UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Complex Security Policy? A Longitudinal Analysis of Deployed Content Security Policies

Samrat Dutta

Seminar paper – July 17, 2020.
System Security Group.

Supervisor: Prof. Dr-Ing. Juraj Somorovsky

Abstract

`Content Security Policy (CSP)` was introduced as a security measure against Cross Site Scripting attacks in 2010. The paper 'Complex Security Policy?' by Roth et al. describes a longitudinal analysis of a large set of the most popular websites from 2012 in order to understand the deployment, evolution and use cases of `CSP` over the years. It discusses the struggles that the developers face while rolling out meaningful security policies for content restrictions and how even secure looking policies may be bypassed because of non maintenance of whitelists or typos. The paper also discusses how `CSP` has outgrown its main use case and is increasingly deployed for both TLS enforcement and for framing control. The authors analyse why `X-Frame-Options` is still more widely used even though it is underspecified and inconsistently implemented over `CSP`'s `frame-ancestors`. Finally, a short survey reveals how website operators are mostly unaware of `CSP`'s framing control features and that it can be used in isolation. The survey reveals that many developers find the deployment of `CSP` to be challenging and complex and hence avoid it.

Contents

1 Introduction

The internet is constantly making the world a more connected place, and over the past few years, almost all aspects of our lives like education, shopping, banking, social networking etc have gone online. As useful as it may be, we need to be mindful that we share the internet with an estimated 4.1 billion people [6], some of whom are malicious. Hence, the security of the web is of utmost concern to the developers and the end-users alike. Website developers have no control over the client devices where their code is executed and rendered, which is a gap that attackers exploit by injecting harmful scripts to the original response. This leaves the unsuspecting end user vulnerable to identity theft among other threats. But since almost the entire share of web browsers used world wide is shared among a handful of vendors[4], this presents us an unique opportunity for developing specifications that can be implemented by these browsers, giving some control to the web developers over how their websites are processed. To this end, browser vendors offer several opt-in security features in an attempt to protect the confidentiality and to stop malicious attacks.

Content Security Policy `CSP` is one such specification that is delivered through HTTP headers and is aimed at letting the website administrator control the resources that a webpage can load. [1] `CSP` is mainly used as a counter measure against `cross-site scripting (XSS)` attacks, data injection attacks etc. This is done by employing content restriction in the webpages. Apart from content restriction, the latest iterations of `CSP` also offers TLS enforcement and Framing control. We discuss these use cases in more detail in a later section.

Motivation The goal of this paper is to discuss and analyse the real-world implementations of `CSP`. Especially to understand how the deployment has evolved over time and look at the use cases that most operators use `CSP` for. This provides us with valuable insights into the requirements and the problems that operators face when deploying policy and allows us to analyse how the those policies affect the overall security of their website. This paper and the following findings are based on a paper written by Roth et al. Complex Security Policy? A Longitudinal Analysis of Deployed Content Security Policies. [3]

Structure

- We start with a background on `CSP` and it's use cases.
- Analyse the methodology used to arrive at the results.
- Discuss `CSP` deployment in light of Content Restriction and how a potential attack is often ignored.
- Talk about the gradual shift towards using `CSP` for TLS enforcement.
- Compare Framing Control features with the widely used but less secure `X-Frame-Options`.
- Then we discuss the authors' analysis of the sites that have abandoned `CSP` as well as those who implement it successfully, to identify the roadblocks to the deployment of a secure policy.

The longitudinal analysis is done by taking advantage of the Internet Archive to obtain historical `CSP` Headers (from 2012 to 2018). This gives us a view into how `CSP` was adopted by various websites, their struggles with it as well as the effects.

2 Background

Content Security Policy is a security specification delivered to the client via HTTP Headers and enforced by the web browser. It was first proposed in 2010 by Stamm et al. Policies are specified inside the `Content Security Policy` header or by including them in the `<meta>` tag in HTML pages. `CSP` can be deployed in an enforcement mode or a report-only mode. When enabled, reports can also be sent to an uri while in enforcement mode.

Need The primary goal of CSP was to mitigate Cross-Site-Scripting (XSS) and other data injection attacks. XSS is a type of attack where a malicious user injects client executable code in an otherwise legitimate website. Similar attacks that involve the injection of data from a third party when the website is loaded can be mitigated by restricting the sources of external content. This was the primary goal of CSP. Along with content restriction, `CSP` has developed other use cases, all of which are discussed in the next sections.

2.1 Content Restriction

Content Restriction being the main goal was specified in the very first version of `CSP`. Developers are allowed to maintain a list of regular expressions resolving to web origins indicating them as secure sources, for a type of resource. Along with that, the keywords ’self’ and ’none’ can also be used to only allow resources from the same origin as the actual document, and to deny any origin including itself, respectively. `t-src` (t : type of resource) indicates secure sources for resources of type ’t’ while `default-src` whitelists sources for all other resource types.
Event handlers and inline scripts are automatically when using these directives. Some of these restrictions can, however, be relaxed by using the `unsafe-inline` or `unsafe-eval` in the `script-src` directive or `default-src` directive in its absence. This however was not secure and defeated the purpose of content restriction.
`CSP Level 2` introduced hashes and nonces. Static scripts can be hashed using `sha256`, `sha384` and `sha512` and the hash can be added to the `script-src` directive.[2]

```
Content-Security-Policy: script-src ’sha256-B2yPHKaXnvFWtRChIbabYmUBFZdVfKKXHbWtWidDVF8=’
```

Dynamic scripts can be tagged with a randomly generated nonce and then the nonce can be added to the `script-src` directive.

```
Content-Security-Policy: script-src 'nonce-2726c7f26c'

<script nonce="2726c7f26c">
  var inline = 1;
</script>
```

Event handlers are still not supported by either of these methods. Also, if nonces and hashes are used, `unsafe-inline` is ignored.
`CSP` Level 3, which is the latest version, released in 2016 further relaxes the constraints by adding the `script-dynamic` expression which when used with hashes and noonces allows scripts with valid hashes to load other scripts which might not have a hash/noonce added to the directive of the original document.

2.2 TLS Enforcement

The enforcement of TLS was not one of the main targets of `CSP`. But over time, `CSP` has extended to handle mixed content. It allows the developer to add a `block-mixed-content` directive in the `CSP` header which forbids the browser explicitly from loading content over less secure channels like HTTP when the main content is served over HTTPS. Another alternative for developers is to use `upgrade-insecure-requests` which forces the browser on page load, to change all HTTP urls into HTTPS. Thus ensuring that all content to the page is loaded via secure connections. This is particularly useful to websites transitioning to HTTPS.

2.3 Framing Control

`CSP` also provides framing control which protects against attacks like Clickjacking. The existing and widely used headers like `X-Frame-Options` do not allow much flexibility and moreover is underspecified, leading to security risks. `CSP` improves framing control by giving the operator more options when choosing to allow specific sites or domains framing rights. This is achieved by using the framing-ancestors directive along with an arbitrary list of source expressions to be allowed.

```
Content-Security-Policy: frame-ancestors <source>;
```

`CSP` also provides security by enforcing the specification on the full chain of ancestors while `X-Frame-Options` mandates only checking the framing website.

3 Methodology

The Internet Archive is like a time machine for websites which gives us a glance into their past states, entire pages along with their headers over the years. It acts as a resource for observing and analysing how the top 10000 most popular websites on the internet used `CSP` headers over an extended period of time.

Gathering Data The authors of the referred paper used Alexa to obtain a set of websites that were among the 50000 most visited websites for each month for at least one day in that month, over a period of almost 7 years (January 2012 to December 2018). They then took the intersection of these sets and ordered them by their average ranking (most visited). The authors used the top 10000 websites for further analysis, to limit the number of queries they would otherwise have to make to the Internet Archive to conclude the study. They then collected the headers from the daily snapshot of each website from January 1, 2012, to December 31, 2018. Since IA removes `meta` tags, the sites that deploy `CSP` through `meta` were not accounted for in the results. Of the 2557 days of gathering data, on average 1031 daily snapshots could be obtained per site. To account for the missing days, the header configuration of the last day before a gap was taken into consideration.

Validity Incorrect archival data in IA could mean an invalid dataset and hence a faulty analysis. The authors used another archival service, Common Crawl to cross-check the validity of headers. They found 38,129 overlapping snapshots with only 729 (1.9%) with inconsistencies between the sites. Out of those, in 96 cases, it was the lack of a `block-all-content` or `upgrade-insecure-requests` directive which was attributed to how Common Crawl (a bug) stores the headers being different. Another 29 cases were attributed to whitelisting CDNs based on the crawler's IP. Though for the remaining cases the exact cause was not determined, it should be noted that in only 238 (0.6%) of all the cases, the use case of the `CSP` differed.
`CSP` can also be deployed using the meta tag. To assess the extent of usage, the authors crawled through all the live web-pages from the shortlisted dataset on June 10, 2019 and collected `CSP` headers and `CSP` meta elements. They found that of all the websites, 1,206 deployed `CSP` and out of those 78 (6.4%) set the policy only through meta tags.
In light of this, it is safe to say that the dataset analysed can be used to draw general conclusions about the deployment of `CSP` throughout the internet.

4 Observations

Adoption CSP adoption started gaining momentum only after 2014 (previously only 8 websites used CSP). The authors noted that out of the 10,000 websites analysed during the study, 1,233 used CSP in enforcement mode for at least one day while in the last month of analysis, only 1032 sites enforced CSP. The report only mode (CSP-RO) reports violations instead of blocking content. This was intended for the developers to test their policies before deploying it in enforcement more. The number of sites using the CSP-RO mode (or both) was significantly less compared to the deployments in enforcement mode. This indicates that the developers were rolling out security policies without first testing them on their user base. This could also potentially indicate that developers are unaware of the benefits of the Report Only mode. (Fig. 1)

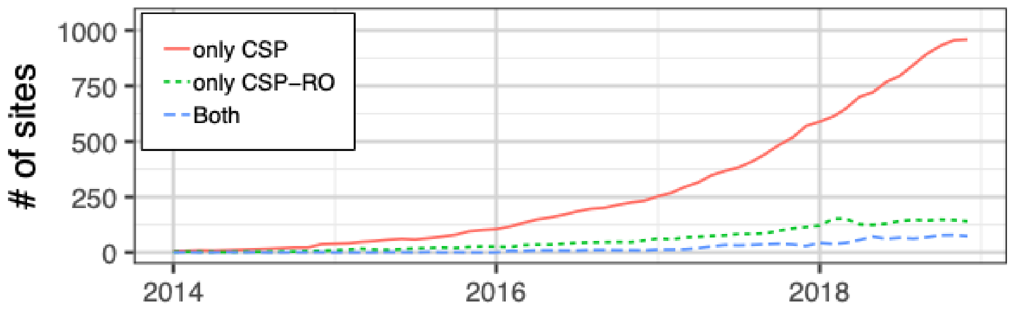

Fig. 1

Maintenance Unlike other header based security mechanisms like HSTS, that are deployed once and remain unchanged for long periods, CSP requires constant maintenance of the whitelists to keep the website secure and functional. This means that for a proper implementation, the CSP headers need to be modified regularly. This matches with the observed data gathered by the authors.

Use Cases The original intended use for Content Security Policy was to mitigate Cross Site Scripting (XSS) attacks. But CSP has extended itself to support many other use cases. From the data collected by the author of the referred paper, it is evident that the increase in deployments in 2015 and 2017 coincided with deployments of Framing Control and TLS enforcement respectively. This clearly shows a shift of usage of CSP from only a Content Restriction usage to TLS enforcement and Framing Control. In the next sections we discuss the effects in each of these catagories.

4.1 Deployment for Script Content Restriction

Of 1,032 sites that were still using `CSP` at the end of the analysis period, only 421 used `CSP` to restrict content. So effectively only 41% of total deployments used `CSP` for its original purpose : content restriction. And among the deployed policies, further analysis reveals security gaps which are discussed in the following sections.

Insecure Practices The extensive use of the `unsafe-inline` and `unsafe-eval` is a huge threat to the application security. Even at the end of the analysis period, the author found that 90% of all the websites using `CSP` for content restriction, used an `unsafe-inline` directive which can be attributed to the developers accommodating event handlers which would otherwise not function. Of the 378 website start pages checked, which employ `unsafe-inline`, 48% carried event handlers. Also the practice of white-listing entire schemes (HTTP/HTTPS) is another security violation that's prevalent.
Along with these issues, the very low (5%) adoption rate of the new policies such as hashes, nonces and the `strict-dynamic` directive shows that the majority of the `CSP` deployments have security vulnerabilities.

White-listing As we have seen, developers often resort to insecure practises and whitelist entire schemes. This along with the trend in the data curated by the author shows that the developers are increasingly trusting low ranked third party websites via their CSP headers to host JavaScript code. 89% of the sites restricting content whitelisted at least one domain for script loading. This opens up the possibility of a security threat.

- Expired Domains : Expired domains that are not removed from whitelists enable the new owners serve content as it acts as a trusted source.
- Typos : Typo domains may actually be owned by someone who may use the misplaced trust to serve malicious content to the unsuspecting website. The author found 11 instances of potential typo domains in the analysed dataset.
- Local Addresses : From the dataset, 15 whitelisted domains from 26 sites resolved to private addresses. This means that anyone on the same LAN as the user, has the ability to serve content to the website.

Of the 373 websites using whitelists, 50 (13%) whitelisted at least one domain that could be abused. Avoiding this security vulnerability is complicated and requires regular maintenance of the whitelists. (`Fig. 2`)

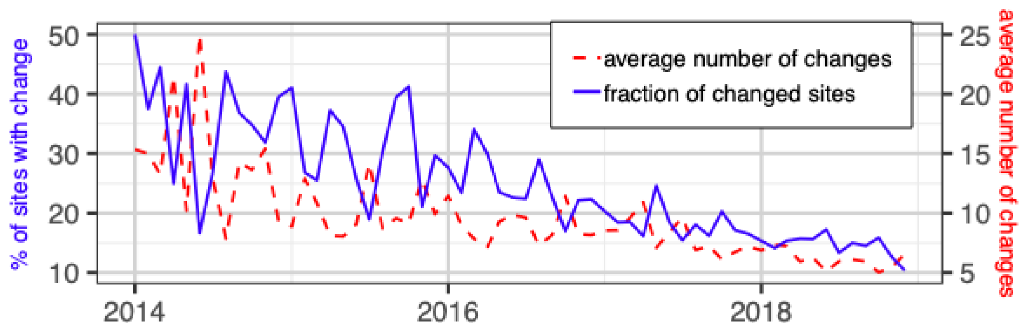

Fig. 2

4.2 Deployment for TLS Enforcement

`CSP` is also used to ensure that HTTPS do not load HTTP resources. Policies specifically targeting TLS started becoming popular after December 2016, especially with the use of `upgrade-insecure-requests` directive. Use of `block-mixed-content` also increased during this period. It must be noted that `block-mixed-content` has no function when used with `upgrade-insecure-requests` in modern browsers as the later upgrades all HTTP connections to HTTPs as soon as the site loads.
A related but different header is the `HSTS` header which ensures that a site is not loaded via HTTP for a set timeout once it has been loaded via HTTPS. The use of HSTS has also risen in the last few years.[**Usenix:2017**]
194 of the 347 websites deploying `upgrade-insecure-requests` also deployed `HSTS` in an effort to ensure that only HTTPS content is loaded after the website is loaded over HTTPS.
Instead of blocking HTTP content outright in an HTTPS site, this header gives the developer to update all the HTTP urls without having to change the code. 251 (72%) of the websites used `upgrade-insecure-requests` while transitioning into HTTPS. Of these, in 77 sites, the author finds resources still linked using HTTP urls after a month from the switch. This shows that `upgrade-insecure-requests` header is of great utility to website developers.

4.3 Deployment for Framing Control

`X-Frame-Options` is widely used as the header of choice when it comes to framing control. But `XFO` is under-specified and it's implementations differ from browser to browser due to the lack of standardization. This leaves websites vulnerable to double framing attacks. The CSP alternative frame-ancestors is much more secure as it mandates enforcing the policy on the entire chain of ancestors. Yet we see a constant growth in the use of XFO till the end of the analysis period.

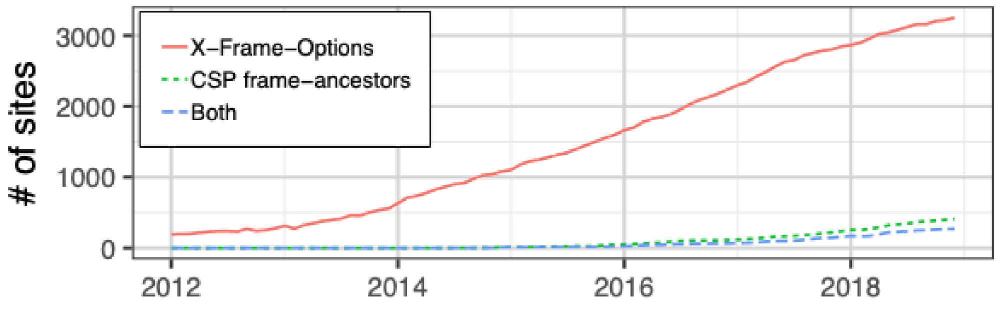

Fig. 3

Flexibility XFO only has 3 options to control which websites can frame website in question. It allows DENY, SAMEORIGIN and ALLOW-FROM. CSP frame-ancestors offers more flexibility by letting the developer use regular expressions to allow multiple domains, most notably entire websites instead of only the particular origin of the page. This is especially useful as we see that this header is often used to whitelist sites from the same company. The author finds that 50% of the websites that implemented CSP-FA from the start of 2015, required the flexibility offered by CSP.

Simultaneous Usage In CSP compatible browser, frame-ancestors directive mandates any XFO directive to be ignored. But on older browsers that do not comply to CSP, XFO is the only security mechanism. In the analysis period 394 websites used frame-ancestors and XFO simultaneously. 290 of them had CSP and XFO headers with different targets. Of those, 70 websites relaxed the SAMEORIGIN policy to allow content from the same site.

Replacing XFO with CSP-FA The risk of double framing attacks has mostly been mitigated in most modern browsers but still because the directive ALLOW-FROM is completely ignored by safari, chrome and browsers based on chrome, using XFO can be dangerous. In December 2018, 116 sites from the dataset made use of XFO directives that are not implemented universally. They require a CSP header to ensure security. Furthermore, it has to be noted that 362 of 3,253 websites at that time which used XFO but no frame-ancestors, did use other CSP directives.

5 Analysis

We now analyse the observations in the light of sites that gave up on implementing `CSP`, sites that continued implementing insecure policies and sites that have been able to deploy secure and effective policies.

5.1 Giving up on CSP

The most intuitive reason why sites may abandon a security feature is that their content kept violating the policy and that they could never find the right balance between allowing secure content using meaningful policies. We analyse violations in sites that abandoned `CSP` for each use case.

5.1.1 Content Violations

By the end of the analysis period, 63 websites had abandoned `CSP` after having tried to enforce content restrictions. Out of those, 15 showed violations of the deployed `CSP`. Out of these 15, 9 pages had third party code that violated the policy. This analysis was only done on the start page of the websites due to a lack of resources. But since 1024/1202 (85%) of the websites have site-wide `CSP`, it is reasonable to assume that these operators abandoned `CSP` due to violations on other pages deeper in the website. On checking manually, 25 websites showed an increasing number of third party origins. This requires regular maintenance of the whitelist which can become really challenging for operators. This could also be a reason why the operators choose to give up on `CSP`

5.1.2 TLS Violations

`TLS` restriction can be violated in 2 ways. Simply by including an Http resource while having the `block-all-mixed-content` or `default-src https://*` headers, `CSP` is violated. But the more difficult to analyse violations are the `upgrade-insecure-requests` violations. The IA crawler does not check for Https requests in place of Http even if there is an `upgrade-insecure-requests` present. This makes analysis through this method challenging. To overcome this, the authors devised a strategy: they assumed that if a site is loaded via https, that implied that all the resources from the same site would also be available over https. They then used the `CDX API` to find out if Http resources were upgradable in a range of 30 days before and after the date. Any resource that did not fall in the discussed categories was marked non-upgradable. 28 out of the 46 domains that gave up on `TLS` enforcement were flagged as having non-upgradable resources. In 4 more cases, all `CSP` headers had been dropped which could indicate that abandoning `TLS` enforcement could be a side effect in those cases.

5.1.3 Framing Violations

Framing violations are not feasible to be detected with the current methodology since it requires figuring out which websites tried to frame each website in question. Instead, the authors utilized the wide coexistence of frame-ancestors and `XFO` headers to understand these violations. They tried to find alterations or removal of `XFO` headers on or the day after the `frame-ancestors` header was dropped; which would clearly indicate the possibility of problems with framing permissions. The authors found out that in their observations, 69 sites dropped `CSP` for framing controls out of which, 42 dropped frame-ancestors and `XFO` at the same time, indicating a general problem with framing. 7 sites started using `XFO same-origin` in the place of explicit `CSP` headers while in 7 other cases the header was dropped alongside other `CSP` headers indicating collateral damage.

5.2 Insecure Policies

Having an insecure policy like `unsafe-inline` or an entire schema in the whitelist, 455 (97%) sites had inline scripts in their start page at least once and 317 sites (68%) made use of event handlers. Sites relied on up to 26 third parties for content. As event handlers are not possible to use without an unsafe-inline header, this means that 68% of the sites did not have the option of deploying a secure `CSP` without stopping the use of event handlers.

5.3 Secure Websites

The authors found 40 sites during their research which had deployed a secure `CSP` policy and continued to do so till the end of the observation period. An additional 87 websites had at some point in time deployed a secure `CSP` but later changed them to an insecure version. 2 out of the 40 sites which deployed strict policies have violating conflicts in their start page, but still continue to use the header. They also noted that 16 adult websites had deployed `CSP` in a meaningful way and most of them did so without having a more relaxed policy before.

6 Further Investigations

The results clearly showed a bias towards `XFO` headers for framing restrictions than `CSP`'s `frame-ancestors`. The authors notified website operators who were still using `XFO` about the advantages of using `frame-ancestors`, in order to understand why they chose one over the other. They monitored the deployed `XFO` and `CSP` directives on sites in their dataset from May 31, 2019, and sent out notifications on June 4, 2019, to 2,699 sites which either had a syntactical error in their frame control header deployment or used an `allow-from` directive or had a `same-origin` policy all of which are insecure.

6.1 Response

Of the 2,699 sites contacted, the authors received replies from 117 sites, which were not automated replies. Of those, 62 replied saying that they were going to deploy the `frame-ancestors` directive shortly. 24 responses indicated that `CSP` was too complex to deploy. Upon further communication, 16 operators stated that they were unaware of any drawbacks of `XFO` and 13 who claimed to have never heard about `frame-ancestors`. The authors continued monitoring the frame-ancestors and `XFO` headers after the notifications and found out that by June 12, 2019, the number of sites using frame-ancestors had grown to 554 from 511 (14 sites from those who had responded to the notification had adopted `frame-ancestors`.)
Several operators indicated that external resources which advocate `XFO` as the only defense against framing attacks, as their reference for the deployment of security headers. In these external resources, `CSP` is only advertised as a means to mitigate `XSS attacks`.

6.2 Follow up survey

The authors conducted a targeted follow up survey involving only the operators who had previously responded in order to understand their familiarity with the `XFO` issues, `CSP`, and the deployability of `frame-ancestors` in isolation. Out of the 39 answers that they received, 27 indicated unawareness of the issues with `XFO`. Although 20 indicated knowledge about framing attacks. 31 responses claimed to be aware of `CSP` but only 12 knew about `frame-ancestors` from whom only 9 knew that it was possible to use it in isolation. 23 operators of these 31 responses admitted that if they used a secure policy right then their website would not function correctly. The results of this survey indicate that the usefulness of `CSP` for content restriction is well known but if often avoided because of its complexity. Operators were well aware of `TLS` enforcement using `CSP` but were largely unaware of framing controls especially about enforcing them using `CSP`.

7 Discussion

7.1 Summary of Use Cases

The authors' analysis shows that most sites use the easily bypass-able `unsafe-inline` header instead of using `nonces` and `hashes`, making the websites vulnerable to `XSS` attacks and thus rendering `CSP` for content restriction useless. It also shows that maintaining whitelists is time and resource-intensive as the list keeps changing very frequently. This is unfeasible for most of the operators. The authors also found out `CSP` is popularly used for `TLS` enforcement. One-third of all the sites that deployed `CSP`, used `CSP` to enforce TLS. The header upgrade-insecure-request is often used not only for security but during migration which automatically changes all `Http` links in the webpages to https links without raising mixed-content warnings. They also found out that the usage of `CSP` for framing control has grown significantly over time. But still, `XFO` is adopted much more widely than `CSP` with `frame-ancestors`. The notification survey leads us to believe that operators are unaware of the `frame-ancestors` directive and that it can be used in isolation and also are unaware of the issues related to `XFO`

7.2 Too Complex?

Content restriction is tricky because the target does not remain the same. `Google.com` moved away from `strict-dynamic` on July 17th, 2018, preferring instead to use `nonced` scripts attaching `nonces` whenever additional scripts are attached. [7] The site then sets another `CSP` with a whitelist from which scripts can be executed. Browsers need to satisfy both the `CSP`s hence, this set up is secure. It should be noted that this setup needs frequent changes, especially the maintenance of the whitelist. This is often too complex for most operators to keep up with.

The longitudinal analysis suggests that operators often avoid using `CSP` despite its multiple security benefits which could potentially be due to the perceived complexity of the policy. On top of that new features for content-restrictions are being added as well as directives like navigate-to which are only making `CSP` more complex. This perceived complexity is a deterrent for the operators who would otherwise benefit from using `CSP`

7.3 Actionable Steps

As a result of the analysis, the authors identified 3 actionable steps that could help CSP being adopted more widely.

1. **unsafe-nonced-elements** : To tackle the problem of blocking of event handlers, the authors suggest that `CSP` be extended to support event handlers for `nonced` elements but not their children. If this is used then `unsafe-inline` can be avoided. This could potentially be vulnerable to nonce reuse attacks or to injections inside nonce elements. The `CSP` standard authors came up with the solution that a `nonced script` is only executed if it does not contain another opening `script` tag inside it.[5] This can then be extended to other elements. The other attack is to inject event handlers inside `nonced` elements. Even if this leaves some vulnerability, it is significantly less than using unsafe-inline.

2. **Incorporate CSP into the development cycle** : The analysis clearly shows that developers have struggled to retrofit `CSP` headers in their existing application and ended up either with an insecure `CSP` deployment or abandoning `CSP` completely. In order to counter these problems, the authors suggest incorporating `CSP` into the development cycle. This can be done by IDEs checking `CSP` compatibility during development and by discouraging use of inline scripts or event handlers. Additionally, third parties that very frequently add additional scripts to a website should declare their dependencies explicitly and their interference with `CSP` which then can be used by the IDE to warn the developer and give him other options.

3. **Updated Informational Materials for Developers** : `CSP` is often shunned as it is perceived as difficult to implement. But there exist use cases like `TLS` enforcement and framing control which can work in isolation and can be deployed easily. But this information is widely circulated in popular internet security resources. So the authors advocate for clearer communication from these resources to educate the developers about all the use cases. The browsers can also help to a certain extent by printing warning on the browser console when using headers like `XFO`.

8 Conclusions

The paper discusses a longitudinal analysis of over 10000 highly ranked websites from 2012 using the Internet Archive's repository to understand the deployment and evolution of `CSP`. The authors found out that though `CSP` was initially introduced for content restrictions, it has evolved to cover multiple use cases such as TLS enforcement and framing control. The analysis highlighted that operators often face issues when trying to enforce content restrictions on their website using `CSP`. This is often due to the complexity in deployment and the resource and time intensive maintenance that is required to maintain a secure policy. Moreover, new directives like `strict-dynamic` are not widely adopted, which indicates that `CSP` for content restriction is not a success. However, `CSP` is increasingly being adopted for TLS enforcement and framing control. But still, the numbers are unsatisfactory. The authors conducted a survey that clearly indicates that due to the bad reputation of `CSP` as difficult to implement, developers avoid even the easy to implement and deploy parts of `CSP` and thus `CSP`'s utility in providing security is largely underutilized.

Bibliography

[1] MDN contributors. *CSP*. 2020 (accessed May 14, 2020). URL: `https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy`.

[2] MDN contributors. *CSP: script-src*. 2020 (accessed May 14, 2020). URL: `https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/script-src`.

[3] Sebastian Roth et al. *Complex Security Policy? A Longitudinal Analysis of Deployed Content Security Policies*. 2019. URL: `https://publications.cispa.saarland/2986/`.

[4] *Browser Share*. URL: `https://gs.statcounter.com/browser-market-share`.

[5] *Prevent nonce stealing by looking for "<script" in attributes of nonced scripts*. URL: `https://github.com/w3c/webappsec-csp/issues/98`.

[6] Statista. *Internet Usage*. URL: `https://www.statista.com/statistics/617136/digital-population-worldwide/`.

[7] L. Weichselbaum and M. Spagnuolo. *CSP - A Successful Mess Between Hardening and Mitigation*. URL: `https://tinyurl.com/yyohn6o6`.