On the Content Security Policy Violations due to the Same-Origin Policy

Dolière Francis Some
Université Côte d'Azur
Inria
France
doliere.some@inria.fr

Nataliiia Bielova
Université Côte d'Azur
Inria
France
nataliia.bielova@inria.fr

Tamara Rezk
Université Côte d'Azur
Inria
France
tamara.rezk@inria.fr

ABSTRACT

Modern browsers implement different security policies such as the Content Security Policy (CSP), a mechanism designed to mitigate popular web vulnerabilities, and the Same Origin Policy (SOP), a mechanism that governs interactions between resources of web pages.

In this work, we describe how CSP may be violated due to the SOP when a page contains an embedded iframe from the same origin. We analyse 1 million pages from 10,000 top Alexa sites and report that at least 31.1% of current CSP-enabled pages are potentially vulnerable to CSP violations. Further considering real-world situations where those pages are involved in same-origin nested browsing contexts, we found that in at least 23.5% of the cases, CSP violations are possible.

During our study, we also identified a divergence among browsers implementations in the enforcement of CSP in srcdoc sandboxed iframes, which actually reveals a problem in Gecko-based browsers CSP implementation. To ameliorate the problematic conflicts of the security mechanisms, we discuss measures to avoid CSP violations.

CCS CONCEPTS

•Security and privacy →Web application security;

ACM Reference format:
Dolière Francis Some, Nataliiia Bielova, and Tamara Rezk. 2016. On the Content Security Policy Violations due to the Same-Origin Policy. In *Proceedings of WWW '17, Perth, Western Australia, April 3–7, 2017,* 9 pages.
DOI: 10.1145/1235

1 INTRODUCTION

Modern browsers implement different specifications to securely fetch and integrate content. One widely used specification to protect content is the Same Origin Policy (SOP) [?]. SOP allows developers to isolate untrusted content from a different origin. An origin here is defined as scheme, host, and port number. If an iframe's content is loaded from a different origin, SOP controls the access to the embedder resources. In particular, no script inside the iframe can access content of the embedder page. However, if the iframe's content is loaded from the same origin as the embedder page, there are no privilege restrictions w.r.t. the embedder resources. In such a case, a script executing inside the iframe can access content of the embedder webpage. Scripts are considered trusted and the *iframe becomes transparent* from a developer view point. A more recent specification to protect content in webpages is the Content Security Policy (CSP) [?]. The primary goal of CSP is to mitigate cross site scripting attacks (XSS), data leaks attacks, and other types of attacks. CSP allows developers to specify, among other features, trusted domain sources from which to fetch content. One of the most important features of CSP, is to allow a web application developer to specify trusted JavaScript sources. This kind of restriction is meant to permit execution of only trusted code and thus prevent untrusted code to access content of the page.

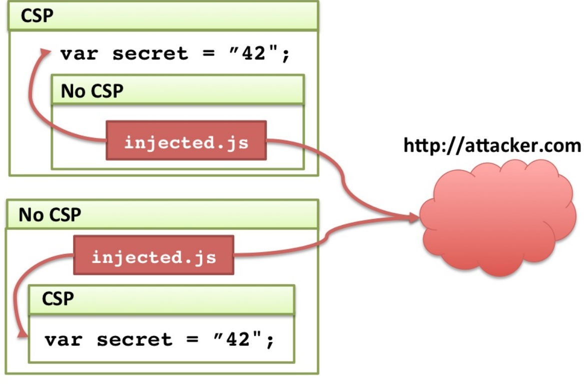

Figure 1: An XSS attack despite CSP.

In this work, we report on a fundamental problem of CSP. CSP[?] defines how to protect content in an isolated page. However, it does not take into consideration the page's context, that is its embedder or embedded iframes. In particular, CSP is unable to protect content of its corresponding page if the page embeds (using the *src* attribute) an iframe of the same origin. The CSP policy of a page will not be applied to an embedded iframe. However, due to SOP, the iframe has complete access to the content of its embedder. Because same origin iframes are transparent due to SOP, this opens loopholes to attackers whenever the CSP policy of an iframe and that of its embedder page are not compatible (see Fig. 1).

We analysed 1 million pages from the top 10,000 Alexa sites and found that 5.29% of sites contain some pages with CSPs (as opposed to 2% of home pages in previous studies [?]). We have identified that in 94% of cases, CSP may be violated in presence

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
WWW '17, Perth, Western Australia
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
978-1-4503-2138-9…$15.00
DOI: 10.1145/1235

of the document.domain API and in 23.5% of cases CSP may be violated without any assumptions (see Table 3).

During our study, we also identified a divergence among browsers implementations in the enforcement of CSP [?] in sandboxed iframes embedded with *srcdoc*, which actually reveals an inconsistency between the CSP and HTML5 sandbox attribute specification for iframes.

We identify and discuss possible solutions from the developer point of view as well as new security specifications that can help prevent this kind of CSP violations. We have made publicly available the dataset that we used for our results in[?]. We have installed an automatic crawler to recover the same dataset every month to repeat the experiment taking into account the time variable. An accompanying technical report with a complete account of our analyses can be found at [?].

In summary, our contributions are: (i) We describe a new class of vulnerabilities that lead to CSP violations. (Section 2). (ii) We perform a large and depth scale crawl of top sites, highlighting CSP adoption at sites-level, as well as sites origins levels. Using this dataset, we report on the possibilities of CSP violations between the SOP and CSP in the wild. (Section 3). (iii) We propose guidelines in the design and deployment of CSP. (Section 4). (iv) We reveal an inconsistency between the CSP specification and HTML5 sandbox attribute specification for iframes. Different browsers choose to follow different specifications, and we explain how any of these choices can lead to new vulnerabilities. (Section 5).

2 CONTENT SECURITY POLICY AND SOP

The Content Security Policy (CSP) [?] is a mechanism that allows programmers to control which client-side resources can be loaded and executed by the browser. CSP (version 2) is an official W3C candidate recommendation [?], and is currently supported by major web browsers. CSP is delivered in the `Content-Security-Policy` HTTP response header, or in a `<meta>` element of HTML.

CSP applicability A CSP delivered with a page controls the resources of the page. However it does not apply to the page's embedding resources [?]. As such, CSP does not control the content of the iframes even if the iframe is from the same origin as the main page according to SOP. Instead, the content of the iframe is controlled by the CSP delivered with it, that can be different from the CSP of the main page.

CSP directives CSP allows a programmer to specify which resources are allowed to be loaded and executed in the page. These resources are defined as a set of origins and known as a *source list*. Additionally to controlling resources, CSP allows to specify allowed destinations of the AJAX requests by the `connect-src` directive. A special header `Content-Security-Policy-Report-Only` configures a CSP in a report-only mode: violations are recorded, but not enforced. The directive `default-src` is a special fallback directive that is used when some directive is not defined. The directive `frame-ancestors` (meant to supplant the HTTP `X-Frame-Options` header[?]), controls in which pages the current page may be included as an iframe, to prevent clickjacking attacks [?]. See Table 1 for the most commonly used CSP directives [?].

Source lists CSP source list is traditionally defined as a *whitelist* indicating which domains are trusted to load the content, or to communicate. For example, a CSP from Listing 1 allows to include scripts only from `third.com`, requires to load frames only over HTTPS, while other resource types can only be loaded from the same hosting domain.

Directive	Controlled content
`script-src`	Scripts
`default-src`	All resources (fallback)
`style-src`	Stylesheets
`img-src`	Images
`font-src`	Fonts
`connect-src`	XMLHttpRequest, WebSocket or EventSource
`object-src`	Plug-in formats (object, embed)
`report-uri`	URL where to report CSP violations
`media-src`	Media (audio, video)
`child-src`	Documents (frames), [Shared] Workers
`frame-ancestors`	Embedding context

Table 1: Most common CSP directives [?].

```
1 Content-Security-Policy: default-src 'self';
2 script-src third.com; child-src https:
```

Listing 1: Example of a CSP policy.

A whitelist can be composed of concrete hostnames (`third.com`), may include a wildcard `*` to extend the policy to subdomains (`*.third.com`), a special keyword `'self'` for the same hosting domain, or `'none'` to prohibit any resource loading.

Restrictions on scripts Directive `script-src` is the most used feature of CSP in today's web applications [?]. It allows a programmer to control the origin of scripts in his application using source lists. When the `script-src` directive is present in CSP, it blocks an execution of any inline script, JavaScript event handlers and APIs that execute string data code, such as `eval()` and other related APIs. To relax the CSP, by allowing the execution of inline `<script>` and JavaScript event handlers, a `script-src` whitelist should contain a keyword `'unsafe-inline'`. To allow `eval()`-like APIs, the CSP should contain a `'unsafe-eval'` keyword. Because `'unsafe-inline'` allows execution of *any* inlined script, it effectively removes any protection against XSS. Therefore, nonces and hashes were introduced in CSP version 2 [?], allowing to control which inline scripts can be loaded and executed.

Sandboxing iframes Directive `sandbox` allows to load resources but execute them in a separate environment. It applies to all the iframes and other content present on the page. An empty `sandbox` value creates completely isolated iframes. One can selectively enable specific features via `allow-*` flags in the directive's value. For example, `allow-scripts` will allow executions of scripts in an iframe, and `allow-same-origin` will allow iframes to be treated as being from their normal origins.

Same-Site and Same-Origin Definitions. In our terminology, we distinguish the web pages that belong to the same site from the pages that belong to the same origin. By *page* we refer to any HTML document – for example, the content of an iframe we call *iframe page*. In this case, the page that embeds an iframe is called a *parent page* or *embedder*.

By *site* we refer to the highest level domain that we extract from Alexa top 10,000 sites, usually containing the domain name and a TLD, for example `main.com`. All the pages that belong to a site, and to any of its subdomains as `sub.main.com`, are considered *same-site* pages.

According to the Same Origin Policy, an *origin* of a page is scheme, host and port of its URL. For example, in http://main.com:81/dir/p.html, the scheme is "http", the host is "main.com" and the port is 81.

2.1 CSP violations due to SOP

Consider a web application, where the main page `A.html` and its iframe `B.html` are located at `http://main.com`, and therefore belong to the same origin according to the same-origin policy. `A.html`, shown in Listing 2, contains a script and an iframe from `main.com`. The local script `secret.js` contains sensitive information given in Listing 3. To protect against XSS, the developer have installed the CSP for its main page `A.html`, shown in Listing 4.

```
1   <html>
2     <script src="secret.js"></script>
3      ...
4     <iframe src="B.html"></iframe>
5   </html>
```

Listing 2: Source code of `http://main.com/A.html`.

```
1     var secret = "42";
```

Listing 3: Source code of `secret.js`.

```
1 Content-Security-Policy: default-src 'none';
2 script-src 'self'; child-src 'self'
```

Listing 4: CSP of `http://main.com/A.html`.

This CSP provides an effective protection against XSS:

2.1.1 Only parent page has CSP. According to the latest version of CSP[1], only the CSP of the iframe applies to its content, and it ignores completely the CSP of the including page. In our case, if there is no CSP in `B.html` then its resource loading is not restricted. As a result, an iframe `B.html` without CSP is potentially vulnerable to XSS, since any injected code may be executed within `B.html` with no restrictions. Assume `B.html` was exploited by an attacker injecting a script `injected.js`. Besides taking control over `B.html`, this attack now propagates to the including page `A.html`, as we show in Fig. 1. The XSS attack extends to the including parent page because of the inconsistency between the CSP and SOP. When a parent page and an iframe are from the same origin according to SOP, a parent and an iframe share the same privileges and can access each other's code and resources.

For our example, `injected.js` is shown in Listing 5.

This script executed in `B.html` retrieves the secret value from its parent page (`parent.secret`) and transmits it to an attacker's server `http://attacker.com` via XMLHttpRequest[2].

```
1 function sendData(obj, url){
2   var req = new XMLHttpRequest();
3   req.open('POST', url, true);
4   req.send(JSON.stringify(obj));
5 }
6 sendData({secret: parent.secret}, 'http://
      attacker.com/send.php');
```

Listing 5: Source code of `injected.js`.

A straightforward solution to this problem is to ensure that the protection mechanism for the parent page also propagates to the iframes from the same domain. Technically, it means that the CSP of the iframe should be the same or more restrictive than the CSP of the parent. In the next example we show that this requirement does not necessarily prevent possible CSP violations due to SOP.

2.1.2 Only iframe page has CSP. Consider a different web application, where the including parent page `A.html` does not have a CSP, while its iframe `B.html` contains a CSP from Listing 4. In this example, `B.html`, shown in Listing 6 now contains some sensitive information stored in `secret.js` (see Listing 3).

```
1   <html>
2      ...
3     <script src="secret.js"></script>
4   </html>
```

Listing 6: Source code of `http://main.com/B.html`.

Since the including page `A.html` now has no CSP, it is potentially vulnerable to XSS, and therefore may have a malicious script `injected.js`. The iframe `B.html` has a restrictive CSP, that effectively contributes to protection against XSS. Since `A.html` and `B.html` are from the same origin, the malicious injected script can profit from this and steal sensitive information from `B.html`. For example, the script may call the `sendData` function with the secret information:

```
1 sendData({secret: children[0].secret}, 'http:
      //attacker.com/send.php');
```

Thanks to SOP, the script `injected.js` fetches the secret from it's child iframe `B.html` and sends it to `http://attacker.com`.

2.1.3 CSP violations due to origin relaxation. A page may change its own origin with some limitations. By using the `document.domain` API, the script can change its current domain to a superdomain. As a result, a shorter domain is used for the subsequent origin checks[3].

Consider a slightly modified scenario, where the main page `A.html` from `http://main.com` includes an iframe `B.html` from its sub-domain `http://sub.main.com`. Any script in `B.html` is able to change the origin to `http://main.com` by executing the following line:

```
1     document.domain = "main.com";
```

If `A.com` is willing to communicate with this iframe, it should also execute the above-written code so that the communication with `B.html` will be possible. The content of `B.html` is now treated by the web browser as the same-origin content with `A.html`, and therefore any of the previously described attacks become possible.

[1] https://www.w3.org/TR/CSP2/#which-policy-applies

[2] The XMLHttpRequest is not forbidden by the SOP for `B.html` because an attacker has activated the Cross-Origin Resource Sharing mechanism [?] on her server `http://attacker.com`.

[3] https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy#Changing_origin

2.1.4 Categories of CSP violations due to SOP. We distinguish three different cases when the CSP violation might occur because of SOP:

Only parent page or only iframe has CSP A parent page and an iframe page are from the same origin, but only one of them contains a CSP. The CSP may be violated due to the unrestricted access of a page without CSP to the content of the page with CSP. We demonstrated this example in Sections 2.1.1 and 2.1.2.

Parent and iframe have different CSPs A parent page and an iframe page are from the same origin, but they have different CSPs. Due to SOP, the scripts from one page can interfere with the content of another page thus violating the CSP.

CSP violation due to origin relaxation A parent page and an iframe page have the same higher level domain, port and scheme, but however they are not from the same origin. Either CSP is absent in one of them, or they have different CSPs – in both cases CSP may be violated because the pages can relax their origin to the high level domain by using `document.domain` API, as we have shown in Section 2.1.3.

3 EMPIRICAL STUDY OF CSP VIOLATIONS

We have performed a large-scale study on the top 10,000 Alexa sites to detect whether CSP may be violated due to an inconsistency between CSP and SOP. For collecting the data, we have used CasperJS [?] on top of PhantomJS headless browser [?]. The User-Agent HTTP header was instantiated as a recent Google Chrome browser.

3.1 Methodology

The overview of our data collection and CSP comparison process is given in Figure 2. The main difference in our data collection process from previous works on CSP measurements in the wild [? ?] is that we crawl not only the main pages of each site, but also other pages. First, we collect pages accessible through links of the main page and pointing to the same site. Second, to detect possible CSP violations due to SOP, we have collected all the iframes present on the home pages and linked pages.

3.1.1 Data Collection. **Home Page Crawler** For each site in top 10,000 Alexa list, we crawl the home page, parse its source code and extract three elements: (1) a CSP of the site's home page stored in HTTP header as well as in `<meta>` HTML tag; we denote the CSPs of the home page by C; (2) to extract more pages from the same site, we analyse the source of the links via `<a href=...>` tag and extract URLs that point to the same site, we denote this list by L. (3) we collect URLs of iframes present on the home page via `<iframe src=...>` tag and record only those belonging to the same site, we denote this set by $\mathcal{F}$.

Page Crawler We crawl all the URLs from the list of pages L, and for each page we repeat the process of extraction of CSP and relevant iframes, similar to the steps (1) and (3) of the home page crawler. As a result, we get a set of CSPs of linked pages C_L and a set of iframes URLs $\mathcal{F}_L$ that we have extracted from the linked pages in L.

Iframe Crawler

For every iframe URL present in the list of home page iframes $\mathcal{F}_H$, and in the list of linked pages iframes $\mathcal{F}_L$, we extract their corresponding CSPs and store in two sets: C_F for home page iframes and C_{LF} for linked page iframes.

3.1.2 CSP adoption analysis. Since CSP is considered an effective countermeasure for a number of web attacks, programmers often use it to mitigate such attacks on the main pages of their sites. However, if CSP is not installed on some pages of the same site, this can potentially leak to CSP violations due to the inconsistency with SOP when another page from the same origin is included as an iframe (see Figure 1). In our database, for each site, we recorded its home page, a number of linked pages and iframes from the same site. This allows us to analyse how CSP is adopted at every popular site by checking the presence of CSP on every crawled page and iframe of each site. To do so, we analyse the extracted CSPs: C for the home page, C_L for linked pages, C_F for home page iframes, and C_{LF} for linked pages iframes.

3.1.3 CSP violations detection. To detect possible CSP violations due to SOP, we have analysed home pages and linked pages from the same site, as well as iframes embedded into them.

CSP Selection

To detect CSP violations, we first remove all the sites where no parent page and no iframe page contains a CSP. For the remaining sites, we pointwise compare (1) the CSPs of the home pages C and CSPs of iframes present on these pages C_F; (2) the CSPs of the linked pages C_L and CSPs of their iframes C_{LF}. To check whether a parent page CSP and an iframe CSP are equivalent, we have applied the CSP comparison algorithm (Figure 2)

CSP Preprocessing We first normalise each CSP policy, by splitting it into its directives.

- If **default-src** directive is present (**default-src** is a fallback for most of the other directives), then we extract the source list s of **default-src**. We analyse which directives are missing in the CSP, and explicitly add them with the source list s.
- If **default-src** directive is absent, we extract missing directives from the CSP. In this case, there are no restrictions in CSP for every absent directive. We therefore explicitly add them with the most permissive source list. A missing **script-src** is assigned * **'unsafe-inline' 'unsafe-eval'** as the most permissive source list [?].
- In each source list, we modify the special keywords: (i) **'self'** is replaced with the origin of the page containing the CSP; (ii) in case of **'unsafe-inline'** with hash or nonce, we remove **'unsafe-inline'** from the directive since it will be ignored by the CSP2. (iii) **'none'** keywords are removed from all the directives; (iv) nonces and hashes are removed from all the directives since they cannot be compared; (iv) each whitelisted domain is extended with a list of schemes and port numbers from the URL of the page includes the CSP[4].

[4]For example, according to CSP2, if the page scheme is `https`, and a CSP contains a source `example.com`, then the user agent should allow content only from `https://example.com`, while if the current scheme is `http`, it would allow both `http://example.com` and `https://example.com`.

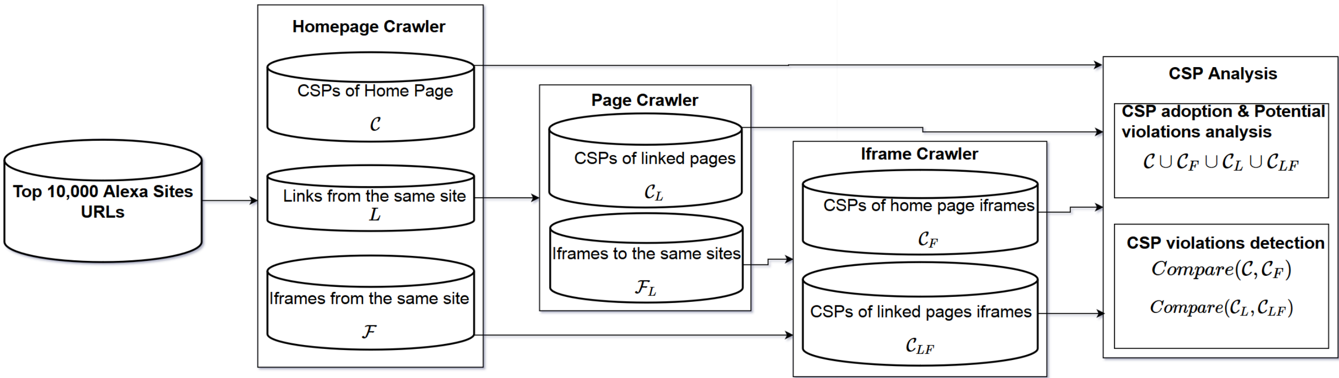

Figure 2: Data Collection and Analysis Process

Sites successfully crawled	9,885
Pages visited	1,090,226
Pages with iframe(s) from the same site	648,324
Pages with same-origin iframe(s)	92,430
Pages with same-origin iframe(s) where page and/or iframe has CSP	692
Pages with CSP	21,961 (2.00%)
Sites with CSP on home page	228 (2.3%)
Sites with CSP on some pages	523 (5.29%)

Table 2: Crawling statistics

CSP Comparison We compare all the directives present in the two CSPs to identify whether the two policies require the same restrictions. Whenever the two CSPs are different, our algorithm returns the names of directives that do not match. The demonstration of the comparison is accessible on[**?**]. For each directive in the policies we compare the source lists and the algorithm proceeds if the elements of the lists are identical in the normalised CSPs.

3.1.4 Limitations. Our methodology and results have two(2) limitations that we explain here.

User interactions The automatic crawling process did not include any real-user-like interactions with top sites. As such the set of iframes and links URLs we have analysed is an underestimate of all links and iframes a site may contain.

Pairs of (parent-iframe) In this study, we consider CSP violations in same origin (parent, iframe) couples only. Their are though further combinations such as couples of sibling iframes in a parent page that we could have considered. Overall, our results are conservative, since the problem might have been worst without those limitations.

3.2 Results on CSP Adoption

The crawling of Alexa top 10,000 sites was performed in the end of August, 2016. To extract several pages from the same site, we have also crawled all the links and iframes on a page that point to the same site. In total, we have gathered 1,090,226 from 9,885 different sites. On median, from each site we extracted 45 pages, with a maximum number of 9,055 pages found on tuberel.com. Our crawling statistics is presented in Table 2. More than half of the

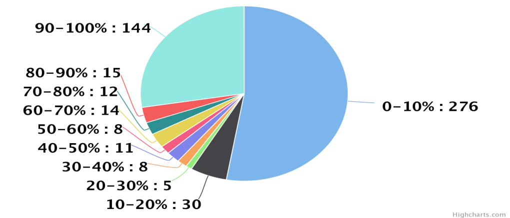

Figure 3: Percentage of pages with CSP per site

pages contain an iframe, and 13% of pages do contain an iframe from the same site. This indicates the potential surface for the CSP violations, when at least one page on the site has a CSP installed. We discuss such potential CSP violation in details in Section 3.3.3. Similarly to previous works on CSP adoption [**? ?**], we have found that CSP is present on only 228 out of 9,885 home pages (2.31%). While extending this analysis to almost a million pages, we have found a similar rate of CSP adoption (2.00%).

Differently from previous studies that anlaysed only home pages, or only pages in separation, we have analysed how many sites have at least some pages that adopted CSP. We have grouped all pages by sites, and found that 5.29% of sites contain some pages with CSPs. It means that CSP is more known by the website developers, but for some reason is not widely adopted on all the pages of the site.

We have then analysed how many pages on each site have adopted CSPs. For each of 523 sites, we have counted how many pages (including home page, linked pages and iframes) have CSPs. Figure 3 shows that more than half of the sites have a very low CSP adoption on their pages: on 276 sites out of 529, CSP is installed on only 0-10% of their pages. This becomes problematic if other pages without CSP are not XSS-free. However, it is interesting that around a quarter of sites do profit from CSP by installing it on 90-100% of their pages.

3.3 Results on CSP violations due to SOP

As described in Section 2.1.4, we distinguish several categories of CSP violations when a parent page and an iframe on this page are from the same origin according to SOP. To account for possible CSP violations, we only consider cases when either parent, or iframe, or both have a CSP installed. From all the 21,961 pages that have CSP

	Same-origin parent-iframe	Possible to relax origin	Total
Only parent page CSP	**83**	**1388**	**1471**
Only iframe CSP	**16**	**240**	**256**
Different CSP	**70**	**44**	**114**
No CSP violations	551 (76.5%)	109 (6%)	660
CSP violations total	**169 (23.5%)**	**1672 (94%)**	**1841**

Table 3: Statistics CSP violations due to Same-Origin Policy

	Same-origin parent-iframe	Possible to relax origin
Only parent page CSP	yandex.ru	twitter.com, yandex.ru, mail.ru
Only iframe CSP	amazon.com, imdb.com	–*
Different CSP	twitter.com	–*

*Not found in top 100 Alexa sites.

Table 4: Sample of sites with CSP violations due to Same-Origin Policy

installed, we have removed the pages, where CSPs are in report-only mode, having left 18,035 pages with CSPs in enforcement mode.

Table 3 presents possible CSP violations due to SOP.

We have extracted the parent-iframe couples that might cause a CSP violation because either (1) only parent or only iframe installed a CSP, or (2) both installed different CSPs. First, to account for direct violations because of SOP, we distinguish couples where parent and iframe are from the same origin (columns 2,3), we have found 720 cases of such couples. Second, we analyse possible CSP violations due to origin relaxation: we have collected 1781 couples that are from different origins but their origins can be relaxed by `document.domain` API (see more in Section 2.1.3) – these results are shown in columns 4 and 5.

In Table 4 we present the names of the domains out of top 100 Alexa sites, where we have found different CSP violations. Each company in this table have been notified about the possible CSP violation. Concrete examples of the page and iframe URLs and their corresponding CSPs for each such violation can be found in the corresponding technical report [**?**]. All the collected data is available online[**?**].

CSP violations in presence of `document.domain` According to our results, in presence of `document.domain`, 94% of (parent, iframe) pages can have their CSP violated. Those violations can occur only if both parent and iframes pages execute `document.domain` to the same top level domain. Thus, our result is an over-approximation, assuming that `document.domain` is used in all of those pages and iframes. According to[**?**], `document.domain` is used in less than 3% of web pages.

3.3.1 *Only parent page or only iframe has CSP.*

We first consider a scenario when a parent page and an iframe are from the same origin, but only one of them contains a CSP. Intuitively, if only a parent page has CSP, then an iframe can violate CSP by executing any code and accessing the parent page's DOM, inserting content, access cookies etc. Among 720 parent-iframe couples from *the same origin*, we have found 83 cases (11.5%) when only parent has a CSP, and 16 cases (2.2%) when only iframe has a CSP. These CSP violations originate from 13 (for parent) and 4 (for iframe) sites.

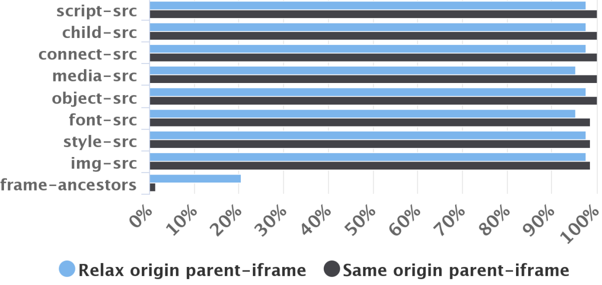

Figure 4: Differences in CSP directives for parent and iframe pages

For example, such possible violations are found on some pages of amazon.com, yandex.ru and imdb.com (see Table 4). CSP of a parent or iframe may also be violated because of *origin relaxation*. We have identified 1388 cases (78%) of parent-iframe couples where such violation may occur because CSP is present only in the parent page. This was observed on 20 different sites, including twitter.com, yandex.ru and others. Finally, in 240 cases (13.5%) only iframe has CSP installed, which was found on 11 different sites. We manually checked the parent and iframes involved in CSP violations for sites in Table 4. In all of those sites, either the parent or the iframe page is login page[**?**]. We furthermore checked how effective are the CSP of those pages, using CSPEvaluator[5], proposed by Lukas et al.[**?**]. and found out that the CSP policies involved in these are moreover all bypassable.

3.3.2 *Parent and iframe have different CSPs.*

In a case when a page and iframe are from the same origin, but their corresponding CSPs are different, may also cause a violation of CSP. From the 720 *same-origin* parent-iframe couples, we have found 70 cases (9.7%) (from 3 sites) when their CSPs differ, and for *an origin relaxation* (from 6 sites) case, we have identified only 44 such cases (2.5%). This setting was found on some pages of twitter.com for instance.

We have further analysed the differences in CSPs found on parent and iframe pages. For all the 114 pairs of parent-iframe (either

[5]https://csp-evaluator.withgoogle.com/

	Pages	Origins	Sites
A same origin page has no CSP	4381	197	197
A same origin page has a different CSP	1223	23	23
A same origin (after relaxation) page has no CSP	4728	340	183
A same origin (after relaxation) has a different CSP	2567	135	44
Potential violations total	12899 (72%)	591 (81%)	379 (52%)

Table 5: Potential CSP violations in pages with CSP

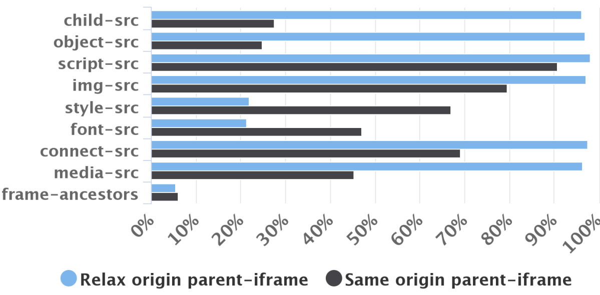

Figure 5: Differences in CSP directives for same-origin and relaxed origin pages

same-origin or possible origin relaxation), we have compared CSPs they installed, directive-by-directive. Figure 4 shows that every parent CSP and iframe CSP differ on almost every directive – between 90% and 100%. The only exception is **frame-ancestors** directive, which is almost the same in different parent pages and iframes. If properly set, this directive gives a strong protection against click-jacking attacks, therefore all the pages of the same origin are equally protected.

3.3.3 Potential CSP violations. A *potential CSP violation* may happen when in a site, either some pages have CSP and some others do not, or pages have different CSP. When those pages get nested as parent-iframe, we can run into CSP violations, just like in the direct CSP violations cases we have just reported above. To analyse how often such violations may occur, we have analysed the 18,035 pages that have CSP in enforcement mode. These pages originate from 729 different origins spread over 442 sites. Table 5 shows that 72% of CSPs (12,899 pages) are potentially violated, and these CSPs originate from pages of 379 different sites (85.75%). To detect these violations, for each page with a CSP in our database, we have analysed whether there exists another page from the same origin, that does not have CSP. This page could embed the page with CSP and violate it because of SOP. We have detected 4381 such pages (24%) from 197 origins. Similarly, we detected 1223 pages (7%) when there are same-origin pages with a different CSP. Similarly, we have analysed when potential CSP violations may happen due to origin relaxation. We have detected 4728 pages (26%), whose CSP may be violated because of other pages with no CSP, and 2567 pages (14%), whose CSP may be violated because of different CSP on other relaxed-origin pages.

For the pages that have different CSPs, we have compared how much CSPs differ. Figure 5 shows that CSPs mostly differ in `script-src` directive, which protects pages from XSS attacks. This means, that if one page in the origin does whitelist an attacker's domain or an insecure endpoints [?], all the other pages in the same origin become vulnerable because they may be inserted as an iframe to the vulnerable page and their CSPs can be easily violated.

3.4 Responses of websites owners

We have reported those issues to a sample of sites owners, using either HackerOne[6], or contact forms when available. Here are some selected quotes from our discussions with them.

> "*Yes, of course we understand the risk that under some circumstances XSS on one domain can be used to bypass CSP on another domain, but it's simply impossible to implement CSP across all (few hundreds) domains at once on the same level. We are implementing strongest CSP currently possible for different pages on different domains and keep going with this process to protect all pages, after that we will strengthen the CSP. We believe it's better to have stronger CSP policy where possible rather than have same weak CSP on all pages or not having CSP at all. Having in mind there are hundreds of domains within mail.ru, at least few years are required before all pages on all domains can have strong CSP.*"
> – `Mail.ru`

> "*[...]the sandbox is a defense in depth mitigation[...]We definitely don't allow relaxing document.domain on www.dropbox.com[...]*" – `Dropbox.com`

> "*While this is an interesting area of research, are you able to demonstrate that this behavior is currently exploitable on Twitter? It appears that the behavior you have described can increase the severity of other vulnerabilities but does not pose a security risk by itself. Is our understanding correct? [...]We consider this to be more of a defensive in depth and will take into account with our continual effort to improve our CSP policy*" – `Twitter.com`

> "*I believe we understand the risk as you've described it.*" – `Imdb.com`

4 AVOIDING CSP VIOLATIONS

Preventing CSP violations due to SOP can be achieved by having the **same** effective CSP for **all** same-origin pages in a site, and prevent origin relaxation.

Origin-wide CSP: Using CSP for all same-origin pages can be manually done but this solution is error-prone. A more effective solution is the use of a specification such as Origin Policy [?] in order to set a header for the whole origin.

Preventing Origin Relaxation: Having an origin-wide CSP is not enough to prevent CSP violations. By using origin relaxation, pages from different origins can bypass the SOP [?]. Many authors

[6] https://hackerone.com

provide guidelines on how to design an effective CSP [?]. Nonetheless, even with an effective CSP, an embedded page from a different origin in the same site can use `document.domain` to relax its origin. Preventing origin relaxation is trickier.

Programmatically, one could prevent other scripts from modifying `document.domain` by making a script run first in a page [?]. The first script that runs on the page would be:

```
1 Object.defineProperty(document, "domain", {
      __proto__: null, writable: false,
      configurable: false});
```

A parent page can also indirectly disable origin relaxation in iframes by sandboxing them. This can be achieved by using **sandbox** as an attribute for iframes or as directive for the parent page CSP. Unfortunately, an iframe cannot indirectly disable origin relaxation in the page that embeds it. However, the **frame-ancestors** directive of CSP gives an iframe control over the hosts that can embed it. Finally, a more robust solution is the use of a policy to deprecate `document.domain` as proposed in the draft of Feature policy [?]. The feature policy defines a mechanism that allows developers to selectively enable and disable the use of various browser features and APIs.

Iframe sandboxing: Combining attribute **allow-scripts** and **allow-same-origin** as values for **sandbox** successfully disables `document.domain` in an iframe[7]. We recommend the use of **sandbox** as a CSP directive, instead of an HTML iframe attribute. The first reason is that **sandbox** as a CSP directive, automatically applies to all iframes that are in a page, avoiding the need to manually modify all HTML iframe tags. Second, the **sandbox** directive is not programmatically accessible to potentially malicious scripts in the page, as is the case for the **sandbox** attribute (which can be removed from an iframe programmatically, replacing the sandboxed iframe with another identical iframe but without the **sandbox** attribute).

Limitations An origin-wide CSP (the same CSP for all same origin pages) can become very liberal if all same origin pages do not require the same restrictions. In order to implement the solution we propose, one needs to consider the intended relation between a parent page and an iframe page, in presence of CSP. In the case where the two(2) pages should be allowed direct access to each other content, then, since same origin pages can bypass page-specific security characteristics [?], the solution is to have the same CSP for both the page and the iframe. However, if direct access to each other content is not a required feature, one can keep different CSPs in parent and iframe, or have no CSP at all in one of the parties, but their contents should be isolated from each other. The solution here is to use sandboxing. Nonetheless, there are other means (such as `postMessage`) by which one can securely achieve communication between the pages.

5 INCONSISTENT IMPLEMENTATIONS

Combining origin-wide CSP with **allow-scripts sandbox** directive would have been sufficient at preventing the inconsistencies between CSP and the same origin policy. Unfortunately, we have discovered that for some browsers, this solution is not sufficient. Starting from HTML5, major browsers, apart from Internet Explorer, supports the new **srcdoc** attribute for iframes. Instead of providing a URL which content will be loaded in an iframe, one provides directly the HTML content of the iframe in the **srcdoc** attribute. According to CSP2 [?], §5.2, the CSP of a page should apply to an iframe which content is supplied in a **srcdoc** attribute. This is actually the case for all majors browsers, which support the **srcdoc** attribute. However, there is a problem when the **sandbox** attribute is set to an **srcdoc** iframe.

Webkit-based[8] and **Blink**-based[9] browsers (Chrome, Chromium, Opera) always comply with CSP. The CSP of a page will apply to all **srcdoc** iframes, even in those iframes which have a different origin than that of the page, because they are sandboxed without **allow-same-origin** .

In contrast, we noticed that in Gecko-based browsers (Mozilla Firefox), the CSP of the page applies to that of the srcdoc iframe if and only if allow-same-origin is present as value for the attribute. Otherwise it does not apply. The problem with this choice is the following. A third party script, whitelisted by the CSP of the page, can create a **srcdoc** iframe, sandboxing it with **allow-scripts** only, and load any resource that would normally be blocked by the CSP of the page if applied in this iframe. This way, the third party script successfully bypasses the restrictions of the CSP of the page. Even though loading additional scripts is considered harmless in the upcoming version 3 [? ?] of CSP, this specification says nothing about violations that could occur due to the loading of other resources inside a **srcdoc** sandboxed iframe, like resources whitelisted by **object-src** directive for instance, additional iframes etc.

We have notified the W3C, and the Mozilla Security Group. Daniel Veditz, a lead at Mozilla Security Group, recognises this as a bug and explains:

> "*Our internal model only inherits CSP into same-origin frames (because in theory you're otherwise leaking info across origin boundaries) and iframe sandbox creates a unique origin. Obviously we need to make an exception here (I think we manage to do the same thing for src=data: sandboxed frames).*"

CSP specification and srcdoc iframes The problem of imposing a CSP to an unknown page is illustrated by the following example [?]. If a trusted third party library, whitelisted by the CSP of the page, uses security libraries inside an isolated context (by sandboxing them in a **srcdoc** iframe, setting **allow-scripts** as sole value for the **sandbox**) then, the page's CSP will block the security libraries and possibly introduce new vulnerabilities. Because of this, it was unclear to us the intent of CSP designers regarding srcdoc iframes. Mike West, one of the CSP editors at the W3C and also Developper Advocate in Google Chrome's team, clarified this to us:

> "*I think your objection rests on the notion of the same-origin policy preventing the top-level document from reaching into it's sandboxed child. That seems accurate, but it neglects the bigger picture: **srcdoc** documents are produced entirely from the*

[7]We found out that dropbox.com actually puts **sandbox** attribute for all its iframes, and therefore avoids the possible CSP violations. We have had a very interesting discussion on Hackerone.com with Devdatta Akhawe, a Security Engineer at Dropbox, who told us more about their security practices regarding CSP in particular.

[8]https://en.wikipedia.org/wiki/WebKit

[9]https://en.wikipedia.org/wiki/Blink_(web_engine)

> *top-level document context. Since those kinds of documents are not delivered over the network, they don't have the opportunity to deliver headers which might configure their settings. We impose the parent's policy in these cases, because for all intents and purposes, the* **srcdoc** *document* is *the parent document.*"

6 RELATED WORK

CSP has been proposed by Stamm et al.[?] as a refinement of SOP[?], in order to help mitigate Cross-Site-Scripting[?] and data exfiltration attacks. The second version[?] of the specification is supported by all major browsers, and the third version [?] is under active development. Even though CSP is well supported [?], its endorsement by web sites is rather slow. Weissbacher et al.[?] performed the first large scale study of CSP deployment in top Alexa sites, and found that around 1% of sites were using CSP at the time. A more recent study by Calzavara et al.[?], show that nearly 8% of Alexa top sites now have CSP deployed in their front pages. Another recent study, by Weichselbaum et al.[?] come with similar results to the study of Weissbacher et al.[?]. Our work extends previous results by analysing the adoption of CSP by site not only considering front pages but all the pages in a site. Almost all authors agree that CSP adoption is not a straightforward task, and lots of (manual) effort are needed in order to reorganize and modify web pages to support CSP.

Therefore, in order to help web sites developers in adopting CSP, Javed proposed CSP Aider, [?] that automatically crawl a set of pages from a site and propose a site-wide CSP. Patil and Frederik[?] proposed UserCSP, a framework that monitors the browser internal events in order to automatically infer a CSP for a web page based on the loaded resources. Pan et al.[?] propose CSPAutoGen, to enforce CSP in real-time on web pages, by rewriting them on the fly client-side. Weissbacher et al.[?] have evaluated the feasibility of using CSP in report-only mode in order to generate a CSP based on reported violations, or semi-automatically inferring a CSP policy based on the resources that are loaded in web pages. They concluded that automatically generating a CSP is ineffective. A difficulty which remains is the use of inline scripts in many pages. The first solution is to externalize inline scripts, as can be done by systems like deDacota[?]. Kerschbaumer et al.[?] find that too many pages are still using **'unsafe-inline'** in their CSPs. They propose a system to automatically identify legitimate inline scripts in a page, thereby whitelisting them in the CSP of the underlying page, using script hashes.

Another direction of research on CSP, has been evaluating its effectiveness at successfully preventing content injection attacks. Calzavara et al.[?] found out that many CSP policies in real web sites have errors including typos, ill-formed or harsh policies. Even when the policies are well formed, they have found that almost all currently deployed CSP policies are bypassable because of a misunderstanding of the CSP language itself. Patil and Frederik found similar errors in their study[?]. Hausknecht et al.[?] found that some browser extensions, modified the CSP policy headers, in order to whitelist more resources and origins. Van Acker et al.[?] have shown that CSP fails at preventing data exfiltration specially when resources are prefetched, or in presence of a CSP policy in the HTML meta tag, because the order in which resources are loaded in a web application is hard to predict. Johns[?] proposed hashes for static scripts, and PreparedJS, an extension for CSP, in order to securely handle server-side dynamically generated scripts based on user input. Weichselbaum et al.[?] have extended nonces and hashes, introduced in CSP level 2[?], to remote scripts URLs, specially to tackle the high prevalence of insecure hosts in current CSP policies. Furthermore, they have introduced **strict-dynamic**. This new keyword states that any additional script loaded by a whitelisted remote script URL is considered a trusted script as well. They also provide guidelines on how to build an effective CSP. Jackson and Barth[?] have shown that same origin pages can bypass page-specific policies, like CSP. Though, their work predates CSP. To the best of our knowledge, we are the first to explore the interactions between CSP and SOP and report possible CSP violations.

7 CONCLUSIONS

In this work, we have revealed a new problem that can lead to violations of CSP. We have performed an in-depth analysis of the inconsistency that arises due to CSP and SOP and identified three cases when *CSP may be violated*.

To evaluate how often such violations happen, we performed a large-scale analysis of more than 1 million pages from 10,000 Alexa top sites. We have found that 5.29% of sites contain pages with CSPs (as opposed to 2% of home pages in previous studies).

We have also found out that 72% of current web pages with CSP, are potentially vulnerable to CSP violations. This concerns 379 (72.46%) sites that deploy CSP. Further analysing the contexts in which those web pages are used, our results show that when a parent page includes an iframe from the same origin according to SOP, in 23.5% of cases their CSPs may be violated. And in the cases where `document.domain` is required in both parent and iframes, we identified that such violations may occur in 94% of the cases.

We discussed measures to avoid CSP violations in web applications by installing an origin-wide CSP and using sandboxed iframes. Finally, our study reveals an inconsistency in browsers implementation of CSP for `srcdoc` iframes, that appeared to be a bug in Mozilla Firefox browsers.

ACKNOWLEDGMENTS

The authors would like to thank the WebAppSec W3C Working Group for useful pointers to related resources at the early stage of this work, Mike West for very insightful discussions that considerably helped improve this work, Devdatta Akhawe for discussing some security practices at *Dropbox*, and anonymous reviewers and Stefano Calzavara for their valuable comments and suggestions.

REFERENCES

[1] Chrome Platform Status. https://www.chromestatus.com/metrics/feature/popularity#DocumentSetDomain.
[2] CSP violations online. https://webstats.inria.fr?cspviolations.
[3] Same Origin Policy. https://www.w3.org/Security/wiki/Same_Origin_Policy.
[4] S. V. Acker, D. Hausknecht, and A. Sabelfeld. Data Exfiltration in the Face of CSP. In X. Chen, X. Wang, and X. Huang, editors, *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2016, Xi'an, China, May 30 - June 3, 2016*, pages 853–864. ACM, 2016.
[5] S. Calzavara, A. Rabitti, and M. Bugliesi. Content security problems?: Evaluating the effectiveness of content security policy in the wild. In Weippl et al. [?], pages 1365–1375.
[6] A. Doupé, W. Cui, M. H. Jakubowski, M. Peinado, C. Kruegel, and G. Vigna. deDacota: toward preventing server-side XSS via automatic code and data separation. In A. Sadeghi, V. D. Gligor, and M. Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 1205–1216. ACM, 2013.
[7] D. Hausknecht, J. Magazinius, and A. Sabelfeld. May I? - Content Security Policy Endorsement for Browser Extensions. In M. Almgren, V. Gulisano, and F. Maggi, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment - 12th International Conference, DIMVA 2015, Milan, Italy, July 9-10, 2015, Proceedings*, volume 9148 of *Lecture Notes in Computer Science*, pages 261–281. Springer, 2015.
[8] A. Hidayat. PhantomJS Headless Browser, 2010-2016.
[9] C. Jackson and A. Barth. Beware of finer-grained origins. In *Web 2.0 Security and Privacy (W2SP 2008)*, 2008.
[10] A. Javed. CSP Aider: An Automated Recommendation of Content Security Policy for Web Applications. In *IEEE Oakland Web 2.0 Security and Privacy (W2SP'12)*, 2012.
[11] M. Johns. PreparedJS: Secure Script-Templates for JavaScript. In K. Rieck, P. Stewin, and J. Seifert, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment - 10th International Conference, DIMVA 2013, Berlin, Germany, July 18-19, 2013. Proceedings*, volume 7967 of *Lecture Notes in Computer Science*, pages 102–121. Springer, 2013.
[12] C. Kerschbaumer, S. Stamm, and S. Brunthaler. Injecting CSP for Fun and Security. In O. Camp, S. Furnell, and P. Mori, editors, *Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP 2016), Rome, Italy, February 19-21, 2016.*, pages 15–25. SciTePress, 2016.
[13] X. Pan, Y. Cao, S. Liu, Y. Zhou, Y. Chen, and T. Zhou. Cspautogen: Black-box enforcement of content security policy upon real-world websites. In Weippl et al. [?], pages 653–665.
[14] K. Patil and B. Frederik. A measurement study of the content security policy on real-world applications. *I. J. Network Security*, 18(2):383–392, 2016.
[15] N. Perriault. CasperJS navigation and scripting tool for PhantomJS, 2011-2016.
[16] G. Rydstedt, E. Bursztein, D. Boneh, and C. Jackson. Busting frame busting: a study of clickjacking vulnerabilities at popular sites. In *in IEEE Oakland Web 2.0 Security and Privacy (W2SP 2010)*, 2010.
[17] K. Singh, A. Moshchuk, H. J. Wang, and W. Lee. On the incoherencies in web browser access control policies. In *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA*, pages 463–478, 2010.
[18] D. F. Some, N. Bielova, and T. Rezk. On the Content Security Policy violations due to the Same-Origin Policy. Technical report. http://www-sop.inria.fr/members/Nataliia.Bielova/papers/CSP-SOP.pdf.
[19] S. Stamm, B. Sterne, and G. Markham. Reining in the web with content security policy. In M. Rappa, P. Jones, J. Freire, and S. Chakrabarti, editors, *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 921–930. ACM, 2010.
[20] N. Swamy, C. Fournet, A. Rastogi, K. Bhargavan, J. Chen, P. Strub, and G. M. Bierman. Gradual typing embedded securely in JavaScript. In S. Jagannathan and P. Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 425–438. ACM, 2014.
[21] A. van Kesteren. Cross Origin Resource Sharing. W3C Recommendation, 2014.
[22] L. Weichselbaum, M. Spagnuolo, S. Lekies, and A. Janc. CSP is dead, long live csp! on the insecurity of whitelists and the future of content security policy. In Weippl et al. [?], pages 1376–1387.
[23] E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016.* ACM, 2016.
[24] M. Weissbacher, T. Lauinger, and W. K. Robertson. Why Is CSP Failing? Trends and Challenges in CSP Adoption. In A. Stavrou, H. Bos, and G. Portokalidis, editors, *Research in Attacks, Intrusions and Defenses - 17th International Symposium, RAID 2014, Gothenburg, Sweden, September 17-19, 2014. Proceedings*, volume 8688 of *Lecture Notes in Computer Science*, pages 212–233. Springer, 2014.
[25] M. West. Content Security Policy: Embedded Enforcement, 2016.
[26] M. West. Content Security Policy Level 3. W3C Working Draft, 2016.
[27] M. West. Origin Policy. A Collection of Interesting Ideas, 2016.
[28] M. West, A. Barth, and D. Veditz. Content Security Policy Level 2. W3C Candidate Recommendation, 2015.
[29] M. West and I. Grigorik. Feature Policy. W3C Draft Community Group Report, 2016.
[30] I. Yusof and A. K. Pathan. Mitigating Cross-Site Scripting Attacks with a Content Security Policy. *IEEE Computer*, 49(3):56–63, 2016.