



You Can't Always Get What You Want

How Web Sites (Often) Lack Consistent Protection

Sebastian Roth and Ben Stock



Sebastian Roth
Postdoc @ TU Vienna



Ben Stock
Faculty @ CISP



Sebastian Roth
Postdoc @ TU Vienna

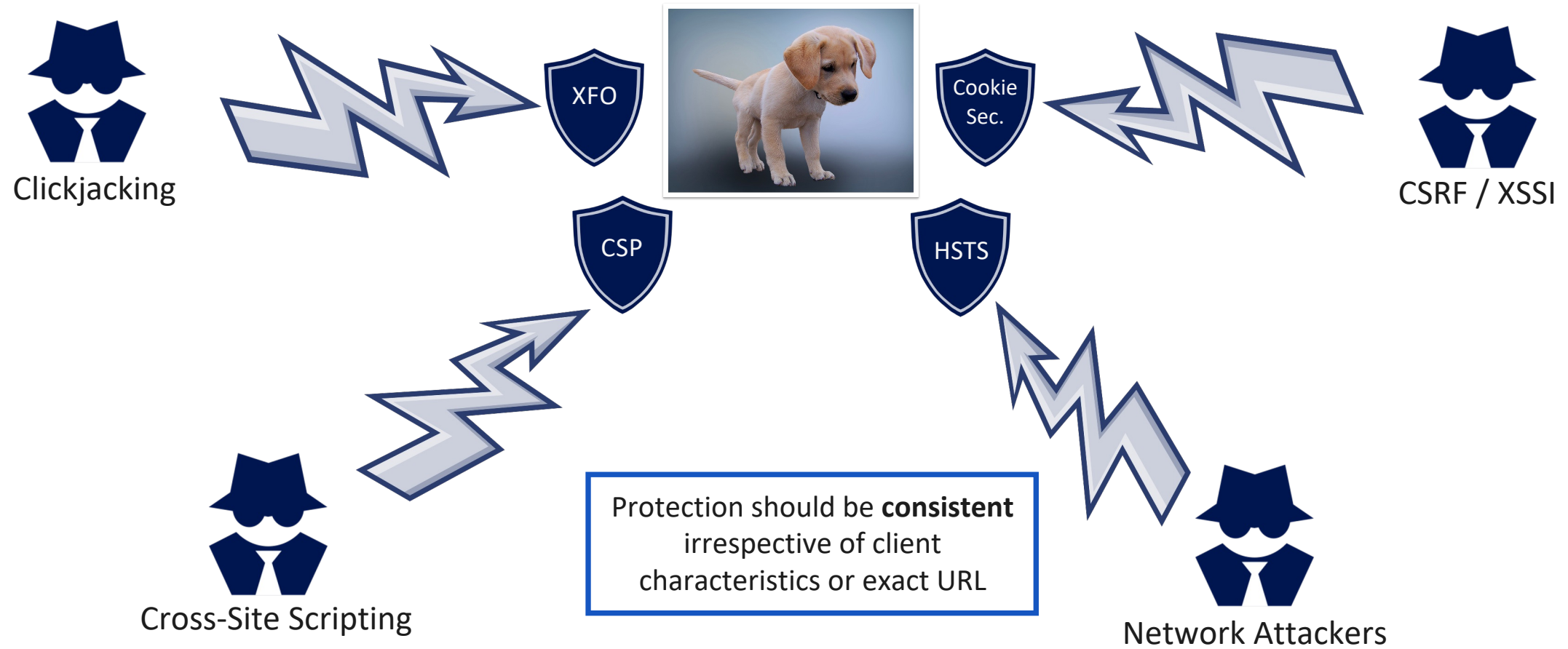


Ben Stock
Faculty @ CISPA



**This talk
requires
interaction!**

Motivation



Such science, much wow

USENIX Security 2022

The Security Lottery: Measuring Client-Side Web Security Inconsistencies

Sebastian Roth[†], Stefano Calzavara[‡], Moritz Wilhelm[†], Alvise Rabitti[‡], Ben Stock[†]

{sebastian.roth,moritz.wilhelm,stock}@cispa.de; {stefano.calzavara,alvise.rabitti}@unive.it

[†] CISPA Helmholtz Center for Information Security [‡] Università Ca' Foscari Venezia

NDSS 2021

Reining in the Web's Inconsistencies with Site Policy

Stefano Calzavara^{*}, Tobias Urban^{†‡}, Dennis Tatang[‡], Marius Steffens[§], and Ben Stock[§]

^{}Università Ca' Foscari Venezia: calzavara@dais.unive.it*

[†]Institute for Internet Security: urban@internet-sicherheit.de

[‡]Ruhr University Bochum: dennis.tatang@rub.de

[§]CISPA Helmholtz Center for Information Security: {marius.steffens,stock}@cispa.saarland

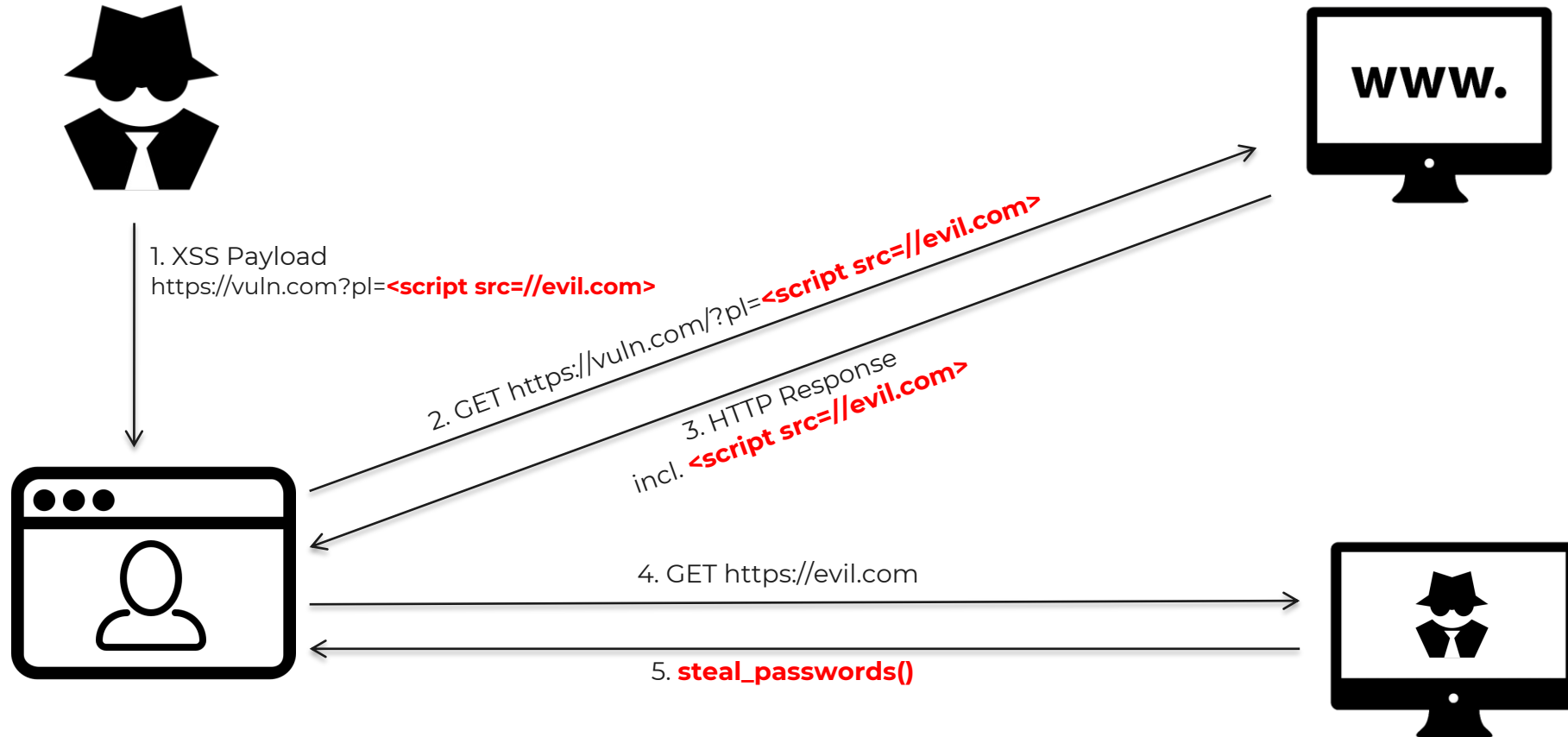
Measuring security inconsistencies in
deployed security headers



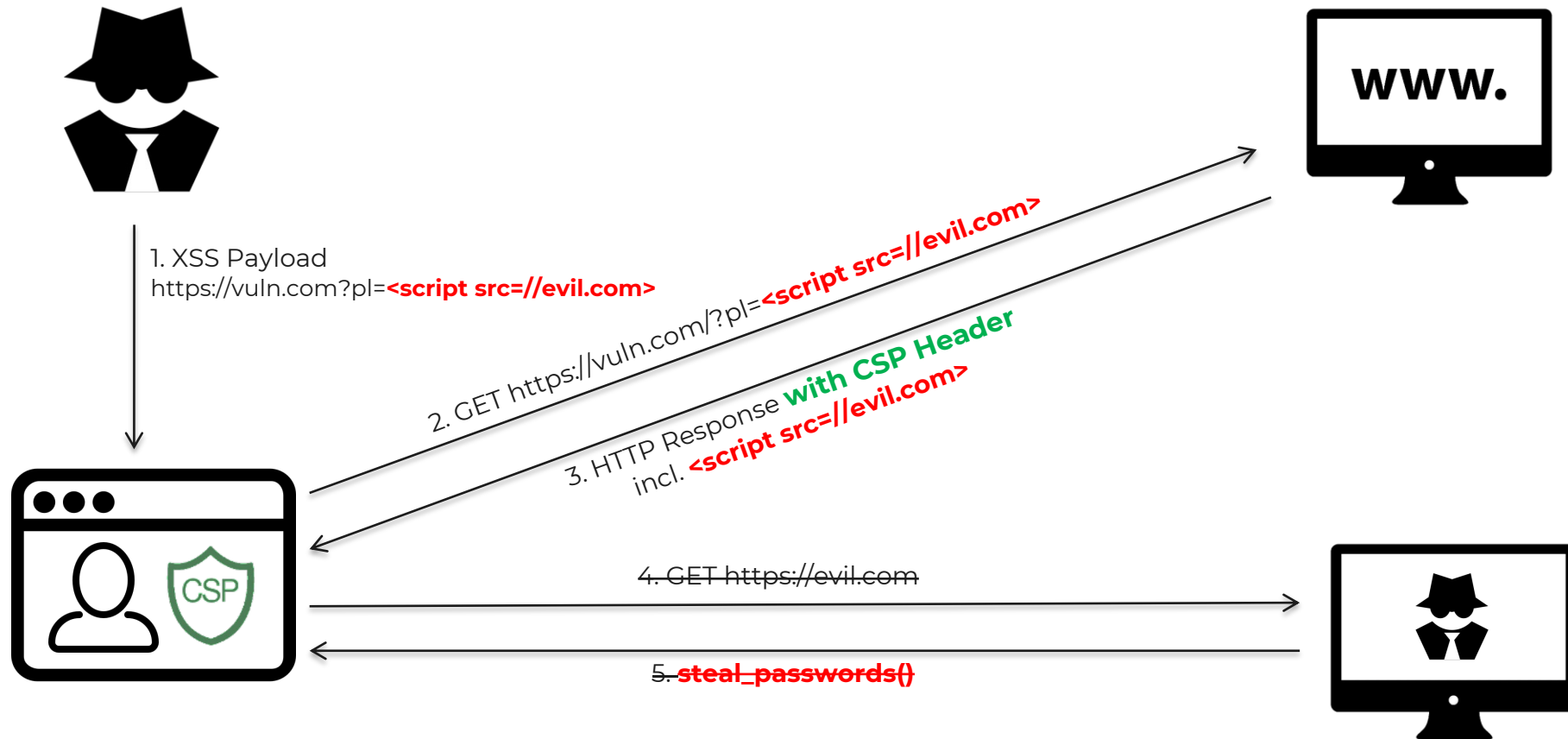
Problems of the Origin Policy and how
we can improve it with “Site Policy”



Recap: Cross-Site Scripting (XSS)



Recap: Content Security Policy (CSP)



Recap: Content Security Policy (CSP)

Example Content Security Policy Header:

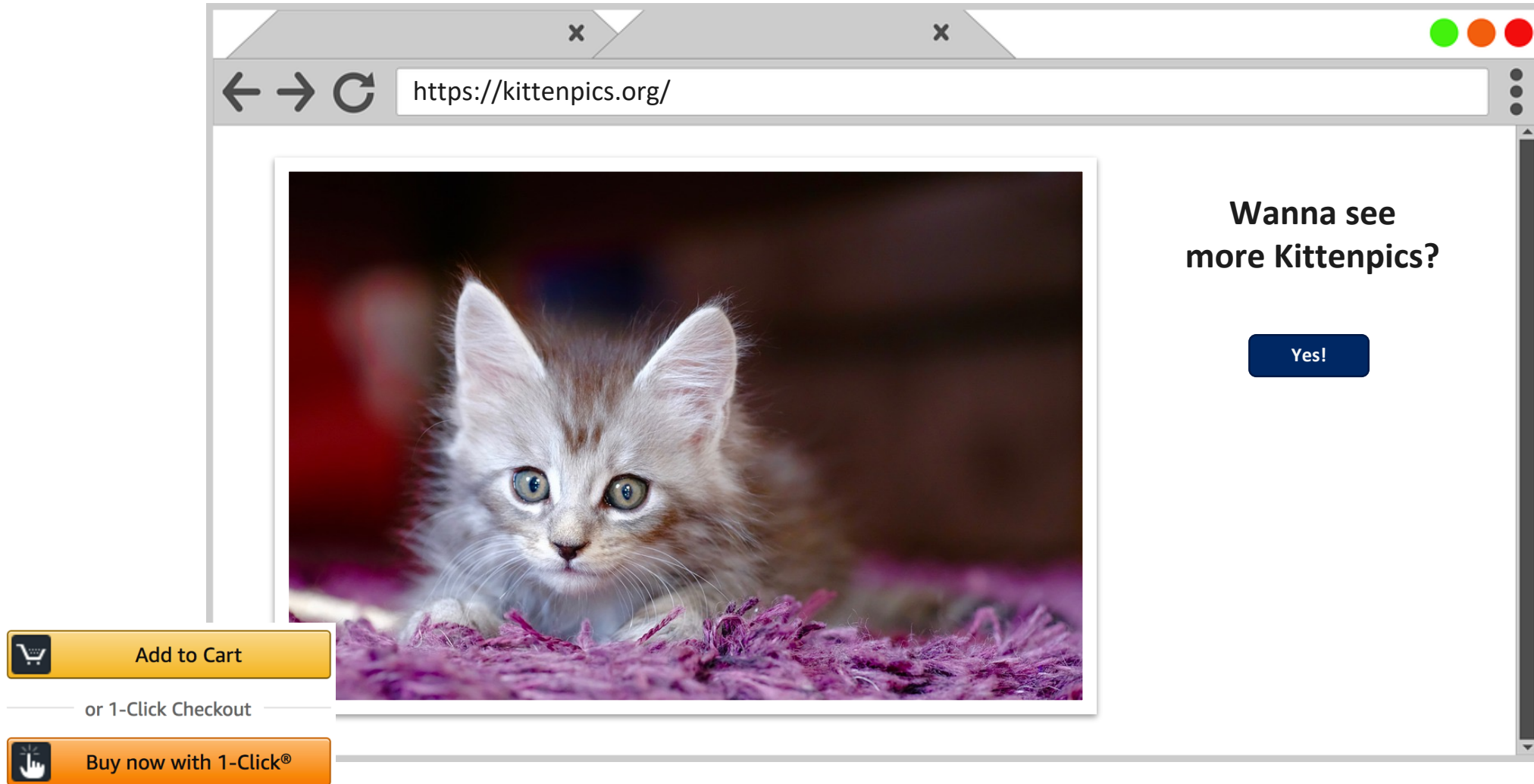
Content-Security-Policy:

```
script-src
    'self'
    advertisement.com
    'nonce-a7b4f9420'
    'sha256-3i[...]FQ=';
```



Let's talk about cat pictures

Recap: Clickjacking



Recap: X-Frame-Options (XFO)

Example X-Frame-Options Headers:

X-Frame-Options: DENY



No one can load me

X-Frame-Options: SAMEORIGIN



Only I can load myself

X-Frame-Options: ALLOW-FROM partnersite.com;



Only my partner can load me

XFO is deprecated since summer 2014!

Recap: CSP (again)

Example Content Security Policy Header:

Content-Security-Policy:

```
script-src  
  'self'  
  advertisement.com  
  'nonce-a7b4f9420'  
  'sha256-3i[...]FQ=';
```

```
frame-ancestors  
  partnersite.com;
```

XSS Mitigation

Framing Control

Recap: Strict Transport Security (HSTS)

Example Strict Transport Security Header:

Strict-Transport-Security:

max-age=63072000;

includeSubDomains;

preload;

}

Time until expiry

}

Apply to all subdomains

}

Inclusion in the preload list^[1]

^[1] <https://hstspreload.org/>

Recap: CSP (again, and again)

Example Content Security Policy Header:

Content-Security-Policy:

```
script-src  
  'self'  
  advertisement.com  
  'nonce-a7b4f9420'  
  'sha256-3i[...]FQ=';
```

```
frame-ancestors  
  partnersite.com;
```

```
upgrade-insecure-requests;
```

XSS mitigation

Framing control

TLS enforcement

Recap: Set-Cookie

Example Set-Cookie Header (with security attributes):

Set-Cookie:

session-id=r4nd0m5tr1ng;

HttpOnly;

Secure;

SameSite=

Strict;

Lax;

None;

] Key / value pair that is stored

] Not accessible via JS (XSS)

] Only sent via HTTPS (MITM)

] Only sent for

.... same-site top-level navigations (CSRF)

.... same-site requests (CSRF)

.... cookies where „Secure“ is set

QUIZ

- _____
- _____
- _____
- _____
- _____
- _____

Quiz time

So, which header will be valid?

- **Strict-Transport-Security:**

`max-age=1234, max-age=256000; includeSubDomains`

So, which header will be valid?

- **Strict-Transport-Security:**

`max-age=1234, max-age=256000; includeSubDomains`

So, which header will be valid?

- **Strict-Transport-Security:**

`max-age=1234, max-age=256000; includeSubDomains`

- **Set-Cookie:**

`sid=r4nd0m1D; Secure`

`sid=r4nd0m1D`

So, which header will be valid?

- **Strict-Transport-Security:**

~~max-age=1234, max-age=256000; includeSubDomains~~

- **Set-Cookie:**

~~sid=r4nd0m1D; Secure~~

sid=r4nd0m1D

So, which header will be valid?

- **Strict-Transport-Security:**

~~max-age=1234, max-age=256000, includeSubDomains~~

- **Set-Cookie:**

~~sid=r4nd0m1D; Secure~~
sid=r4nd0m1D

- **X-Frame-Options:**

SAMEORIGIN, DENY

So, which header will be valid?

- **Strict-Transport-Security:**

~~max-age=1234, max-age=256000, includeSubDomains~~

- **Set-Cookie:**

~~sid=r4nd0m1D; Secure~~

sid=r4nd0m1D

- **X-Frame-Options:**

~~SAMEORIGIN, DENY~~

So, which header will be valid?

- **Strict-Transport-Security:**

~~max-age=1234, max-age=256000, includeSubDomains~~

- **Set-Cookie:**

~~sid=r4nd0m1D; Secure~~
sid=r4nd0m1D

- **X-Frame-Options:**

~~SAMEORIGIN, DENY~~

- **Content-Security-Policy:**

script-src 'unsafe-inline'; script-src 'nonce-r4nd0m'

So, which header will be valid?

- **Strict-Transport-Security:**

~~max-age=1234, max-age=256000, includeSubDomains~~

- **Set-Cookie:**

~~sid=r4nd0m1D; Secure~~
sid=r4nd0m1D

- **X-Frame-Options:**

~~SAMEORIGIN, DENY~~

- **Content-Security-Policy:**

script-src 'unsafe-inline'; ~~script-src 'nonce-r4nd0m'~~

So, which header will be valid?

- **Strict-Transport-Security:**

~~max-age=1234, max-age=256000; includeSubDomains~~

- **Set-Cookie:**

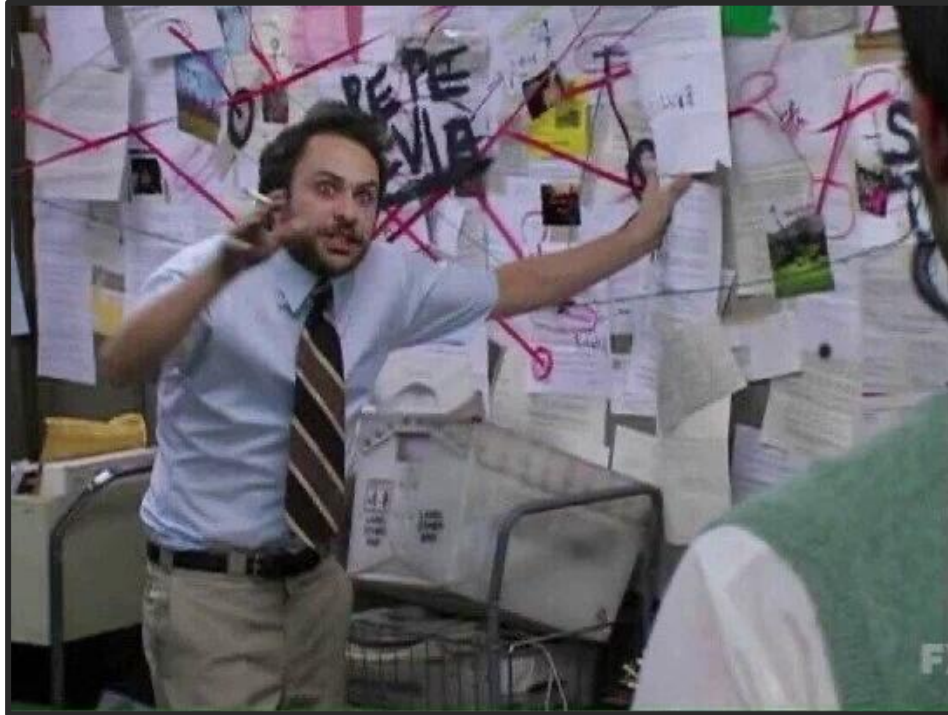
~~sid=r4nd0m1D; Secure~~
sid=r4nd0m1D

- **X-Frame-Options:**

~~SAMEORIGIN, DENY~~

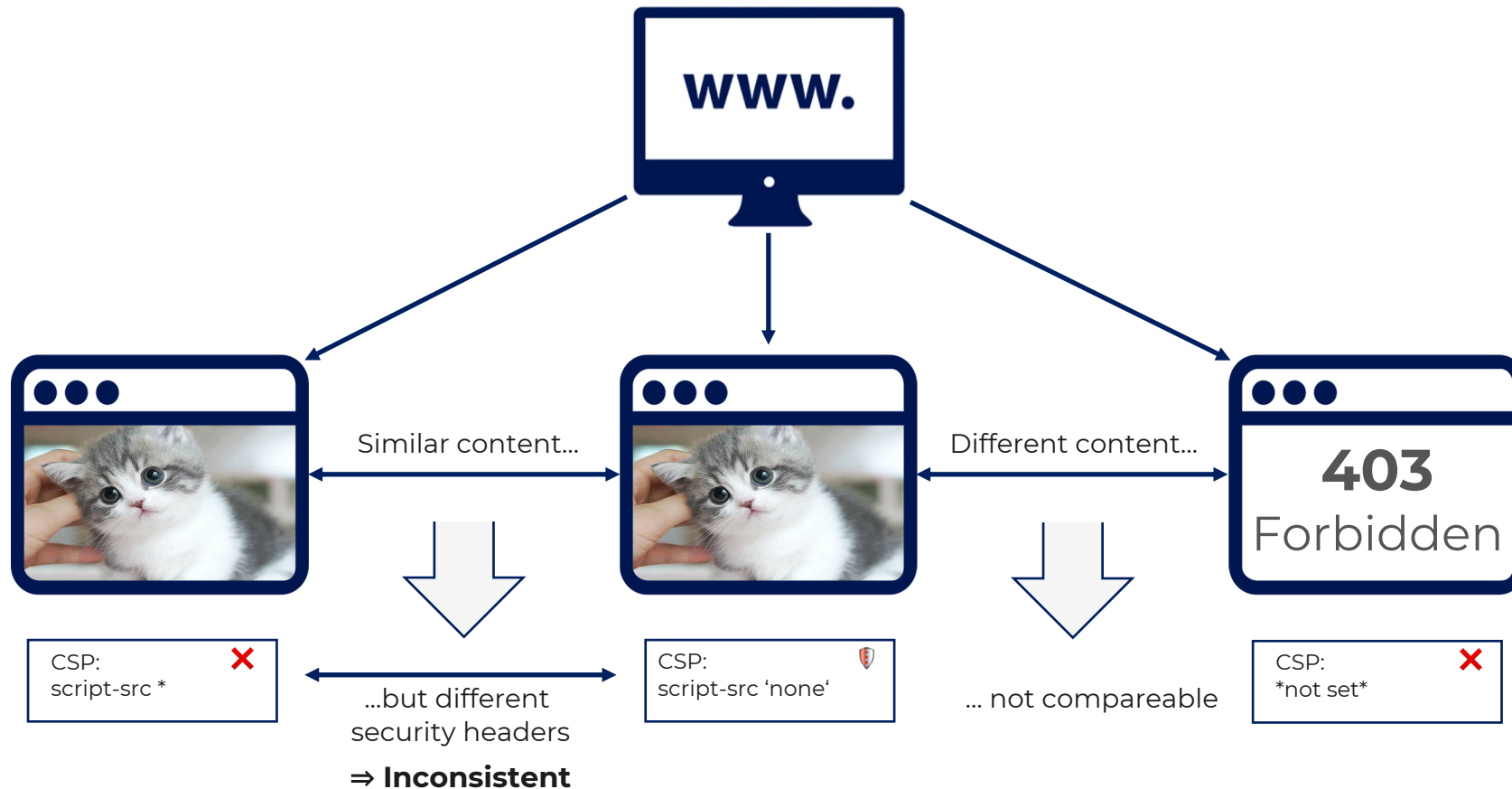
- **Content-Security-Policy:**

~~script-src 'unsafe-inline'; script-src 'nonce-r4nd0m'~~
script-src 'unsafe-inline', script-src 'nonce-r4nd0m'

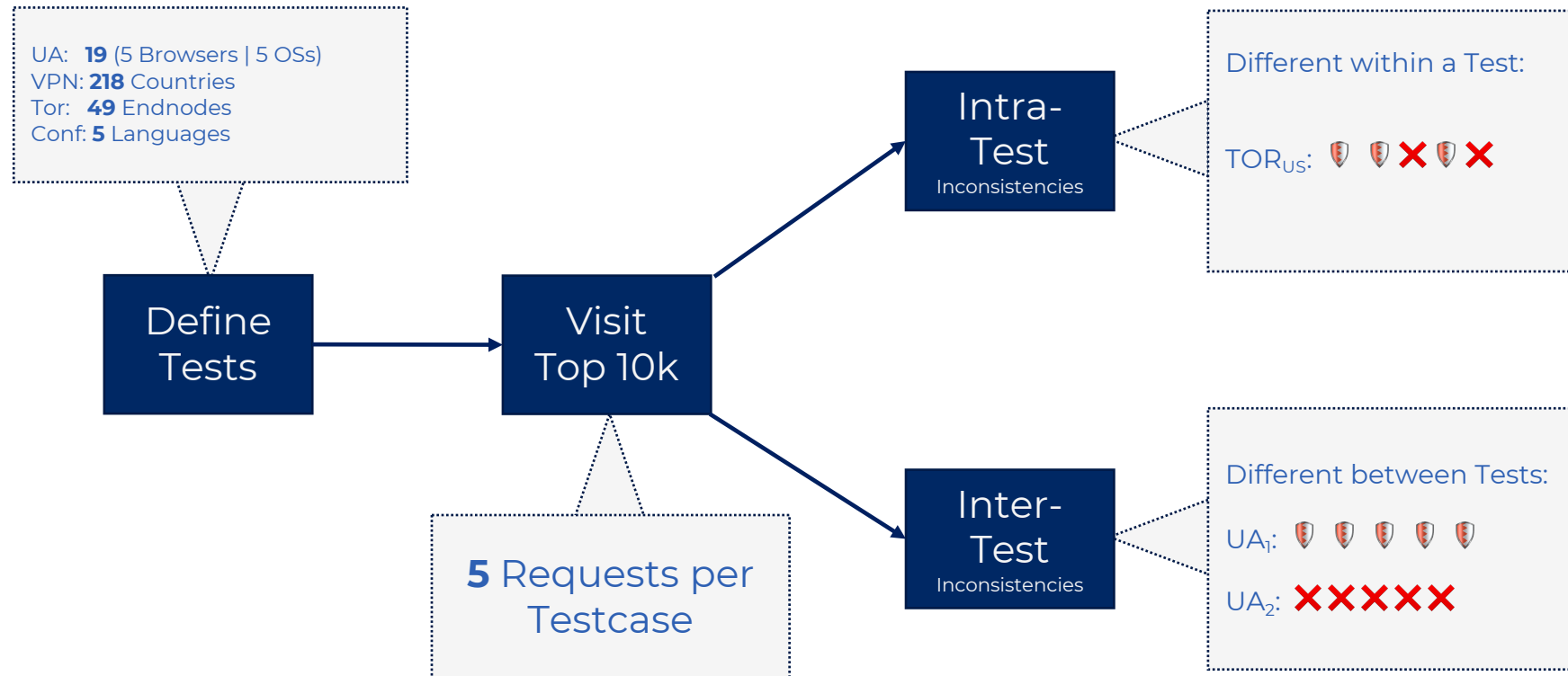


The Security Lottery

What are Security Inconsistencies?



Such science, much wow (again)



Inter-Test Inconsistencies

Mechanism	User-Agent	Language	VPN	Tor	ANY
XFO	7	-	29	13	37
HSTS	8	-	23	16	35
CSP	15	-	29	18	47
↳ XSS	9	-	1	1	10
↳ Framing	2	-	16	5	20
Cookie	150	1	13	8	167
↳ Secure	144	-	8	3	152
↳ SameSite	6	1	4	4	14
↳ HttpOnly	2	-	3	2	6
ANY	180	1	94	55	286

User-Agent parsing & traps

- Only XFO for desktop but not for mobile browsers
- Sane CSP only for non-Apple users
- Discrimination of specific browsers (Firefox, Opera, Safari)

⇒ Behaviour doesn't make sense (more later)

- No secure cookies for Firefox on iOS

⇒ User-Agent parsing issues:

- Firefox on iOS had a different version number!

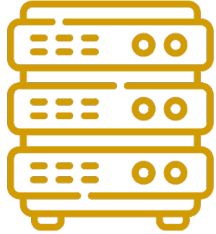
Browser traps make NO sense!

User-Agent traps to deploy certain mechanism only to supporting browser does not make any sense:

1. Users might change their UA due to privacy concerns
2. Developer (unnecesarrily) need to maintain them
3. Headers are backwards compatible!

e.g., if there is a CSP with an unknown source expression, this expression is just ignored.

Misconfigured origin servers



Intra-Test Inconsistencies

Mechanism	Intra-Test Incon.
CSP	36
XFO	50
Cookies	16
HSTS	38
-> Preload	10

⇒ Attackers can
opportunistically attack
a victim until the attack
succeeds.

Which HSTS inconsistency is worse?

1. max-age=31536000; preload
2. max-age=31536000; preload
3. <no header>
4. max-age=31536000; preload
5. <no header>

1. max-age=31536000; preload
2. max-age=31536000; preload
3. max-age=31536000
4. max-age=31536000; preload
5. max-age=31536000

Enter a domain to remove from the HSTS preload list:

Information

This form can be used to remove domains from the [HSTS preload list](#).

Removal Requirements

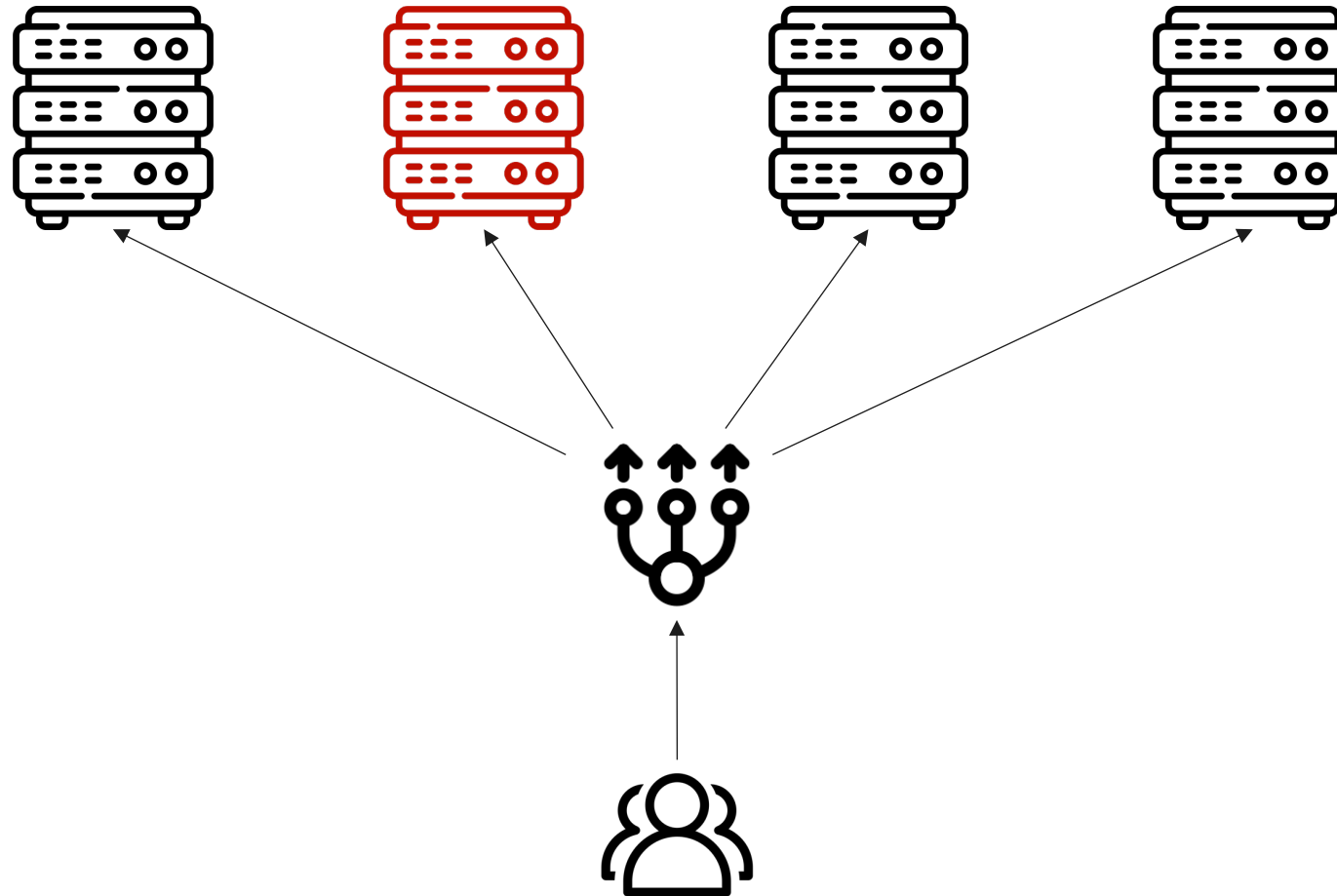
If a preloaded site sends a valid HSTS header **without** the preLoad directive, it is considered to be requesting removal from the preload list.

In order to be removed from the HSTS preload list through this form, your site must demonstrate the removal request by satisfying the following set of requirements:

1. Be **preloaded** or **pending preload** through [hstspreload.org](#).
2. Serve HTTPS with a **valid certificate**.
3. Send a **valid HSTS header**.
 - The header must **not contain** the preload directive.

<https://hstspreload.org/removal/>

Load balancing with multiple origin servers



Caching practices



Age: 9. May 2023

...

XFO: DENY



Age: 28. April 2023

...

XFO: ALLOW-FROM example.com

Take-Away Messages

Client-side security is not equally delivered to all clients!

➤ **321 sites** had some security inconsistencies!

Misconfigured servers for specific countries and browser traps enable deterministic attacks (inter-test inconsistencies).

Non-deterministic inconsistencies play into the hands of opportunistic attackers and impact Web measurements.



**When
“randomness”
bites you**

lip6.fr

Content-Security-Policy:

script-src

'nonce-R4nd0m-mail'

'nonce-R4nd0m-pub'

'nonce-R4nd0m-twitter'

'nonce-R4nd0m-2x'

'nonce-R4nd0m-showhidediv'

[...]

ahasso.heart.org (**A**merican **H**earth **A**ssociation)

Content-Security-Policy:

`script-src`

```
'nonce-base64("ahassoapplicationinsightsnonce")'  
'nonce-base64("AHASSOGoogleAnalyticsNonce")'  
'nonce-base64("ahacspbootstrap")'  
'nonce-base64("ahassophonevalidationnonce")'  
'nonce-base64("ahatokenverificationmodal")'  
'nonce-base64("ahassocustomfootercontact")'  
'nonce-base64("ahaxregexnonceval")'  
'nonce-base64("ahassorecaptchaverification!")'  
'nonce-base64("nonceforretrieveaccountaha")'  
[...]
```

parcoursup.fr

Content-Security-Policy:

```
script-src  
  'nonce-MTY4MzgWMzI0MA=='  
  [...]
```

`base64decode("MTY4MzgWMzI0MA==") <=> 1683803240`

`1683803240 <=> Thursday, 11. May 2023 13:07:20 GMT+2`

nuxt.js & caches

Feature request: static nonce for script and style tags to avoid 'unsafe inline' CSP header field [aws Amplify use case] #8646

New issue

Open

rvaneijk opened this issue on Jan 14, 2021 · 1 comment

<https://github.com/nuxt/nuxt.js/issues/8646>

For more funky case studies and how cache impacts nonce reuse:
Stay tuned for the paper!

```

    },
    "optout_session_cookie": {
      "<default>": {
        "secure": true,
        "httponly": true,
        "samesite": "lax"
      },
      "session": {
        "secure": true,
        "httponly": true,
        "samesite": "None"
      }
    }
  },
  "domaincookie-policies": {
    "domain.com": {
      "<default>": {
        "secure": true,
        "httponly": true,
        "samesite": "lax"
      },
      "CID": {
        "secure": true,
        "httponly": false,
        "samesite": "lax"
      }
    }
  }
},

```

Solving it?

Such science, much wow (again and again)

- For NDSS 2021 study, we looked at consistency both vertically and horizontally
 - Top 15k sites, up to 300 subpages from the same site
- Expectation: same object (e.g., origin or domain-scoped cookie) should have same security
- Results are sad:
 - 9% of all cookies have differing security attributes
 - Biggest fraction originates from Secure flag
 - Safe CSP (already rare) undermined by 50% of sites with at least one unsafe (or lacking) CSP
 - HSTS inconsistent on 81% of sites

Reining in the Web's Inconsistencies with Site Policy

Stefano Calzavara*, Tobias Urban[†], Dennis Tatang[‡], Marius Steffens[§], and Ben Stock[§]

*Università Ca' Foscari Venezia: calzavara@dais.unive.it

[†]Institute for Internet Security: urban@internet-sicherheit.de

[‡]Ruhr University Bochum: demis.tatang@rub.de

[§]CISPA Helmholtz Center for Information Security: {marius.steffens,stock}@cispa.saarland

Abstract—Over the years, browsers have adopted an ever-increasing number of client-enforced security policies deployed through HTTP headers. Such mechanisms are fundamental for web application security, and usually deployed on a per-page basis. This, however, enables inconsistencies, as different pages within the same security boundaries (in form of origins or sites) can express conflicting security requirements. In this paper, we formalize inconsistencies for cookie security attributes, CSP, and HSTS, and then quantify the magnitude and impact of inconsistencies at scale by crawling 15,000 popular sites. We show that numerous sites endanger their own security by omission or misconfiguration of the aforementioned mechanisms, which lead to unnecessary exposure to XSS, cookie theft, and HSTS deactivation. We then use our data to analyse to which extent the recent *Origin Policy* proposal can fix the problem of inconsistencies. Unfortunately, we conclude that the current *Origin Policy* design suffers from major shortcomings which limit its practical applicability to address security inconsistencies while catering to the need of real-world sites. Based on these insights, we propose *Site Policy*, designed to overcome *Origin Policy*'s shortcomings and make any insecurity explicit. We make a prototype implementation of *Site Policy* publicly available, along with a supporting toolchain for initial policy generation, security analysis, and test deployment.

1. INTRODUCTION

Web applications are the key access point to a plethora of online services which we use daily. However, they are also notoriously hard to secure, given the increasing amount and complexity of involved technologies [34]. Browsers implement an ever-increasing amount of server-specified, yet *client-enforced*, security policies to support secure Web application development. These policies are typically deployed through HTTP headers. Prominent examples of such security policies include cookie security attributes [21], *Content Security Policy* (CSP) [40], and *HTTP Strict Transport Security* (HSTS) [14]. Modern Web applications cannot be deemed secure unless such mechanisms are set up and correctly configured.

A key problem of existing client-side security policies is that they build on top of an extremely *fine-grained* enforcement model. Header-based security policies like CSP work at the granularity of individual HTTP responses, i.e., different pages within the security boundary of the same origin can deploy different security policies. While such expressiveness is sometimes useful in practice — since site operators might want

to fine-tune security policies on different pages for generic reasons — this threatens their sites' security by allowing for inconsistencies.

To assess the dangers of such fine-grained configuration, our paper starts from an intuitive definition of *inconsistent policy*, a general notion formalizing the dangers coming from the different adoption of the same security mechanism on different pages. Based on our definition, we analyze real-world security policies collected by crawling 15,000 popular sites and quantify inconsistencies at scale. Our analysis highlights several dangerous or potentially insecure practices, providing the first experimental evidence of the need for site-wide security policies in the wild. In particular, we show that inconsistencies might harm the expected guarantees of cookies activating specific security attributes, introduce CSP loopholes enabling script injection on apparently secure sites, and entirely disable protection on HSTS-enabled sites.

Naturally, inconsistencies leading to security issues can be rectified by deploying an *origin-wide* (or even *site-wide*) policy on all pages. The *Origin Policy* (OP) mechanism has been recently proposed specifically for this task, towards saving bandwidth in header communication and mitigating the risk of deploying incorrect security policies on some pages, e.g., error pages [10]. However, OP is not yet implemented in commercial browsers and only received limited attention by the security community so far [36]. Based on the insights of our real-world measurement, we show that the identified inconsistencies can be mitigated by OP only to a very limited extent: the “single policy per origin” model advocated by OP does not match the expectations of real sites, which sometimes deploy multiple policies on the same origin. For example, we observe that 10% of the origins that we crawled deploy more than one CSP. Even worse, we show that the origin boundary of OP is insufficient to fix inconsistencies, e.g., 81% of the sites deploy HSTS inconsistently, yet cannot take advantage of OP to rectify this issue. Finally, we identify thousands of cases where inconsistencies are introduced by the omission of security headers. While we can only make educated guesses on whether such omissions are intended, their amount and security impact is concerning enough to question the header-based, opt-in security model of OP.

Based on our analysis, we propose *Site Policy* (SP), which is designed to implement robust countermeasures to the issues we identified and overcome the limitations of *Origin Policy*. SP provides support for multiple policies within the same origin and also allows for fixing inconsistencies across the whole site, while proposing an opt-out model for security exceptions, thus making any insecurity *explicit*. At the same time, SP centralizes

Network and Distributed Systems Security (NDSS) Symposium 2021
21-24 February 2021, San Diego, CA, USA
ISBN 1 891962 46 7
<https://doi.org/10.14722/ndss.2021.23091>
www.ndss-symposium.org

For CSP and HSTS, majority of cases are omitted headers

Origin Policy to the rescue?

- **(Now deprecated) W3C proposal**

domenic Acknowledge that this is no longer being worked on

2683f92 on Apr 28, 2022

- Idea: single manifest file to specify origin-wide policies
 - HTTP header to specify Origin-Policy should be used
- **Several drawbacks**
 - Does not have “selector”, only cache identifier
 - Only one policy per origin possible
 - Omitting Origin-Policy header means: no security
 - Cookie and HSTS go beyond origin boundary

Our Proposal: Site Policy

- Take good parts from Origin-Policy
 - Central manifest file
- Fix problematic parts of Origin-Policy
 - Site-wide defaults (if header is omitted, **fall back** on that)
 - Security **exceptions** must be made **explicit** (by specifying and selecting insecure policy)
 - Parsing manifest implies understanding **worst-case security guarantees**
- ... we need to talk to W3C at some point ;-)

```
{
  "max-age": 3600,
  "csp-policies": {
    "empty": "",
    "secure_csp": "script-src 'self'"
  },
  "hsts-policies": {
    "empty": "",
    "hsts1": {
      "max-age": 63072000,
      "includeSubDomains": false
    },
    "secure_hsts": {
      "max-age": 31536000,
      "includeSubDomains": true
    }
  },
  ...
},
{
  "default_policies": {
    "domain.com": "policy_default",
    "www.domain.com": "policy1",
    "optout.domain.com": "policy_optout"
  }
}
```

QUIZ

- _____
- _____
- _____
- _____
- _____
- _____

Quiz bonus round!

So, which header will be valid? (bonus round!)

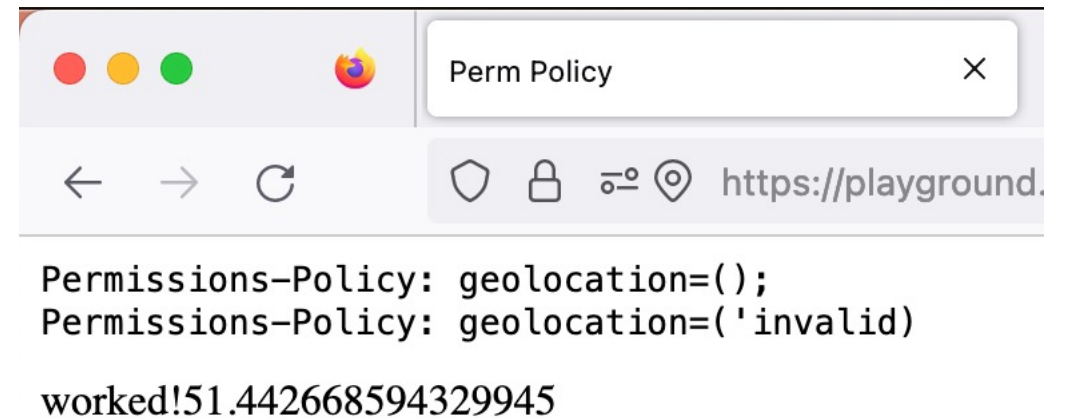
- **Permissions-Policy:**

- Permissions-Policy: geolocation ()
- Permissions-Policy: geolocation ('invalid')

So, which header will be valid? (bonus round!)

- **Permissions-Policy:**

- Permissions-Policy: ~~geolocation()~~
- Permissions-Policy: ~~geolocation('invalid')~~



So, which header will be valid? (bonus round!)

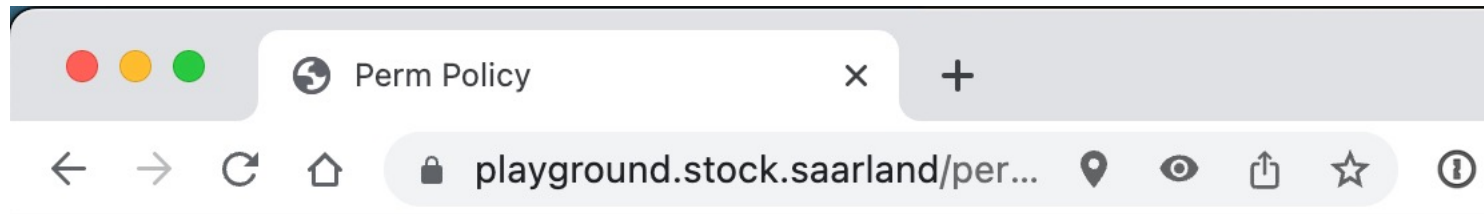
- **Permissions-Policy:**

- Permissions-Policy: geolocation ()
- Permissions-Policy: geolocation ("https://origin.com")

So, which header will be valid? (bonus round!)

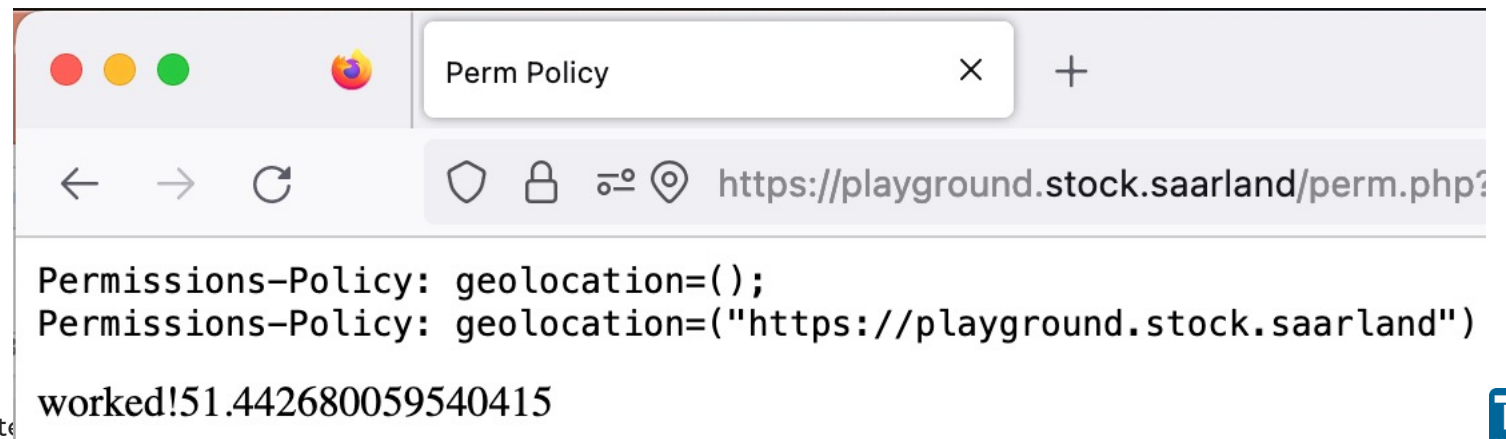
- **Permissions-Policy:**

- Permissions-Policy: ~~geolocation~~ (→)
- Permissions-Policy: geolocation ("https://origin.com")



```
Permissions-Policy: geolocation=();  
Permissions-Policy: geolocation=(\"https://playground.stock.saarland\")
```

worked!51.4428277



So, which header will be valid? (bonus round!)

- **Permissions-Policy:**

- Permissions-Policy: geolocation ()
- Permissions-Policy: geolocation ("https://*.origin.com")

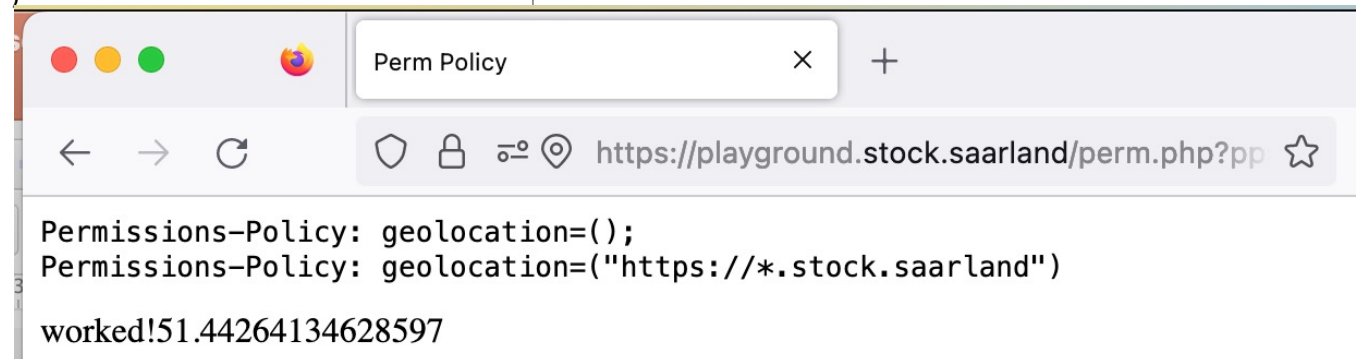
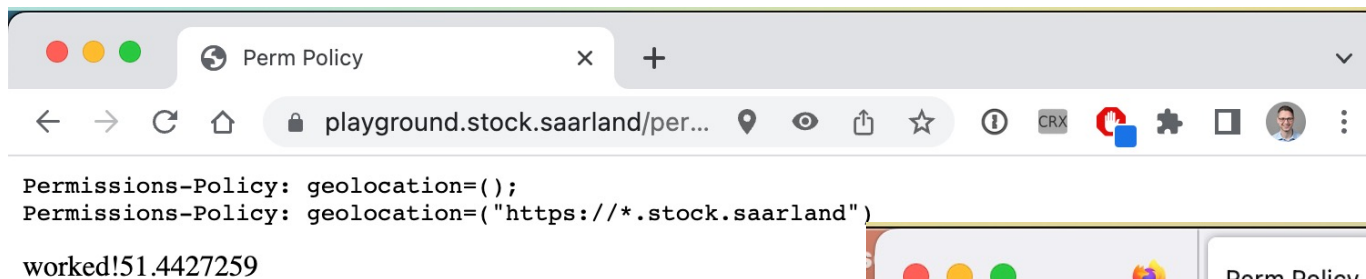
serialized-origin is the [serialization of an origin](#). However, the code points U+0027 ('), U+0021 (*), U+002C (,) and U+003B (;) MUST NOT appear in the serialization. If they are required, they must be percent-encoded as "%27", "%2A", "%2C" or "%3B", respectively.

So, which header will be valid? (bonus round!)

- **Permissions-Policy:**

- Permissions-Policy: ~~geolocation~~ (→)
- Permissions-Policy: geolocation ("https://*.origin.com")

serialized-origin is the [serialization of an origin](#). However, the code points U+0027 ('), U+0021 (*), U+002C (,) and U+003B (;) MUST NOT appear in the serialization. If they are required, they must be percent-encoded as "%27", "%2A", "%2C" or "%3B", respectively.



So, which header will be valid? (bonus round!)

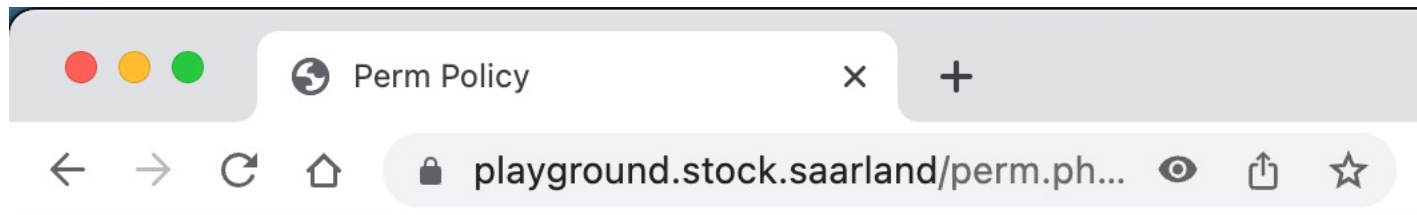
- **Permissions-Policy:**

- Permissions-Policy: geolocation ()
- Permissions-Policy: geolocation (https://origin.com)

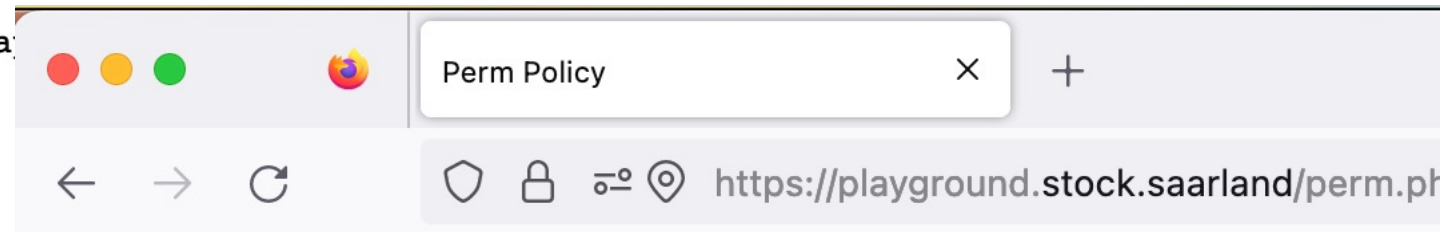
So, which header will be valid? (bonus round!)

- **Permissions-Policy:**

- Permissions-Policy: ~~geolocation=()~~
- Permissions-Policy: geolocation (https://origin.com)



Permissions-Policy: geolocation=();
Permissions-Policy: geolocation=(https://pla
disallowed

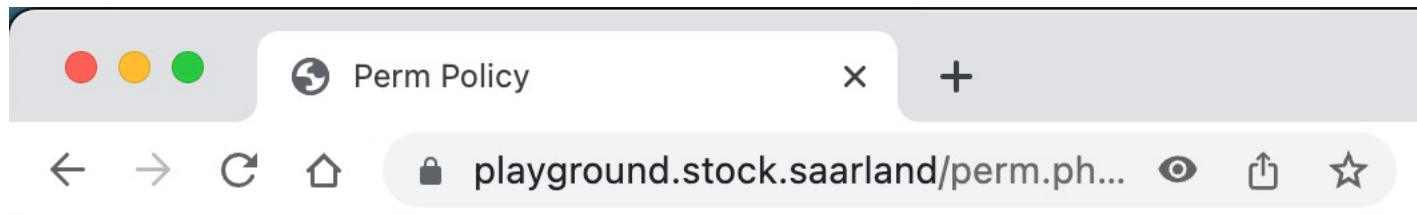


Permissions-Policy: geolocation=();
Permissions-Policy: geolocation=(https://playground.stock.saarland)
worked!51.44267755735329

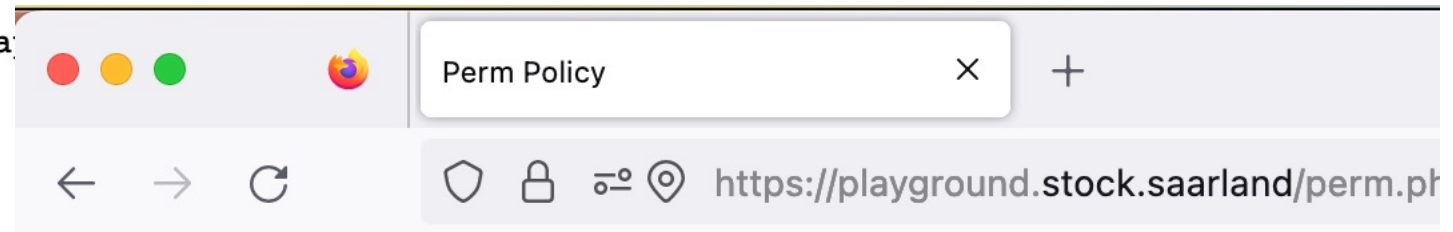
So, which header will be valid? (bonus round!)

- **Permissions-Policy:**

- Permissions-Policy: ~~geolocation=()~~
- Permissions-Policy: geolocation (~~https://origin.com~~)



```
Permissions-Policy: geolocation=();  
Permissions-Policy: geolocation=(https://pla  
disallowed
```




```
Permissions-Policy: geolocation=();  
Permissions-Policy: geolocation=(https://playground.stock.saarland)  
worked!51.44267755735329
```



**Until Site
Policy, some
best practices**

Summary & Best Practices

- Misconfigurations are **common**
 - Some due to misunderstanding of headers
 - Some due to the CDN/origin servers setup
- Scan your sites in-depth for blind spots
- Avoid **unnecessary browser switches**
 - All relevant mechanisms are backwards compatible
- Beware of duplicate header rules
 - CSP: Composition, HSTS: **first**, Permissions-Policy: last, Set-Cookie: last, Referrer-Policy: last, XFO: wtf

 [cispa/the-security-lottery](https://github.com/cispa/the-security-lottery)

Let's discuss!