Evaluating Website Security: A Study on Content Security Policy Implementation

Isabella Fernandes de Oliveira, Mengxia Ren
Colorado School of Mines – Computer Science Department

Content Security Policies (CSPs) are vital defenses against cross-site scripting (XSS) and unauthorized web resource manipulation. This study investigates CSP implementation across the top 100 websites listed in TRANCO, focusing on resilience to external manipulation. Our analysis reveals that over 50% of domains lack adequate protection against XSS attacks, despite CSP implementation. Vulnerabilities include the misuse of 'unsafe-inline' directives and reliance on white-listing-based policies over nonce-based alternatives. These findings highlight the need for a comprehensive CSP approach, prioritizing script and web control. Strategies to enhance website security, such as stricter CSP configurations, are discussed, emphasizing nonce-based strategies and comprehensive directive coverage.

Background

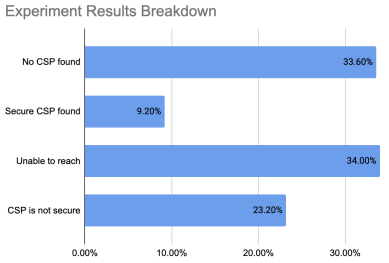
Content Security Policies **(CSPs)** allow web developers to define and enforce a set of rules that dictate which resources can be loaded and executed by a web page. These resources include scripts, stylesheets, fonts, images, and more. By specifying the allowed sources for each type of content, CSP helps to prevent malicious scripts from executing and unauthorized content from being loaded, thereby **enhancing the security of web applications.** CSP works by instructing the web browser to only execute or render resources from trusted sources, as defined by the website's policy. Any attempt to load content from unauthorized origins is blocked, reducing the risk of XSS attacks and other security exploits.

CSPs use different directories to specify which types of resources to let through. For example, the line ***script-src 'self'*** details that the only scripts allowed should be the ones from the original domain itself.

```
// An ideal, secure CSP
default-src 'none';
frame-ancestors https://dzen.ru;
connect-src 'self';
script-src 'nonce-8bc...' 'self'
'strict-dynamic' 'unsafe-inline';
img-src 'self'
```

Analysis Breakdown

When looking at the 100 websites listed as the most resilient to external manipulation, 34% of them were Content Delivery Networks which are simply used to direct traffic. This means that these websites are not open to the public and direct specified content that does not need to be protected. 33% of websites had no CSPs found at all, but 100% of those could successfully apply a general policy through the CSP Generator. **32% of websites had CSPs deployed, but only 9% of them were deemed secure upon further inspectio**n. This information is displayed in the graph on the right. The 23% of websites with unsafe policies were grouped into 5 types of issues: small scope of the policy, use of wildcards, use of 'unsafe-eval,' use of 'unsafe-inline' without the use of 'strict-dynamic,' and white-list-based policies instead of nonce-based.

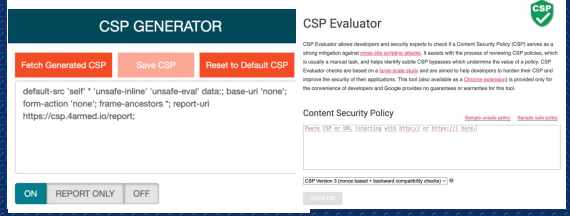


Conclusions

Given the results of our experiments, we can conclude that, while many websites may be deemed secure because of their deployed CSP, **only a third of them are truly safe** from cross-site scripting. Our problem analysis was to **ensure deployment effectiveness** and that policies were compatible with every level of CSP. We've seen that **over 60% of websites with CSPs have unsafe policies** involving their scope of coverage, white-listing-based policies, and the use of unsafe qualifiers like wildcards and unsafe-inline/eval. This presents a great issue for the utilization of Content Security Policies and may signify a need for more comprehensive directories to ensure security regardless of how it is used. The use of wildcards presents the largest issue and should be the first to be addressed. Given these experiments, we recommend that the next CSP level provides other wildcards besides * which will represent different allowed domains (for example, **using % to allow any HTTP or HTTPS domain**). The next most concerning issue is the use of 'unsafe-eval' without a restrictor (like the case for 'strict-dynamic' and 'unsafe-inline'). We recommend the case where a restrictor can be applied where eval() is only allowed in specific domains: for example, **unsafe-eval restrictor: 'self',** meaning unsafe-eval would only be applied to resources from the original website domain. In the case where websites have too small of a scope for their policy, a **warning should be implemented in the policy report**. This issue is ultimately a decision for the website administrator so there is only so much the policy can do. Policies can direct a report of use for the admin, so this will give the user a warning and possibly advice on how to improve their policy coverage.

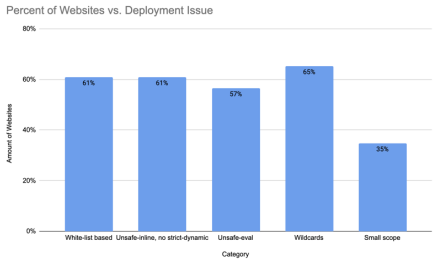
Experiment Design

The types of attacks we are most concerned with are regarding the 'script-src' directive. By not properly securing this directory, **any 3rd part resource regarding scripting can be compromised**, and it will still be read in. This applies more directly to whitelisting/nonces, wildcards that allow any 3rd party resource in, the use of inline scripts, and the use of 'unsafe-eval' which allows any evaluation of text lines as code[1]. Our experiments were done using the first 100 websites from the TRANCO list[4]. This list details the websites most hardened against outside manipulation. Once the landing page is found, the CSP Evaluator[3] checks if the domain uses a Content Security Policy. If so, the Evaluator displays some areas where the policy may be unsafe. If not, the CSP Generator is used to find the strictest policy the website can support. The one drawback of the Generator is that it limits the use of nonces or hashes. This experiment was used to **detail the most common CSP deployment issues** in websites that get a lot of traffic.



CSP Deployment Issues

One of the largest issues found was the use of 'unsafe-inline' (which allows for inline scripts) without the **restriction from 'strict-dynamic' which will override the 'unsafe-inline'** for browsers that support this level of CSP[2]. The use of 'unsafe-eval' is a similar, if not more troubling, problem as it has no override and allows for **the evaluation of any text within any 3rd party resource** to be loaded as long as it uses the eval() command. Wildcards present the most encountered issue and are especially troubling since they **allow any resource from any domain** to be loaded into the website. The use of wildcards and whitelists provide issues comparable to each other as they may allow **unsafe resources to be loaded without outright permission** from the website administrator. The small-scope problem relates to the coverage of the policy as it relates to directories. This only applies to 35% of the websites discussed here but it presents an issue of itself as policies that do not cover 'script-src' are **not prioritizing script and web control safety** and are completely vulnerable to cross-scripting attacks for scripting and web page control.




```
// Low coverage CSP & use of wildcards
frame-ancestors 'self' *

// 'Unsafe-inline' and 'Unsafe-eval'
block-all-mixed-content;
script-src 'self' data:
'unsafe-inline' 'unsafe-eval' https: blob: wss:;
// White-listing based CSP
connect-src 'self' blob: https://gcp.api.snapchat.com
https://web-frontend-dot-sc-analytics.appspot.com
https://aws.api.snapchat.com
https://.google-analytics.com ...
```

References

[1] 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," May 2016.
[2] M. Steffens, M. Musch, M. Johns, and B. Stock, "Who's Hosting the Block Party? Studying Third-Party Blockage of CSP and SRI," May 2020.
[3] Google Security Team, "CSP Evaluator," https://csp-evaluator.withgoogle.com/
[4] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. 2019. "Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation," https://doi.org/10.14722/ndss.2019.23386