key-Pins: 12345`

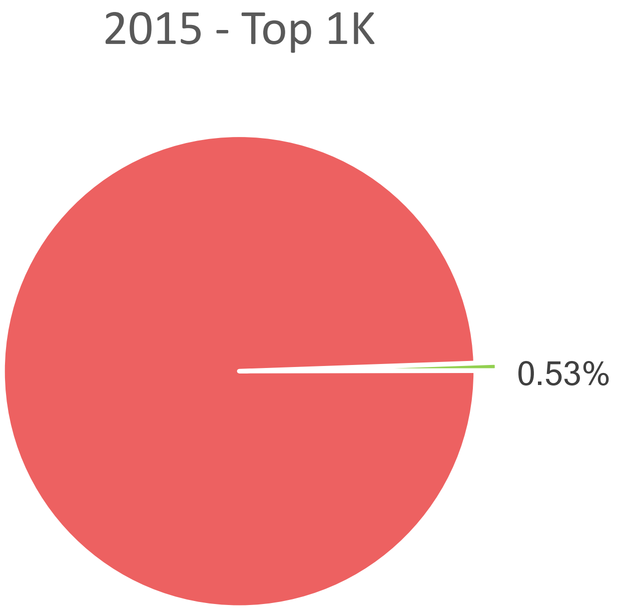
Initiate TLS connection to ***some-shop.com***

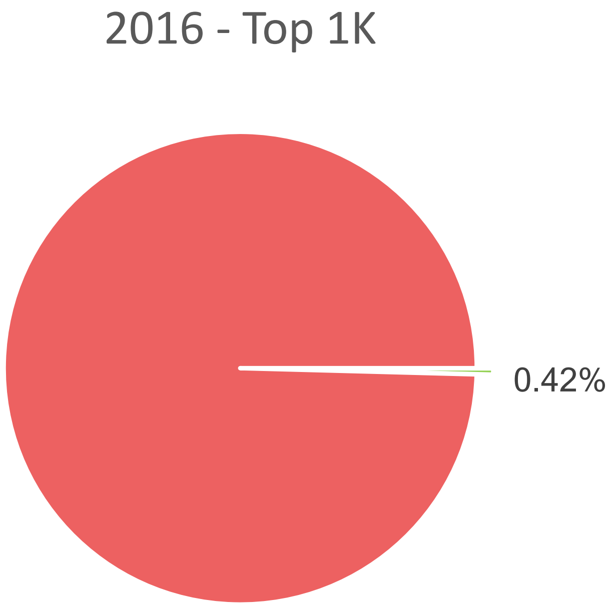
Verify Pin

Send Valid Cert

some-shop.com

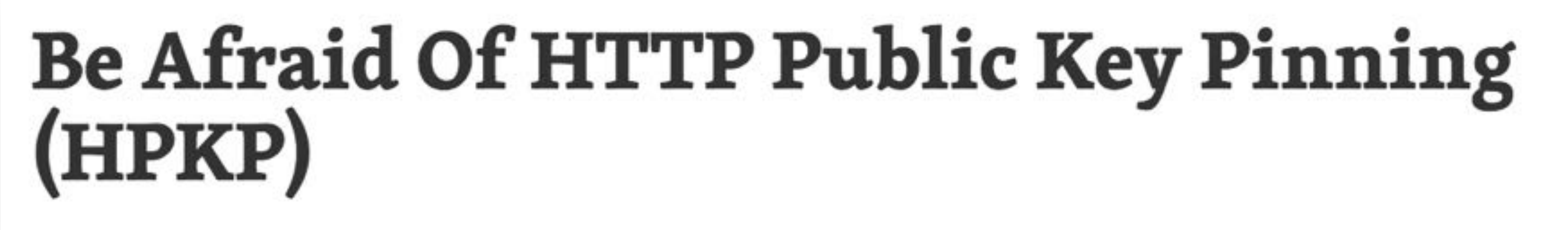
HPKP USAGE STATISTICS



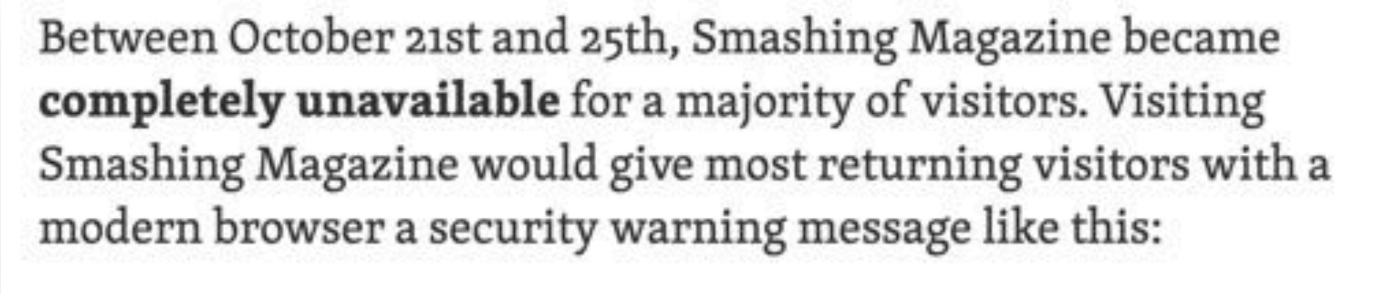


https://www.owasp.org/index.php/OWASP_Secure_Headers_Project#tab=Stats

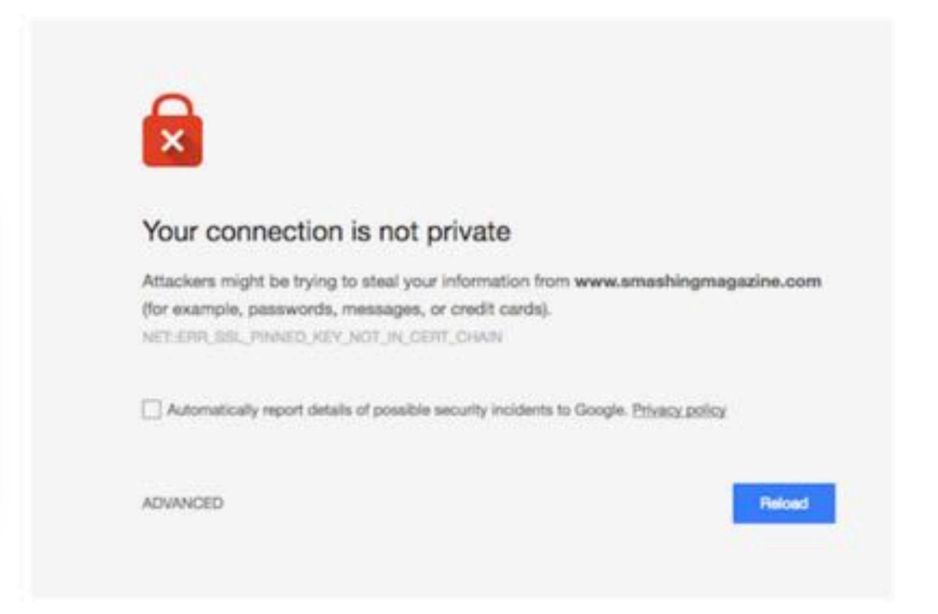
HPKP is awesome, ask Smashing Magazine



Be Afraid Of HTTP Public Key Pinning (HPKP)



Between October 21st and 25th, Smashing Magazine became **completely unavailable** for a majority of visitors. Visiting Smashing Magazine would give most returning visitors with a modern browser a security warning message like this:



Your connection is not private

Attackers might be trying to steal your information from **www.smashingmagazine.com** (for example, passwords, messages, or credit cards).

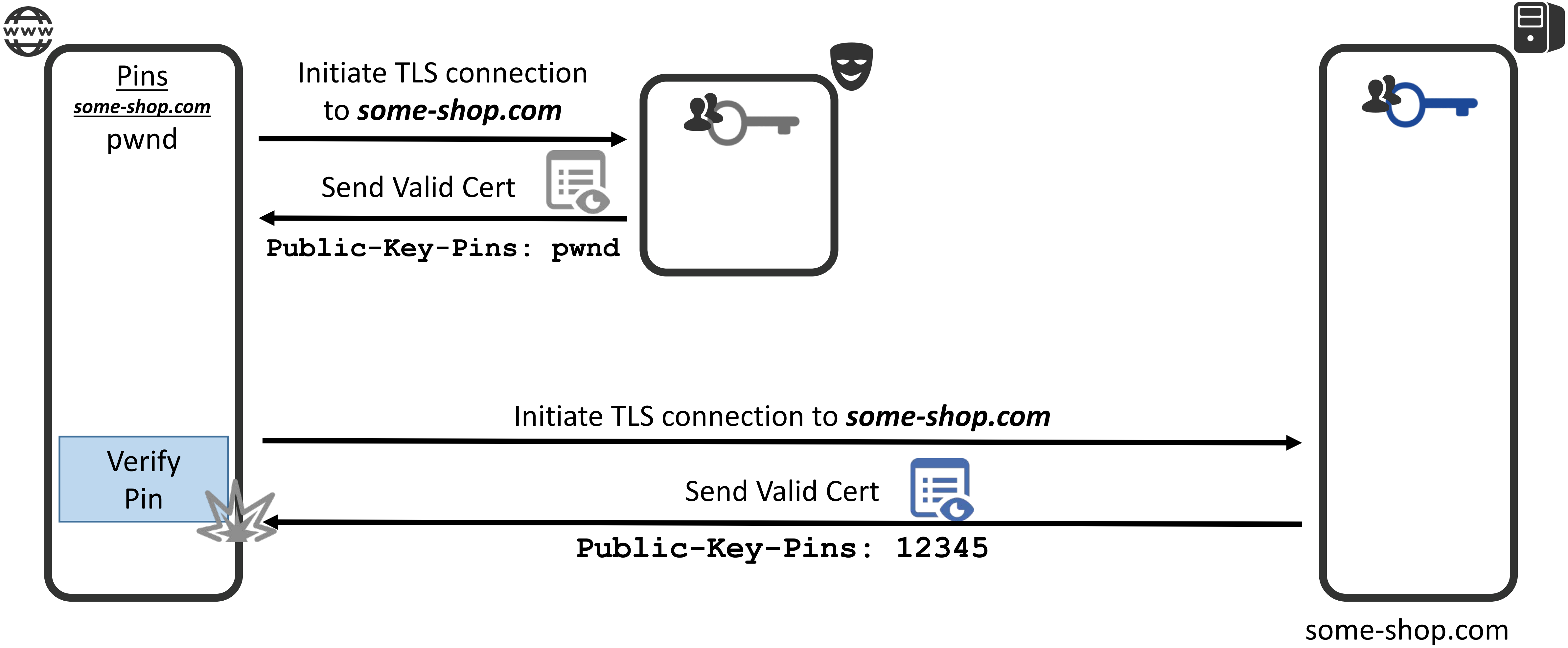
NET::ERR_SSL_PINNED_KEY_NOT_IN_CERT_CHAIN

□ Automatically report details of possible security incidents to Google. Privacy policy

The warning message most of Smashing Magazine's visitors were seeing.

https://www.smashingmagazine.com/be-afraid-of-public-key-pinning/

WHAT CAN GO WRONG WITH HPKP?



Dealing with Hostile Pinning

- Has been coined as HPKP Suicide or RansomPKP
  - Concerns scenarios where your server is compromised
  - Pins are served to your users, and this cannot be easily undone

- Hostile pinning is a difficult problem to solve
  - Spec suggests that browsers limit the duration of max-age
  - Use complementary solutions like Certificate Transparency

- You probably do not need HPKP on your site
  - You can deploy HPKP in report-only mode, giving you reports about potential problems
  - However, powerful attackers can simply suppress reports as well

X-XSS-Protection

Case study 3

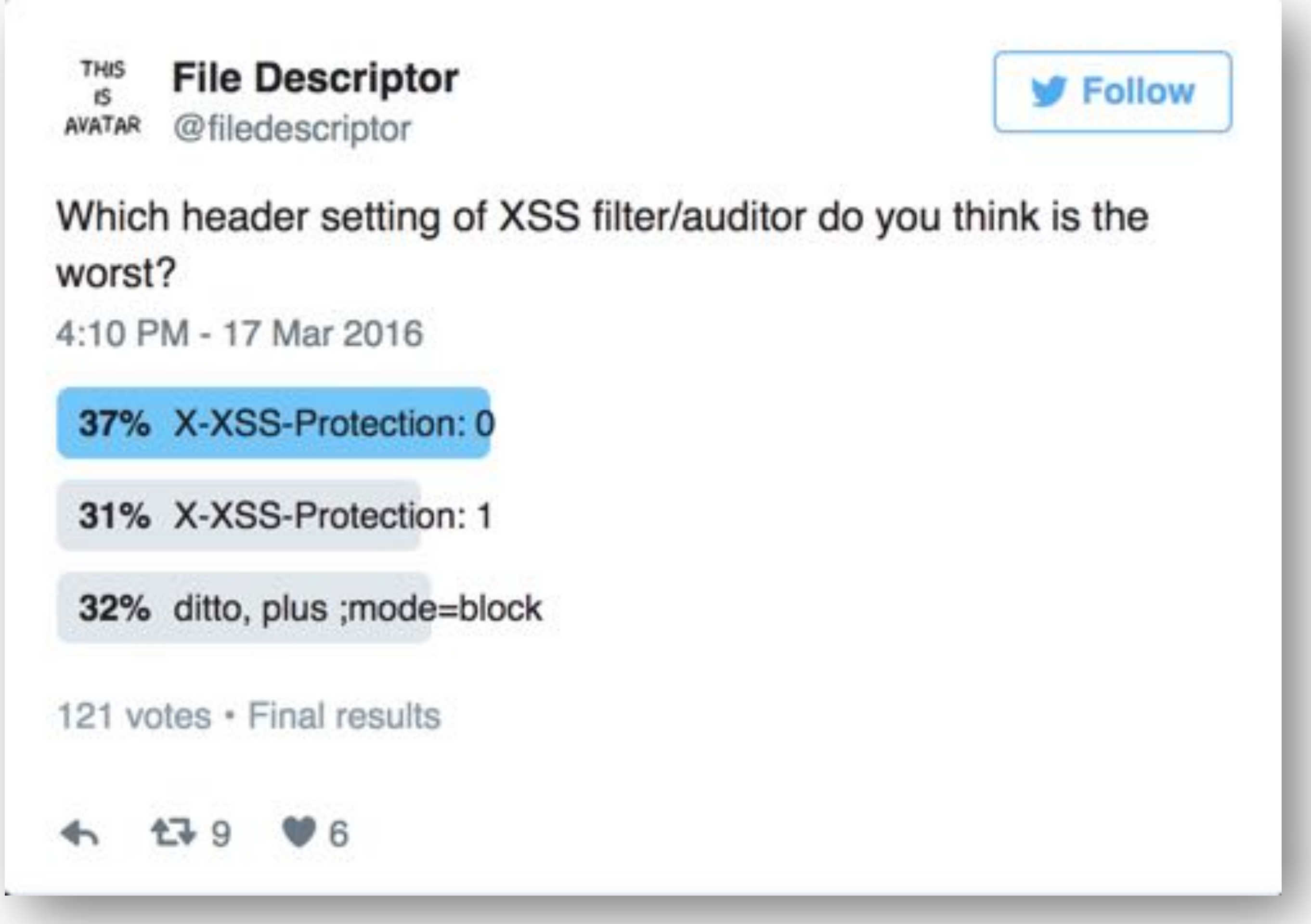
AUTOMATIC BROWSER-BASED XSS PROTECTION

- Browser-based protection against reflected XSS
  - Scan outgoing requests for potential payloads (URL, body)
  - Inspect if the payload is reflected back in the response

- Initial version introduced in IE8, known as XSS filter
  - Chrome and Safari have something similar with the XSS Auditor
  - Intended as a defense-in-depth mechanism, not a core security feature

- Mechanism can be configured with the X-XSS-Protection header
  - Default behavior is to try and remove the malicious payload
  - Response is rewritten before it is rendered

What is the best header setting?



THIS IS AVATAR **File Descriptor** @filedescriptor

Which header setting of XSS filter/auditor do you think is the worst?

4:10 PM - 17 Mar 2016

37% X-XSS-Protection: 0

31% X-XSS-Protection: 1

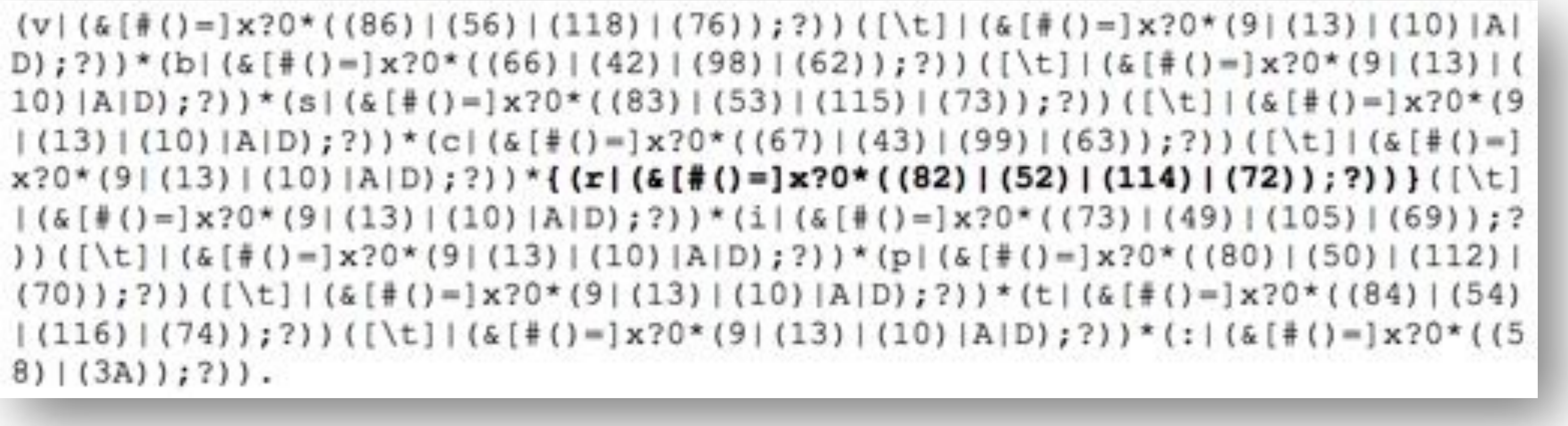
32% ditto, plus ;mode=block

121 votes • Final results

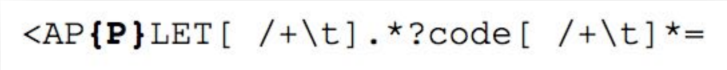
http://blog.innerht.ml/the-misunderstood-x-xss-protection/

The dangers of automated sanitization

- IE rewrites the response to render the payload harmless
  - – # is inserted to change the meaning of the code, thus preventing the attack
  - – The process is regex based



```
(v|(&[#()=]x?0*((86)|(56)|(118)|(76));?))([\t]|(&[#()=]x?0*(9|(13)|(10)|A|
D);?))*(b|(&[#()=]x?0*((66)|(42)|(98)|(62));?))([\t]|(&[#()=]x?0*(9|(13)|(
10)|A|D);?))*(s|(&[#()=]x?0*((83)|(53)|(115)|(73));?))([\t]|(&[#()=]x?0*(9
|(13)|(10)|A|D);?))*(c|(&[#()=]x?0*((67)|(43)|(99)|(63));?))([\t]|(&[#()=]
x?0*(9|(13)|(10)|A|D);?))*{(r|(&[#()=]x?0*((82)|(52)|(114)|(72));?))}([\t]
|(&[#()=]x?0*(9|(13)|(10)|A|D);?))*(i|(&[#()=]x?0*((73)|(49)|(105)|(69));?
))([\t]|(&[#()=]x?0*(9|(13)|(10)|A|D);?))*(p|(&[#()=]x?0*((80)|(50)|(112)|
(70));?))([\t]|(&[#()=]x?0*(9|(13)|(10)|A|D);?))*(t|(&[#()=]x?0*((84)|(54)
|(116)|(74));?))([\t]|(&[#()=]x?0*(9|(13)|(10)|A|D);?))*(:|(&[#()=]x?0*((5
8)|(3A));?)).
```



```
<AP{P}LET[ /+\t].*?code[ /+\t]*=
```

http://p42.us/ie8xss/Abusing_IE8s_XSS_Filters.pdf

THE DANGERS OF AUTOMATED SANITIZATION

- IE rewrites the response to render the payload harmless
  - # is inserted to change the meaning of the code, thus preventing the attack
  - The process is regex based

- IE can be tricked into rewriting harmless code into XSS code

```
<img alt=“x onload=alert(0) x” src=”x.png”>
```

```
<img alt#“x onload=alert(0) x” src=”x.png”>
```

http://p42.us/ie8xss/Abusing_IE8s_XSS_Filters.pdf

So just block the page load when an attack is detected

- The header can be configured to block the page load completely
  - The context remains `about:blank` instead of loading the HTML from the response

- Seems like a solid protection mechanism, but Facebook may disagree
  - People chained a couple of bugs to steal OAuth 2.0 access tokens
  - Awarded $5000 bug bounty from Facebook, and resulted in a patch in Chrome
  - Facebook turns off `X-XSS-Protection` completely

- A brief overview of what causes these problems
  - `about:blank` inherits the origin of the parent page
  - After blocking the page load, `document.referrer` contains the last seen URL
  - Because of origin inheritance, this value is accessible to the parent frame

http://homakov.blogspot.be/2013/02/hacking-facebook-with-oauth2-and-chrome.html

Content Security Policy

Case study 4

```
<h1>My PHP app</h1>
<h3>Hi <script>alert(1)</script></h3>

<button onclick="doSomething()">
  Click me
</button>
<script>
  function doSomething() { ... }
</script>
<p>
  ...
  <script src="http://evil.com/hackme.js"></script>
</p>
```

Reining in the Web with Content Security Policy

Sid Stamm
Mozilla
sid@mozilla.com

Brandon Sterne
Mozilla
bsterne@mozilla.com

Gervase Markham
Mozilla
gerv@mozilla.org

ABSTRACT

The last three years have seen a dramatic increase in both awareness and exploitation of Web Application Vulnerabilities. 2008 and 2009 saw dozens of high-profile attacks against websites using Cross Site Scripting (XSS) and Cross Site Request Forgery (CSRF) for the purposes of information stealing, website defacement, malware planting, clickjacking, etc. While an ideal solution may be to develop web applications free from any exploitable vulnerabilities, real world security is usually provided in layers.

We present content restrictions, and a content restrictions enforcement scheme called Content Security Policy (CSP), which intends to be one such layer. Content restrictions allow site designers or server administrators to specify how content interacts on their web sites—a security mechanism desperately needed by the untamed Web. These content restrictions rules are activated and enforced by supporting web browsers when a policy is provided for a site via HTTP, and we show how a system such as CSP can be effective to lock down sites and provide an early alert system for vulner-

exploiting browser or site-specific vulnerabilities to steal or inject information.

Additionally, browser and web application providers are having a hard time deciding what exactly should be a "domain" or "origin" when referring to web traffic. With the advent of DNS rebinding [8] and with the gray area regarding ownership of sibling sub-domains (like `user1.webhost.com` versus `user2.webhost.com`), it may be ideal to allow the service providers who write web applications the opportunity to specify, or fence-in, what they consider to be their domain.

1.1 Uncontrolled Web Platform

Web sites currently execute in a mostly uncontrolled web browser environment. The sole protection currently afforded to websites with regards to policies restricting content is the same-origin policy (SOP) [20]. Although this policy is deployed in browsers, attackers are still able to subvert the policy by directly attacking the site and injecting their own script into the content. For example, an attacker may post a message to `messageboard.com` that is rendered for all future

http://www.ambuehler.ethz.ch/CDstore/www2010/www/p921.pdf

The goal of Content Security Policy (CSP)

- CSP is intended as a defense-in-depth mechanism against injection attacks
  - Gives developers a way to lock down their application in various ways
  - Constrains an attacker in case of an injection vulnerability in the application
  - ***CSP is not a replacement for traditional XSS mitigation techniques***

- CSP places two kinds of restrictions on a page
  - It disables “dangerous features” (e.g. inline scripts, inline styles and the use of eval)
  - It only loads resources that are explicitly whitelisted, and blocks everything else

- CSP is an extensive security policy, with a wide variety of features
  - We will focus on its capabilities to restrict XSS attacks first

USING CSP TO RESTRICT INJECTED SCRIPTS

Injection of inline scripts

```
<h1>You searched for <script>…</script></h1>
```

By default, CSP prevents the execution of inline script blocks

Injection of remote scripts

```
<h1>You searched for <script src=“//example.com/evil.js”></script></h1>
```

Unless you whitelist this host/file, CSP will not load the external file

Defining a CSP policy with whitelists

```
Content-Security-Policy:
      script-src ‘self’ https://www.example.com *.websec.be
```

- The browser enforces a CSP policy consisting of directives (e.g. `script-src`)
  - Delivered alongside the page as an HTTP response header
  - Included in the page as an HTML meta tag

- A directive can have numerous valid values
  - Keywords: **`‘none’`**, **`‘self’`**, **`*`**
  - Expressions: **`https://websec.be`**, **`https:`**, **`https://websec.be/jquery.js`**, **`*.websec.be`**

```
<h1>My PHP app</h1>
<h3>Hi <script>alert(1)</script></h3>

<button onclick="doSomething()">
  Click me
</button>
<script>
  function doSomething() { ... }
</script>
<p>
  ...
  <script src="http://evil.com/hackme.js"></script>
</p>
```

```
<h1>My PHP app</h1>
<h3>Hi ~~<script>aler~~

<button onclick="doSomething(
  Click me
</button>
<script>
  function doSomething() { ... }
</script>
<p>
  ...
  ~~<script src="http://evil.com/hackme.js"></script>~~
</p>
```

```
document.querySelector("button")
  .addEventListener("click", doSomething);

function doSomething() { … }
```

```
<script src="myapp.js"></script>
```

Reining in the Web with Content Security Policy

Sid Stamm
Mozilla
sid@mozilla.com

Brandon Sterne
Mozilla
bsterne@mozilla.com

Gervase Markham
Mozilla
gerv@mozilla.org

We propose the use of content restrictions to lock down websites behavior, and have provided an implementation of content restrictions called Content Security Policy.

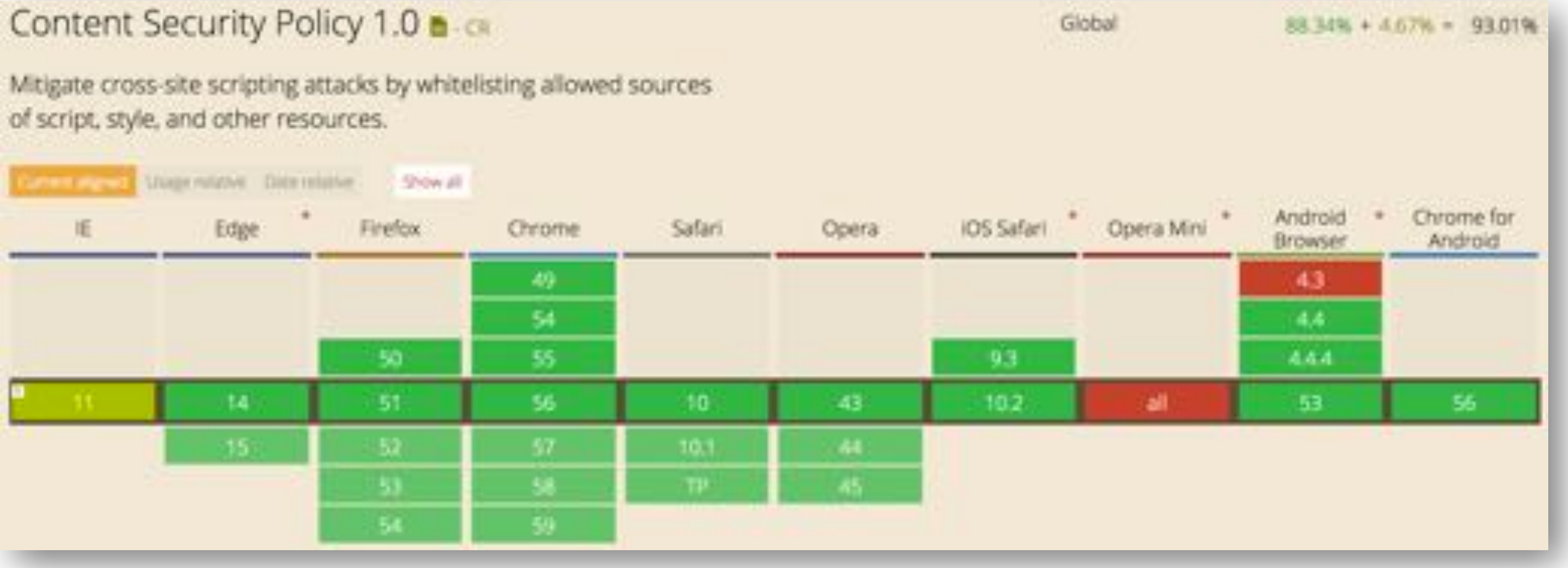
enforcement scheme called Content Security Policy (CSP), which intends to be one such layer. Content restrictions allow site designers or server administrators to specify how content interacts on their web sites—a security mechanism desperately needed by the untamed Web. These content restrictions rules are activated and enforced by supporting web browsers when a policy is provided for a site via HTTP, and we show how a system such as CSP can be effective to lock down sites and provide an early alert system for vulner-

1.1 Uncontrolled Web Platform

Web sites currently execute in a mostly uncontrolled web browser environment. The sole protection currently afforded to websites with regards to policies restricting content is the same-origin policy (SOP) [20]. Although this policy is deployed in browsers, attackers are still able to subvert the policy by directly attacking the site and injecting their own script into the content. For example, an attacker may post a message to messageboard.com that is rendered for all future

http://www.ambuehler.ethz.ch/CDstore/www2010/www/p921.pdf

Browser support – Content Security Policy level 1



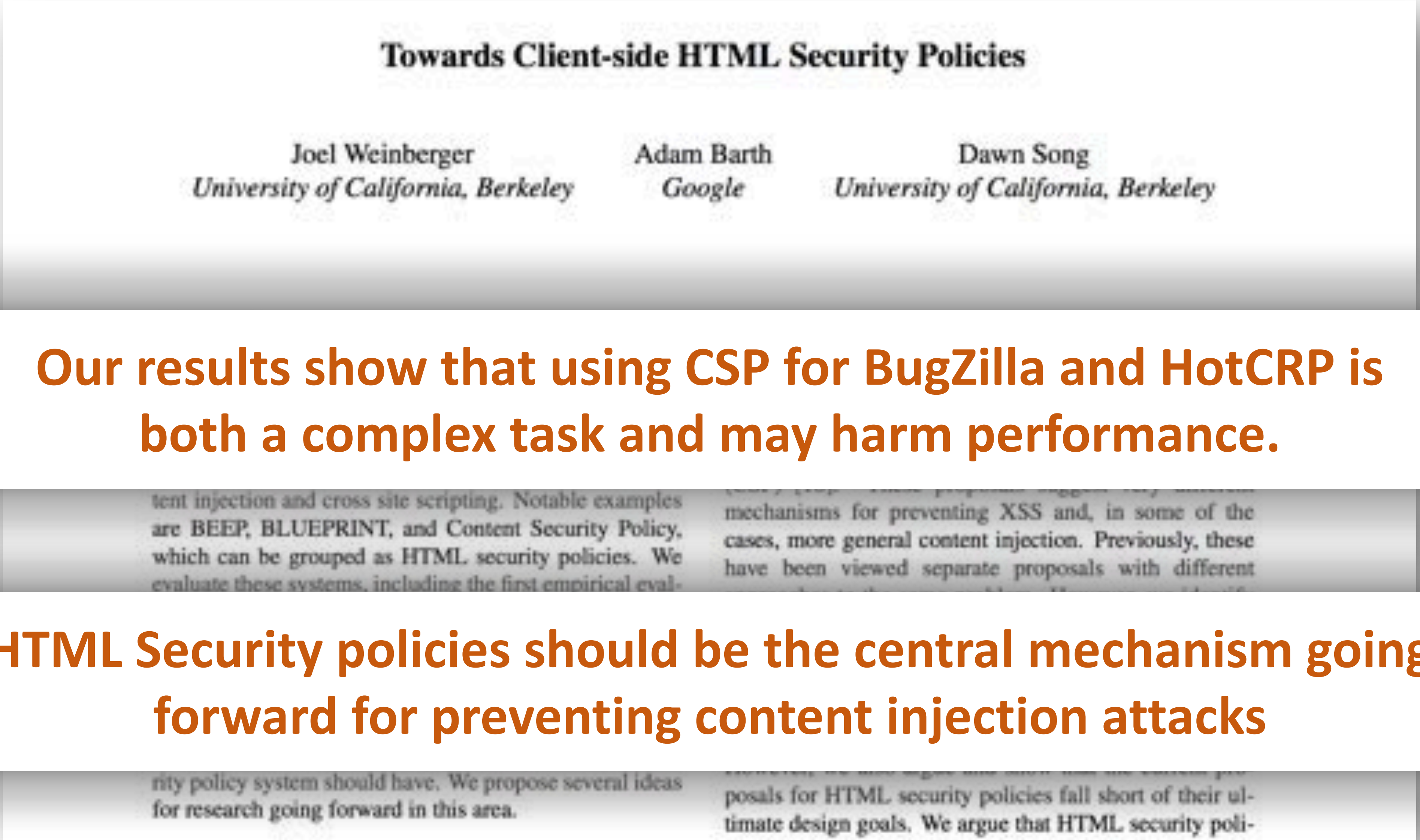
Content Security Policy 1.0 - CR

Global 88.34% + 4.67% = 93.01%

Mitigate cross-site scripting attacks by whitelisting allowed sources of script, style, and other resources.

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			49					4.3	
			54					4.4	
		50	55			9.3		4.4.4	
11	14	51	56	10	43	10.2	all	53	56
	15	52	57	10.1	44				
		53	58	TP	45				
		54	59						

http://caniuse.com/#search=content



Towards Client-side HTML Security Policies

Joel Weinberger
University of California, Berkeley

Adam Barth
Google

Dawn Song
University of California, Berkeley

Our results show that using CSP for BugZilla and HotCRP is both a complex task and may harm performance.

tent injection and cross site scripting. Notable examples are BEEP, BLUEPRINT, and Content Security Policy, which can be grouped as HTML security policies. We evaluate these systems, including the first empirical eval-

mechanisms for preventing XSS and, in some of the cases, more general content injection. Previously, these have been viewed separate proposals with different

HTML Security policies should be the central mechanism going forward for preventing content injection attacks

rity policy system should have. We propose several ideas for research going forward in this area.

posals for HTML security policies fall short of their ultimate design goals. We argue that HTML security poli-

https://www.usenix.org/legacy/events/hotsec11/tech/final_files/Weinberger.pdf

Do not Re-enable inline scripts with `unsafe-inline`

```
Content-Security-Policy:
        script-src ‘self’ http://platform.twitter.com
        https://cdn.syndication.twimg.com ‘unsafe-inline’
```

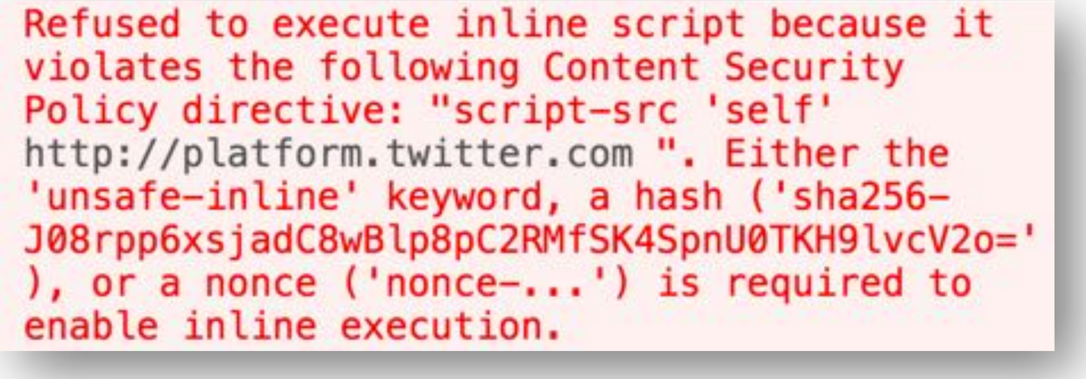
- Legacy applications are riddled with inline scripts
  - Script blocks and event handlers everywhere
- It’s tempting to use `‘unsafe-inline’` to re-enable inline script
  - But this would disable all protection against XSS attacks
- CSP level 2 allows inline script blocks using hashes and nonces
  - Only script blocks can be re-enabled, not inline event handlers

RE-ENABLING INLINE SCRIPTS WITH HASHES

```
Content-Security-Policy:
        script-src ‘self’ http://platform.twitter.com
        'sha256-qznLcsROx4GACP2dm0UCKCzCG-HiZ1guq6ZZDob_Tng='
```

- You can whitelist inline script blocks by adding their hash to the policy
  - The hash is a simple checksum of the script block’s contents
  - Chrome calculates the hash for you when it encounters a violating script block

- The use of hashes causes the browser to ignore **`unsafe-inline`**



```
Refused to execute inline script because it
violates the following Content Security
Policy directive: "script-src 'self'
http://platform.twitter.com ". Either the
'unsafe-inline' keyword, a hash ('sha256-
J08rpp6xsjadC8wBlp8pC2RMfSK4SpnU0TKH9lvcV2o='
), or a nonce ('nonce-...') is required to
enable inline execution.
```

```
<h1>My PHP app</h1>
<h3>Hi ~~<script>alert(1)</script>~~</h3>

<button ~~onclick="doSomething()"~~>
  Click me
</button>
<script>
  document.querySelector(“button”)
    .addEventListener(“click”, doSomething);

  function doSomething() { … }
</script>
<p>
  ...
  ~~<script src="http://evil.com/hackme.js"></script>~~
</p>
```

```
script-src ‘sha256-…’
```

RE-ENABLING INLINE SCRIPTS WITH NONCES

```
Content-Security-Policy:
        script-src ‘self’ http://platform.twitter.com
        https://cdn.syndication.twimg.com 'nonce-EDNnf03nceIOfn39fn3e9h3sdfa'
```

- Nonces mark inline script blocks as trusted
  - – The server needs to add a random nonce to the policy and to the script blocks
  - – The nonce should be freshly generated on every request
  - – The attacker will not be able to predict the nonce, so injected script will be ignored

- The use of nonces causes the browser to ignore `unsafe-inline`

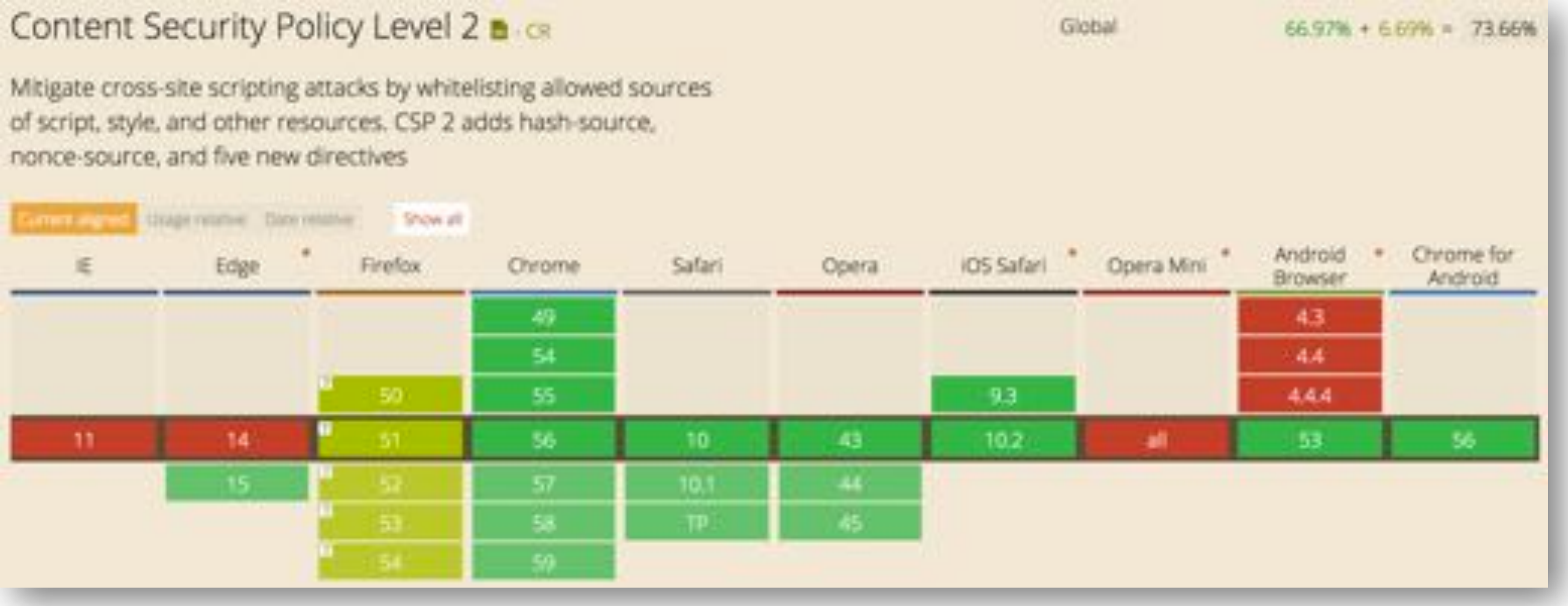
```
<script nonce=”EDNnf03nceIOfn39fn3e9h3sdfa”>…</script>
```

```
<h1>My PHP app</h1>
<h3>Hi ~~<script>alert(1)</script>~~</h3>

<button ~~onclick="doSomething()"~~>
  Click me
</button>
<script nonce=“aT1a32n4SA”>
  document.querySelector(“button”)
    .addEventListener(“click”, doSomething);

  function doSomething() { … }
</script>
<p>
  ...
  ~~<script src="http://evil.com/hackme.js"></script>~~
</p>
```

```
script-src 'unsafe-eval' https://www.dropbox.com/static/compiled/js/
https://www.dropbox.com/static/javascript/ https://www.dropbox.com/static/api/
https://cfl.dropboxstatic.com/static/compiled/js/
https://www.dropboxstatic.com/static/compiled/js/ https://cfl.dropboxstatic.com/static/javascript/
https://www.dropboxstatic.com/static/javascript/ https://cfl.dropboxstatic.com/static/api/
https://www.dropboxstatic.com/static/api/ 'unsafe-inline' 'nonce-EtRYI0CtY17XHMVxdxsV' ;
default-src 'none' ;
worker-src blob: ;
style-src https://* 'unsafe-inline' 'unsafe-eval' ; connect-src https://* ws://127.0.0.1:*/ws ;
child-src https://www.dropbox.com/static/serviceworker/ blob: ;
form-action 'self' https://dl-web.dropbox.com/ https://photos.dropbox.com/
https://accounts.google.com/ https://api.login.yahoo.com/ https://login.yahoo.com/ ; base-uri
'self' api-stream.dropbox.com https://showbox-tr.dropbox.com ;
img-src https://* data: blob: ; report-uri https://www.dropbox.com/log/csp_enforced ;
frame-src https://* carousel://* dbapi-6://* dbapi-7://* dbapi-8://* itms-apps://* itms-appss://*
;
object-src https://cfl.dropboxstatic.com/static/ https://www.dropboxstatic.com/static/ 'self'
https://flash.dropboxstatic.com https://swf.dropboxstatic.com https://dbxlocal.dropboxstatic.com ;
media-src https://* blob: ;
font-src https://* data:
```

https://blogs.dropbox.com/tech/tag/content-security-policy/

BROWSER SUPPORT – CONTENT SECURITY POLICY LEVEL 2



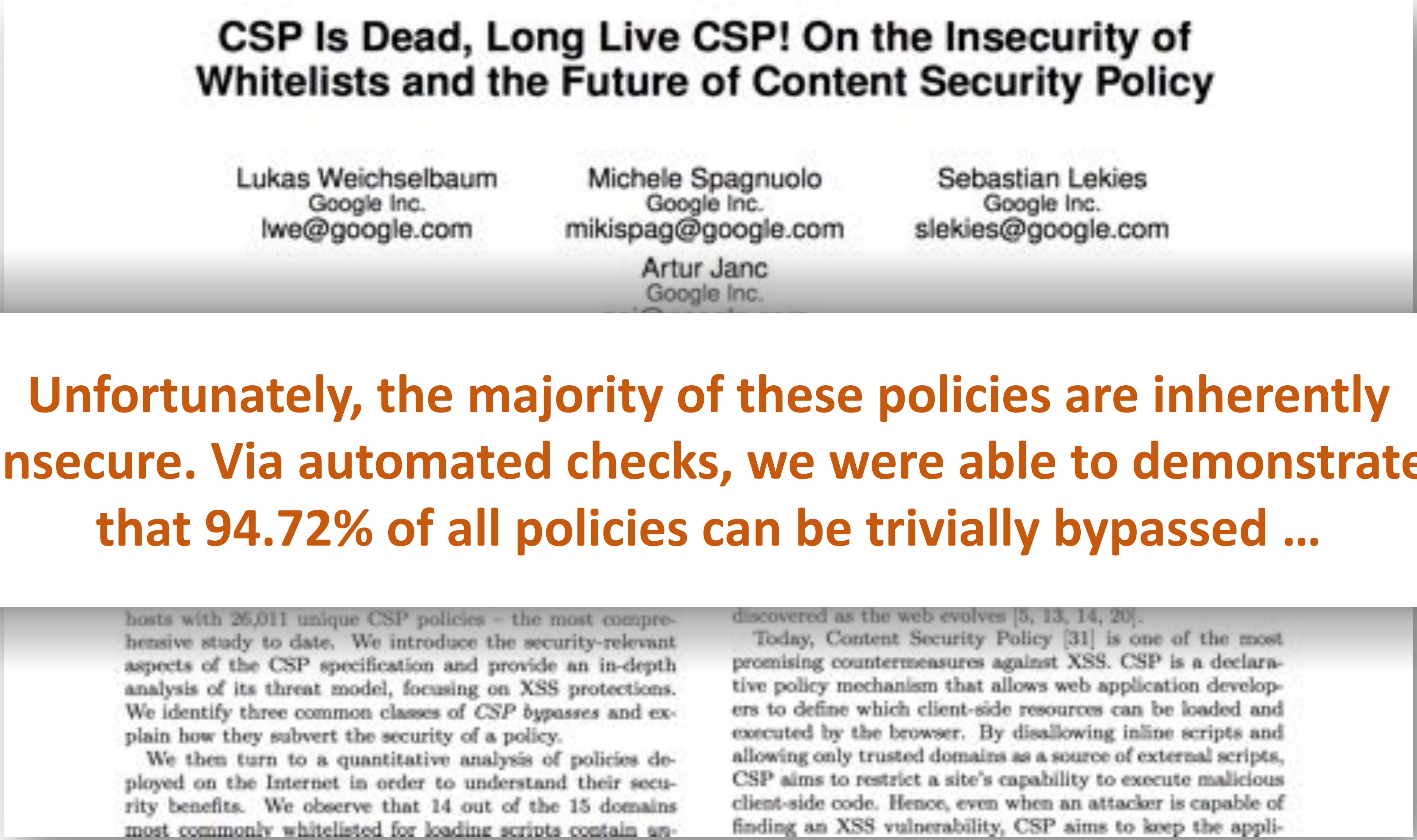
Content Security Policy Level 2 - CR

Global 66.97% + 6.69% = 73.66%

Mitigate cross-site scripting attacks by whitelisting allowed sources of script, style, and other resources. CSP 2 adds hash-source, nonce-source, and five new directives

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			49					4.3	
			54					4.4	
		50	55			9.3		4.4.4	
11	14	51	56	10	43	10.2	all	53	56
	15	52	57	10.1	44				
		53	58	TP	45				
		54	59						

http://caniuse.com/#search=csp



CSP Is Dead, Long Live CSP! On the Insecurity of Whitelists and the Future of Content Security Policy

Lukas Weichselbaum
Google Inc.
lwe@google.com

Michele Spagnuolo
Google Inc.
mikispag@google.com

Sebastian Lekies
Google Inc.
slekies@google.com

Artur Janc
Google Inc.

Unfortunately, the majority of these policies are inherently insecure. Via automated checks, we were able to demonstrate that 94.72% of all policies can be trivially bypassed …

hosts with 26,011 unique CSP policies – the most comprehensive study to date. We introduce the security-relevant aspects of the CSP specification and provide an in-depth analysis of its threat model, focusing on XSS protections. We identify three common classes of *CSP bypasses* and explain how they subvert the security of a policy.

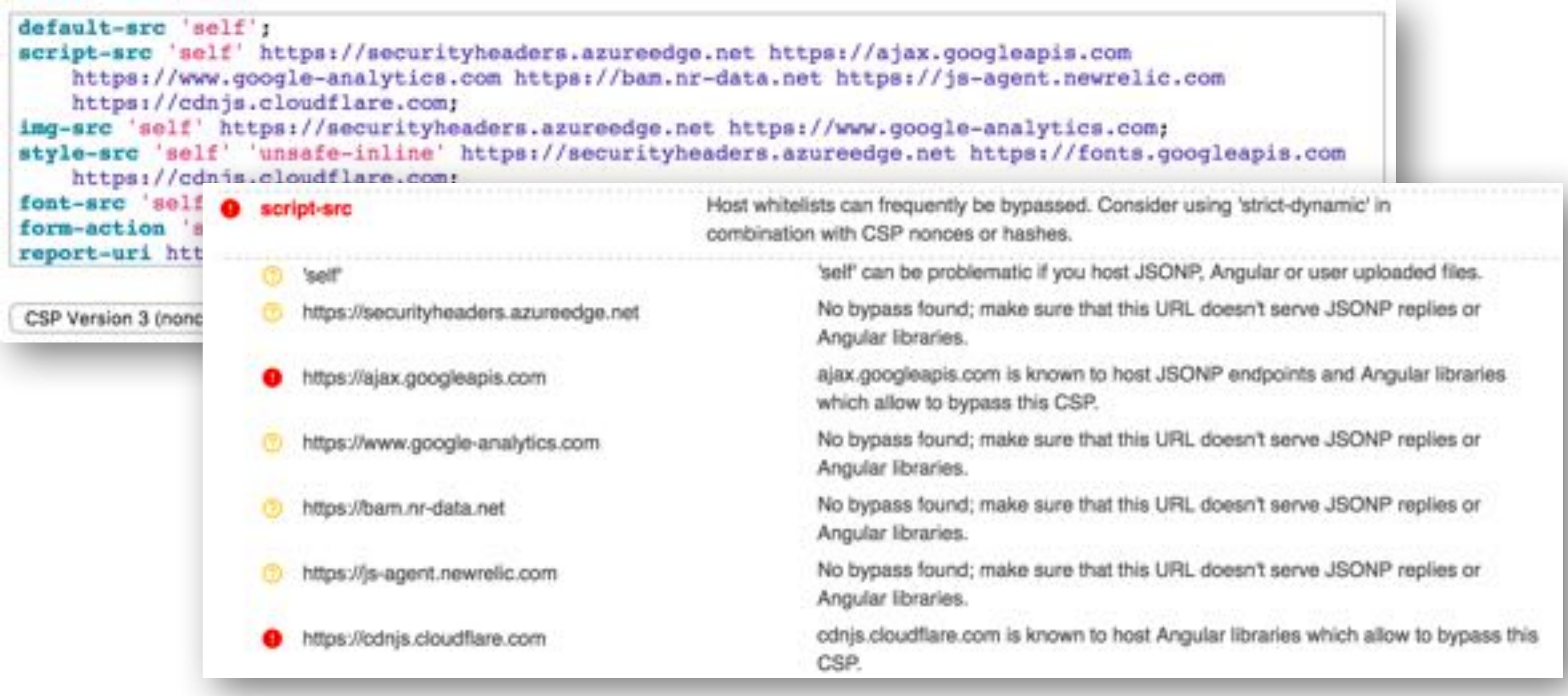
We then turn to a quantitative analysis of policies deployed on the Internet in order to understand their security benefits. We observe that 14 out of the 15 domains most commonly whitelisted for loading scripts contain un-

discovered as the web evolves [5, 13, 14, 20].

Today, Content Security Policy [31] is one of the most promising countermeasures against XSS. CSP is a declarative policy mechanism that allows web application developers to define which client-side resources can be loaded and executed by the browser. By disallowing inline scripts and allowing only trusted domains as a source of external scripts, CSP aims to restrict a site's capability to execute malicious client-side code. Hence, even when an attacker is capable of finding an XSS vulnerability, CSP aims to keep the appli-

http://delivery.acm.org/10.1145/2980000/2978363/p1376-weichselbaum.pdf

BUT HOW SECURE IS YOUR CSP POLICY REALLY?



```
default-src 'self';
script-src 'self' https://securityheaders.azureedge.net https://ajax.googleapis.com
    https://www.google-analytics.com https://bam.nr-data.net https://js-agent.newrelic.com
    https://cdnjs.cloudflare.com;
img-src 'self' https://securityheaders.azureedge.net https://www.google-analytics.com;
style-src 'self' 'unsafe-inline' https://securityheaders.azureedge.net https://fonts.googleapis.com
    https://cdnjs.cloudflare.com;
font-src 'self
form-action 's
report-uri htt
```

CSP Version 3 (nonc

●	script-src	Host whitelists can frequently be bypassed. Consider using 'strict-dynamic' in combination with CSP nonces or hashes.
○	'self'	'self' can be problematic if you host JSONP, Angular or user uploaded files.
○	https://securityheaders.azureedge.net	No bypass found; make sure that this URL doesn't serve JSONP replies or Angular libraries.
●	https://ajax.googleapis.com	ajax.googleapis.com is known to host JSONP endpoints and Angular libraries which allow to bypass this CSP.
○	https://www.google-analytics.com	No bypass found; make sure that this URL doesn't serve JSONP replies or Angular libraries.
○	https://bam.nr-data.net	No bypass found; make sure that this URL doesn't serve JSONP replies or Angular libraries.
○	https://js-agent.newrelic.com	No bypass found; make sure that this URL doesn't serve JSONP replies or Angular libraries.
●	https://cdnjs.cloudflare.com	cdnjs.cloudflare.com is known to host Angular libraries which allow to bypass this CSP.

https://csp-evaluator.withgoogle.com/

Common mistakes and bypass attacks

Missing object-src (or default-src)

```
script-src ‘self’
```

```
<object type=“application/x-
shockwave-flash” data=“URL with
reflected XSS in parameter”><param
name=“AllowScriptAccess”
value=“always”></object>
```

Combining ‘self’ with uploads

```
script-src ‘self’;
object-src ‘none’
```

```
<script
src=“user_upload/evil_cat.jpg.js”>
</script>
```

Whitelist bypass with JSONP

```
script-src ‘self’ https://whitelist.cdn.com
```

```
<script src=“https://whitelist.cdn.com/
jsonp?callback=alert”>
```

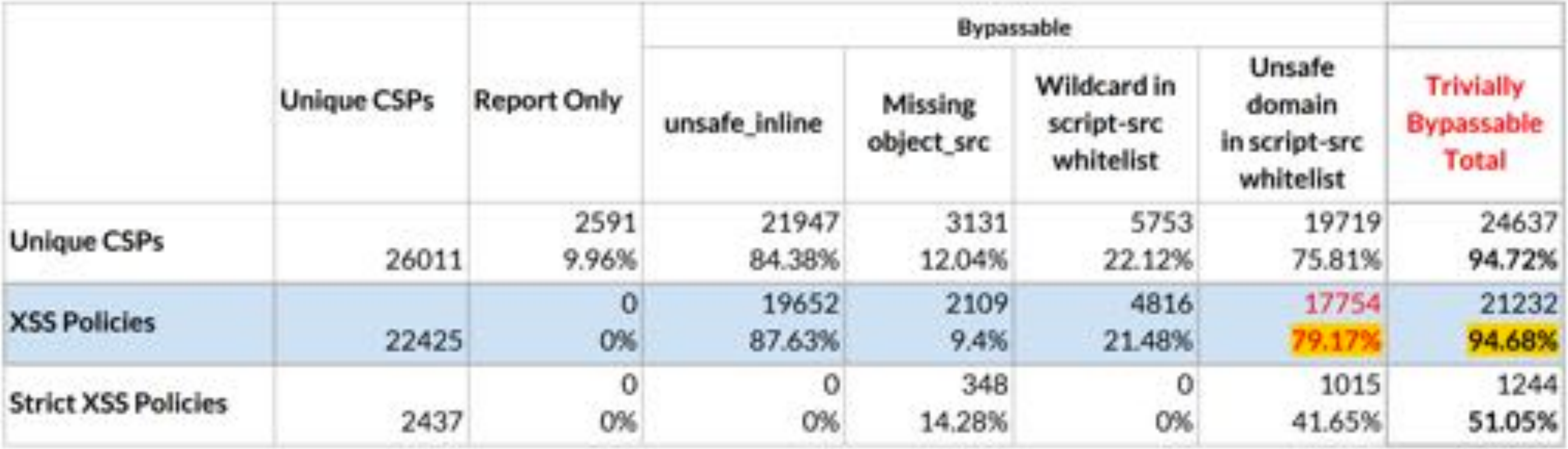
Whitelist bypass with AngularJS

```
script-src ‘self’ https://whitelist.cdn.com
```

```
<script src=“https://whitelist.cdn.com/angular.js”>
<div ng-app ng-csp ng-
click=“$event.view.alert(1337)”></div>
```

https://speakerdeck.com/mikispag/making-csp-great-again-michele-spagnuolo-and-lukas-weichselbaum

IT TURNS OUT ALMOST NOBODY GETS CSP RIGHT



			Bypassable				
	Unique CSPs	Report Only	unsafe_inline	Missing object_src	Wildcard in script-src whitelist	Unsafe domain in script-src whitelist	Trivially Bypassable Total
Unique CSPs	26011	2591 9.96%	21947 84.38%	3131 12.04%	5753 22.12%	19719 75.81%	24637 **94.72%**
XSS Policies	22425	0 0%	19652 87.63%	2109 9.4%	4816 21.48%	17754 79.17%	21232 **94.68%**
Strict XSS Policies	2437	0 0%	0 0%	348 14.28%	0 0%	1015 41.65%	1244 **51.05%**

https://speakerdeck.com/mikispag/acm-ccs-2016-csp-is-dead-long-live-csp

How Google proposes to fix CSP

```
Content-Security-Policy:
        script-src 'nonce-{random}' 'strict-dynamic'
```

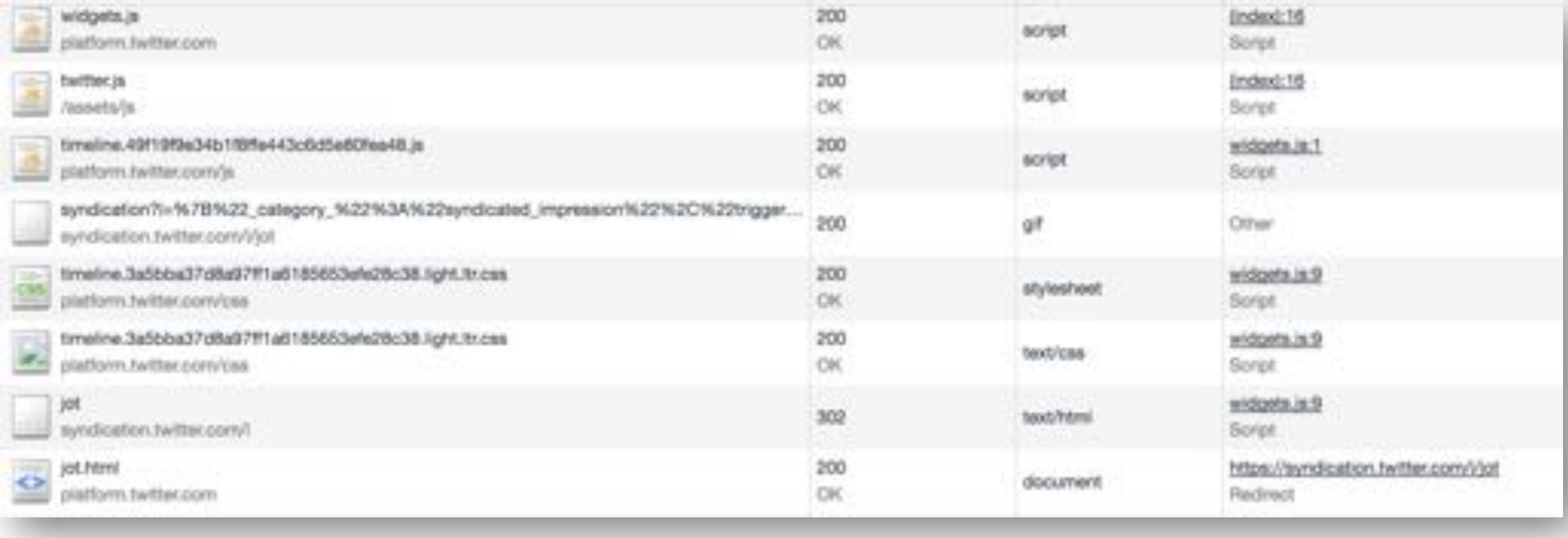
- Google tried to use CSP with whitelists, but it just doesn’t work
  - Cascading script loading makes them too hard to maintain
  - Too difficult to lock down a whitelist against bypass attacks

- With ‘strict-dynamic’, trusted scripts can dynamically load additional scripts
  - This trust propagation makes sense, as the trusted script already has full access
  - ’strict-dynamic’ only applies to scripts being loaded via DOM APIs
  - Parser-inserted script (e.g. document.write) will still be blocked

- This limits the attack surface to the use of DOM APIs
  - This is a lot easier to check for during a security review

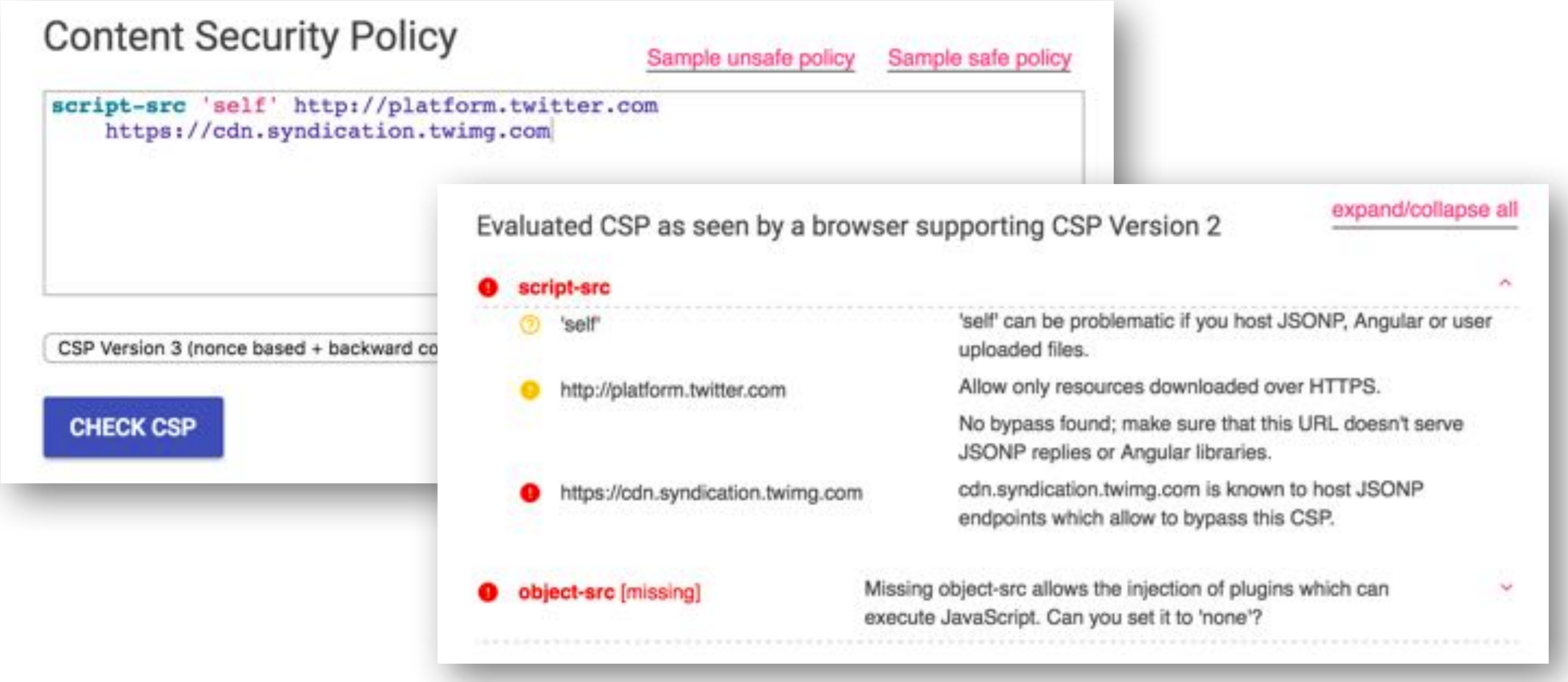
https://csp.withgoogle.com/docs/strict-csp.html

```
 <a class="twitter-timeline"  href="https://twitter.com/PhilippeDeRyck" data-widget-
id="697784323848736772">Tweets by @PhilippeDeRyck</a>

<script nonce="j08AD4S8zH">
!function(d,s,id){var
js,fjs=d.getElementsByTagName(s)[0],p=/^http:/.test(d.location)?'http':'https';if(!d
.getElementById(id)){js=d.createElement(s);js.id=id;js.src=p+"://platform.twitter.co
m/widgets.js";fjs.parentNode.insertBefore(js,fjs);}}(document,"script","twitter-
wjs");
</script>
```



widgets.js platform.twitter.com	200 OK	script	(index):16 Script
twitter.js /assets/js	200 OK	script	(index):16 Script
timeline.49f19f9e34b1f8ffe443c6d5e80fea48.js platform.twitter.com/js	200 OK	script	widgets.js:1 Script
syndication?l=%7B%22_category_%22%3A%22syndicated_impression%22%2C%22trigger... syndication.twitter.com/i/jot	200	gif	Other
timeline.3a5bba37d8a97ff1a6185653efe28c38.light.ltr.css platform.twitter.com/css	200 OK	stylesheet	widgets.js:9 Script
timeline.3a5bba37d8a97ff1a6185653efe28c38.light.ltr.css platform.twitter.com/css	200 OK	text/css	widgets.js:9 Script
jot syndication.twitter.com/i	302	text/html	widgets.js:9 Script
jot.html platform.twitter.com	200 OK	document	https://syndication.twitter.com/i/jot Redirect

Whitelisting these hosts is not a good idea



Content Security Policy

```
script-src 'self' http://platform.twitter.com
    https://cdn.syndication.twimg.com
```

CSP Version 3 (nonce based + backward co

Evaluated CSP as seen by a browser supporting CSP Version 2

script-src

- 'self' — 'self' can be problematic if you host JSONP, Angular or user uploaded files.
- http://platform.twitter.com — Allow only resources downloaded over HTTPS. No bypass found; make sure that this URL doesn't serve JSONP replies or Angular libraries.
- https://cdn.syndication.twimg.com — cdn.syndication.twimg.com is known to host JSONP endpoints which allow to bypass this CSP.

object-src [missing] — Missing object-src allows the injection of plugins which can execute JavaScript. Can you set it to 'none'?

https://csp-evaluator.withgoogle.com/

TRUST PROPAGATION ACTUALLY MAKES A LOT OF SENSE

- We have already trusted Twitter to run code in our context
  - If it needs additional resources, we are very likely to allow them to be loaded
  - `strict-dynamic` simply makes this implicit trust explicit through trust propagation

- Trusted scripts can load resources through appropriate APIs

```
 <a class="twitter-timeline"  href="https://twitter.com/PhilippeDeRyck" data-widget-
id="697784323848736772">Tweets by @PhilippeDeRyck</a>

<script nonce=“j08AD4S8zH”>
!function(d,s,id){var
js,fjs=d.getElementsByTagName(s)[0],p=/^http:/.test(d.location)?'http':'https';if(!d.getElemen
tById(id)){js=d.createElement(s);js.id=id;js.src=p+"://platform.twitter.com/widgets.js";fjs.pa
rentNode.insertBefore(js,fjs);}}(document,"script","twitter-wjs");
</script>
```

A universal CSP policy against XSS attacks

```
Content-Security-Policy:
  object-src 'none';
  script-src 'nonce-{random}' 'unsafe-inline' 'unsafe-eval' 'strict-dynamic' https: http:;
  report-uri https://your-report-collector.example.com/
```

- Inline scripts and remote scripts are marked as trusted with a nonce
  - Subsequent script-loading operations are enabled through ‘strict-dynamic’
  - If no plugins (flash / java) are loaded, object-src should be set to ‘none’

- The other expressions enable compatibility with non-compliant browsers
  - Because of the nonces, modern browsers ignore `unsafe-inline`
  - Because of ’strict-dynamic’, modern browsers ignore the whitelist (https: / http:)

- This policy only protects you if you run a modern browser
  - But on an older browser, it still works as before

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

FROM ‘STRICT-DYNAMIC’ TO A UNIVERSAL CSP

```
Content-Security-Policy:
  object-src 'none';
  script-src ’nonce-{random}' 'strict-dynamic' 'unsafe-inline' 'unsafe-eval' https: http:;
  report-uri https://your-report-collector.example.com/
```

✓ **Remote**
✓ **Inline**

```
Content-Security-Policy:
  object-src 'none';
  script-src ’nonce-{random}’ 'strict-dynamic’ 'unsafe-eval';
  report-uri https://your-report-collector.example.com/
```

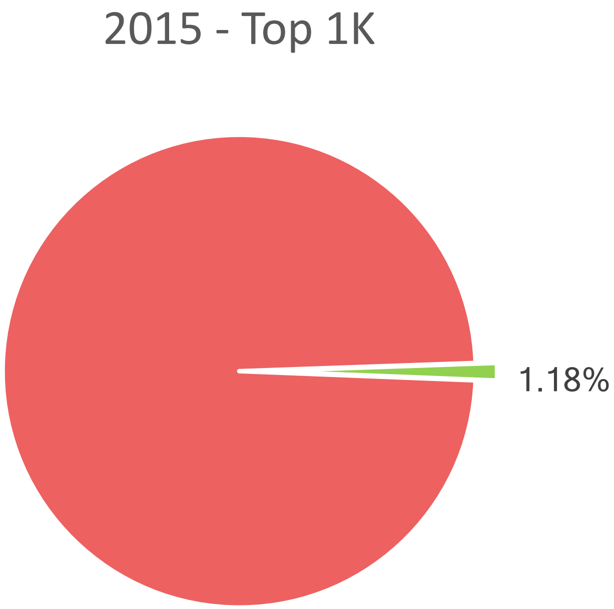
✗ **Remote**
✓ **Inline**

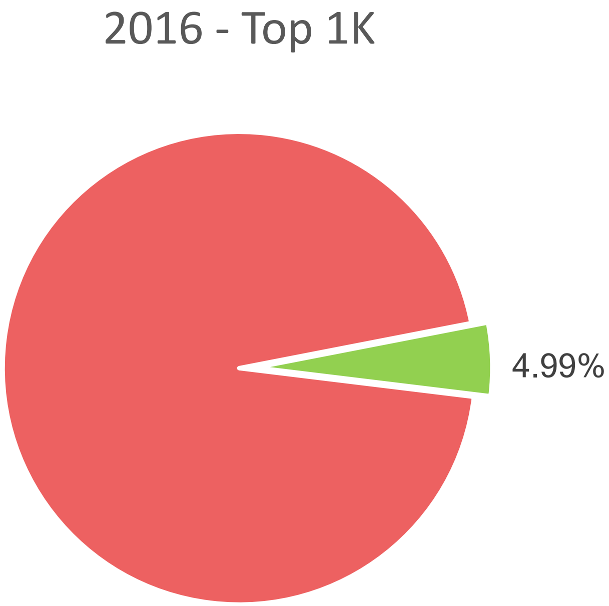
```
Content-Security-Policy:
  object-src 'none';
  script-src ’nonce-{random}’ 'unsafe-eval' https: http:;
  report-uri https://your-report-collector.example.com/
```

✗ **Remote**
✗ **Inline**

```
Content-Security-Policy:
  object-src 'none';
  script-src 'unsafe-inline' 'unsafe-eval' https: http:;
  report-uri https://your-report-collector.example.com/
```

CSP USAGE STATISTICS





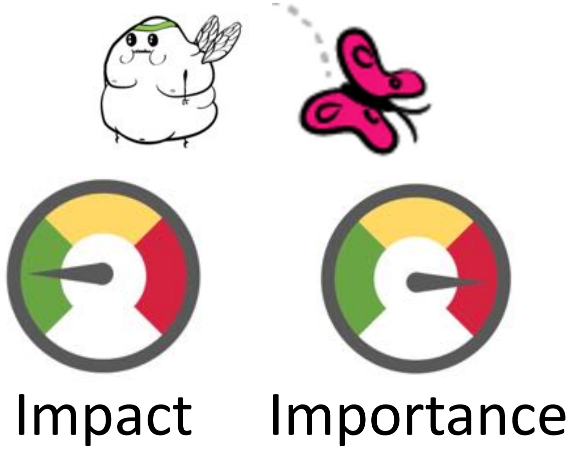
https://www.owasp.org/index.php/OWASP_Secure_Headers_Project#tab=Stats

Focusing your efforts in 2017

A BIG DIFFERENCE BETWEEN EXISTING AND NEW SYSTEMS

HTTPS AS A SECURITY BASELINE

- HTTPS is considered mandatory for all web applications
  - Sensitive features are only available to *Secure Contexts*
- All communication should happen over HTTPS, with HSTS enabled
  - Should be easy if HTTPS is already in place
  - Recommended to apply HSTS to all subdomains as well
  - Recommended to preload HSTS
- HPKP is probably overkill for you
  - Getting it right is more difficult than it seems
  - HPKP is also dangerous when you get it wrong

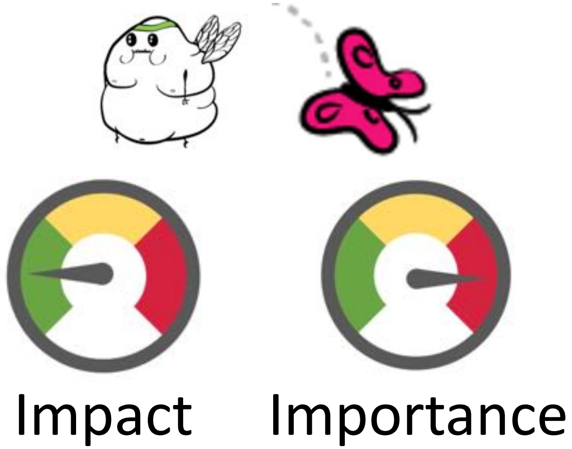


COOKIES SHOULD BE PROPERLY PROTECTED

- Basic cookie security features have been around for a while
  - All cookies should be marked *Secure* (HTTPS is a baseline requirement)
  - Most cookies can be marked as *HttpOnly*

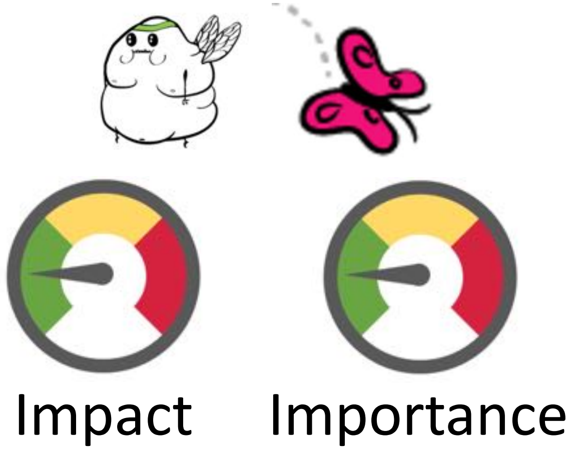
- Recently, two new security features have been proposed
  - *SameSite* helps prevent CSRF attacks
  - *Cookie prefixes* enable additional browser protections

- Browser support is a bit limited, but it will pick up
  - Enabling these features now future-proofs your cookies



Browser-based XSS protection should be enabled

- Rule of thumb: never leave it default
  - Either turn it off, or enable blocking mode
- Issues with X-XSS-Protection are very limited
  - Turning it off is only OK if you are 100% sure that you do not have reflected XSS
  - This requires a lot of discipline, and separation between data and code
- In general, blocking mode is the way to go



VERIFY WHAT YOU'RE LETTING IN YOUR CONTEXT

- Subresource Integrity allows you to verify script files and style sheets
  - Prevents the loading of malicious code by verifying its checksum
  - Small amount of effort to add checksums manually
  - Build systems are capable of doing this automatically

- Many CDNs are compatible with SRI
  - Requires basic support for Cross-Origin Resource Sharing

- If you offer public libraries, make sure SRI works for them
  - Enable the appropriate CORS headers

Impact

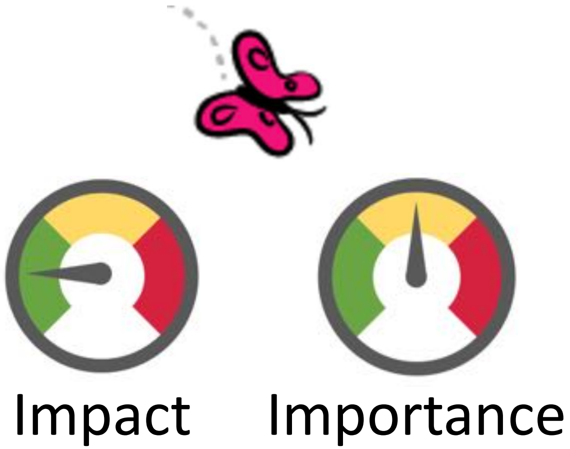
Importance

VERIFY WHAT YOU'RE LETTING IN YOUR CONTEXT

- Subresource Integrity allows you to verify script files and style sheets
  - Prevents the loading of malicious code by verifying its checksum
  - Small amount of effort to add checksums manually
  - Build systems are capable of doing this automatically

- Many CDNs are compatible with SRI
  - Requires basic support for Cross-Origin Resource Sharing

- If you offer public libraries, make sure SRI works for them
  - Enable the appropriate CORS headers

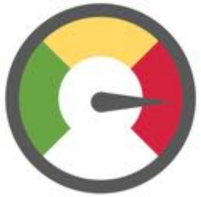


Restrict what's already loaded in your context

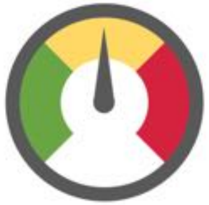
- Content Security Policy controls what resources can be loaded
  - – Disallows inline code / style, which has a significant impact.
  - – Restricts the default *allow-all* policy from the browser

- CSP has evolved a lot since the first version
  - – Nonces and hashes re-enable inline scripts
  - – Strict-dynamic makes CSP very useful
  - – Additional directives have been added

- CSP will become even more important in the future
  - – Therefore, compatibility with CSP is really important



Impact



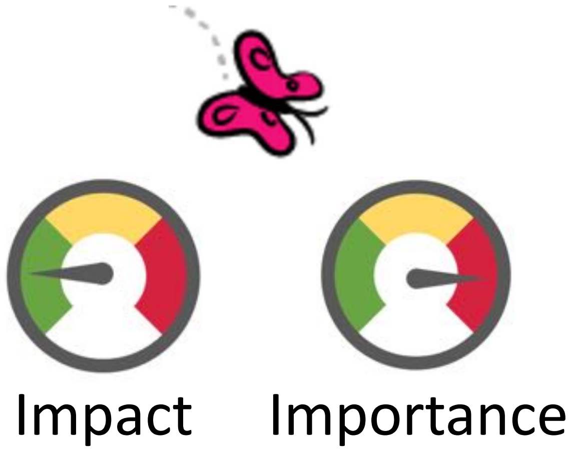
Importance

Restrict what’s already loaded in your context

- Content Security Policy controls what resources can be loaded
  - Disallows inline code / style, which has a significant impact.
  - Restricts the default *allow-all* policy from the browser

- CSP has evolved a lot since the first version
  - Nonces and hashes re-enable inline scripts
  - Strict-dynamic makes CSP very useful
  - Additional directives have been added

- CSP will become even more important in the future
  - Therefore, compatibility with CSP is really important

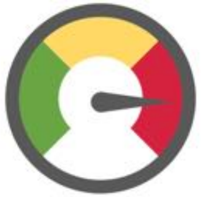


PRIVILEGE SEPARATION BY DESIGN WITH A SECURE ARCHITECTURE

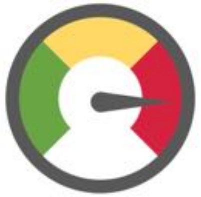
- The best approach for security is building a client-side architecture
  - The Same-Origin Policy is the default security policy of the browser
  - Additional building blocks allow you to build security into the design

- Frontend development is more than simply coding some JavaScript

- SecAppDev covered numerous topics to support this
  - Essential web security concepts
  - Threat modeling and SDLC activities
  - Access control concepts
  - …



Impact



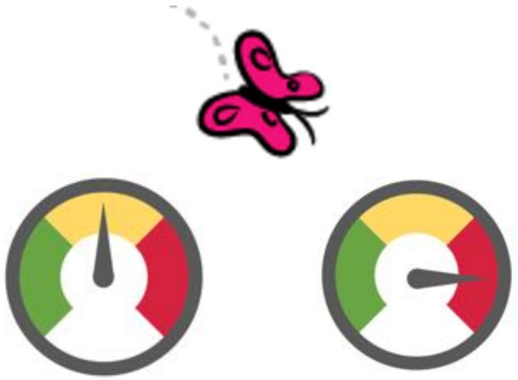
Importance

Privilege separation by design with a secure architecture

- The best approach for security is building a client-side architecture
  - The Same-Origin Policy is the default security policy of the browser
  - Additional building blocks allow you to build security into the design

- Frontend development is more than simply coding some JavaScript

- SecAppDev covered numerous topics to support this
  - Essential web security concepts
  - Threat modeling and SDLC activities
  - Access control concepts
  - …



WHAT YOU SHOULD TAKE AWAY FROM THIS TALK / COURSE

- Building secure applications requires a conscious effort
  - Like any other application, web applications require a well thought-out architecture
  - An important part of that architecture resides in the front end nowadays

- A modern developer's toolbox is full of security tools
  - Frameworks, protocols and browsers offer good security features
  - But they require knowledge to handle them correctly

- The focus of this talk was front end security
  - Front end and back end security are complementary
  - But front end security is worthless without solid back end security

Now it’s up to you …

Secure

Follow

Share

Web Security Essentials

April 24 – 25, Leuven, Belgium

https://essentials.websec.be

@PhilippeDeRyck https://www.websec.be philippe.deryck@cs.kuleuven.be /in/philippederyck