Semantically Sound Analysis of Content Security Policies

Stefano Calzavara, Alvise Rabitti, Michele Bugliesi

Semantically Sound Analysis of Content Security Policies

Stefano Calzavara[1], Alvise Rabitti[1], and Michele Bugliesi[1]

Università Ca' Foscari Venezia
name.surname@unive.it

Abstract. Content Security Policy (CSP) is a W3C standard designed to prevent and mitigate the impact of content injection vulnerabilities on websites. CSP is supported by all major web browsers and routinely used by thousands of web developers in the world to improve the security of their web applications. In this paper we review our formalization of a core fragment of CSP, which we fruitfully employed to reason on the security import of flawed CSP implementations and deployments, as well as to perform a longitudinal analysis of how existing policies are evolving as the result of maintenance operations.

Keywords: Content Security Policy · Formal methods · Web security

1 Introduction

Content injection is arguably one of the most severe threats on the Web. In a content injection attack, a malicious user manages to craft an attack payload, typically a script, which gets injected into a benign web application and becomes indistinguishable from legitimate web contents, thus inheriting their privileges. This way, the attack payload can steal confidential information from the web application or redress the user interface to fool the victim into unknowingly performing security-sensitive operations.

Content injection can be prevented by means of safe coding practices [4], yet it is now widely acknowledged that this is difficult in practice and thus security practitioners rely on a *defense-in-depth* approach against content injection, where protection is achieved by implementing mitigation at several different layers. One such layer is Content Security Policy (CSP), which is now supported by all major web browsers and routinely used by thousands of web developers in the world to improve the security of their web applications. CSP has undergone several authoritative studies as of now [6, 5, 2], with our article *Semantics-Based Analysis of Content Security Policy Deployment* being the latest research work on the topic [3]. The distinctive trait of our approach with respect to previous studies is the use of *formal methods* techniques to tackle a rigorous investigation of CSP. Specifically, we defined the *denotational semantics* of a significant fragment of CSP, called CoreCSP, which we fruitfully employed to reason on the security import of flawed CSP implementations and deployments, as well as to perform a longitudinal analysis of how existing policies are evolving as the result of maintenance operations.

2 Background on Content Security Policy

A content security policy is a list of *directives*, restricting content inclusion for web pages by means of a white-listing mechanism. Directives bind content types to lists of sources from which a CSP-protected web page is allowed to include resources of that type. For instance, the directive `img-src https://a.com` specifies that the web page is allowed to load images just from the host `a.com` using the HTTPS protocol. CSP is a client-server defense mechanism: content security policies are sent from the server to the browser by means of HTTP headers or `<meta>` elements in HTML pages, while their enforcement is performed at the browser side on a per-page basis. If a content security policy does not include an explicit directive for a given content type, the `default-src` directive is applied as a fallback. Allowed sources for content inclusion are defined using *source expressions*, a sort of regular expressions used to express sets of web origins in a convenient way. Content inclusion from a given URL is only allowed if the URL matches any of the source expressions specified for the appropriate content type.

To exemplify how CSP works, we show a very simple policy below:

```
script-src https://www.unive.it;
img-src https://*.unive.it;
default-src https://*
```

The policy above states that scripts can only be loaded from `www.unive.it`, images can be loaded from any sub-domain of `unive.it` and all the other contents, e.g., style-sheets, can be loaded from everywhere; in all cases, the HTTPS protocol must be used. Moreover, the policy automatically disables the execution of inline scripts, which are the most dangerous threats for content injection. This default restriction can be voided by including the `'unsafe-inline'` source expression in the `script-src` directive.

3 Research Summary

We used our CoreCSP model of the CSP semantics to:

1. reveal a *wrong implementation* of the CSP specification in Microsoft Edge, which was deemed dangerous and fixed after our report (CVE-2017-11863). Moreover, we identified a *subtle quirk* in all browser implementations, which deserved a careful security analysis using our semantics to be shown secure;
2. automatically analyze the security of existing content security policies against script injection. Our analysis showed that *91.6% of the existing policies are trivially bypassable and provide no protection at all*;
3. automatically track which changes to deployed content security policies have been performed in the name of security (more restrictive policies), to preserve compatibility (more permissive policies) and for maintenance reasons (unrelated policies). Our analysis showed that *less than 3% of policies changes were done to improve security*, which confirms that CSP is failing as a security mechanism against content injection.

In the next section, we provide an overview of our technical treatment.

4 Technical Overview

4.1 Syntax and Semantics of CoreCSP

We let *str* range over a denumerable set of strings. The syntax of policies is shown in Table 1, where we use dots (...) to denote additional omitted elements of a syntactic category. A policy p is either a single list of directives $d_1, \ldots, d_n$ or the conjunction of two policies $p_1 + p_2$, which requires both p_1 and p_2 to be enforced. Directives, in turn, bind content types t to directive values v; their syntax includes a default directive, applied to all the contents not restricted by other directives. Directive values are sets of source expressions $\{se_1, \ldots, se_n\}$.

Content types	t	$::= \texttt{script} \mid \texttt{img} \mid \ldots$	$(t \neq \texttt{default})$
Schemes	sc	$::= \texttt{http} \mid \texttt{https} \mid \ldots$	
Policies	p	$::= d_1, \ldots, d_n \mid p + p$	$(n \in \mathbb{N})$
Directives	d	$::= t\texttt{-src}\ v \mid \texttt{default-src}\ v$	
Directive values	v	$::= \{se_1, \ldots, se_n\}$	$(n \in \mathbb{N})$
Source expressions	se	$::= h \mid \texttt{unsafe-inline}$	
Hosts	h	$::= sc \mid he \mid (sc, he)$	$(sc \neq \texttt{inl})$
Host expressions	he	$::= * \mid *.str \mid str$	

Table 1. Syntax of CoreCSP (excerpt)

The formal semantics of CoreCSP is defined on top of three main entities: *locations* l are uniquely identifiable sources of contents, e.g., URLs; *subjects* s are HTTP(S) web pages enforcing the content security policy; and *objects* o are contents available to subjects for inclusion, e.g., images hosted at a given URL. The semantics follows the denotational style and is based on judgements like:

$$p \vdash s \hookleftarrow_t O,$$

reading as: the policy p allows the subject s to include as content of type t the set of objects O. It is possible to order policies using a subject-indexed binary relation $\leq_s$ such that, for all policies p_1 and p_2, we have $p_1 \leq_s p_2$ if and only if p_1 is no more permissive than p_2 when deployed at s. More formally, this means that $p_1 \vdash s \hookleftarrow_t O_1$ and $p_2 \vdash s \hookleftarrow_t O_2$ imply $O_1 \subseteq O_2$ for all the content types t.

4.2 Applications of CoreCSP

Wrong Implementations of CSP. We empirically observed a few inaccurate implementations of the CSP specification in major browsers by means of a set of test cases we manually crafted. To formally reason on the security import of such cases, we defined policy-to-policy compilations which embed the inaccurate behaviors of the browsers directly in the CoreCSP semantics. For example, we

defined a compilation $|\cdot|$ which removes all the conjunctions (+) from policies, which captures the incorrect CSP implementation provided by Microsoft Edge. It is possible to prove that, for all policies p and subjects s, we have $p \leq_s |p|$, which formally shows that the CSP implementation of Microsoft Edge can only make policies more permissive than intended (and it actually does).

Vulnerability to Script Injection. We defined syntactic conditions on policies which capture their vulnerability to script injection and proved that such conditions are both sound and complete, i.e., they capture all and only the ways script injection might happen when CSP is deployed. We used such conditions to implement an automated security checker for existing content security policies, which we employed to show the insecurity of the very large majority of the policies deployed in the wild (91.6%).

Policy Changes. Since $p_1 \leq_s p_2$ if and only p_1 is no more permissive than p_2 when deployed at s, we can use the $\leq_s$ relation to capture the nature of policy changes in the wild, i.e., to understand whether observed policy changes lead to more restrictive or more permissive policies. We automated such analysis and performed it on a large scale, showing that only a tiny fraction of changes (less than 3%) is intended to improve security by making policies more restrictive.

5 Conclusion

Formal methods hold promise to support a more principled and rigorous analysis of the web platform, as shown by our analysis of the current CSP deployment. We encourage interactions between the formal methods community and the web security community on challenging research problems, which require both theoretical foundations and a practical point of view. We refer the interested readers to an extensive survey on formal methods for web security [1].

References

1. Bugliesi, M., Calzavara, S., Focardi, R.: Formal methods for web security. J. Log. Algebr. Meth. Program. **87**, 110–126 (2017)
2. Calzavara, S., Rabitti, A., Bugliesi, M.: Content Security Problems? Evaluating the effectiveness of Content Security Policy in the wild. In: CCS. pp. 1365–1375 (2016)
3. Calzavara, S., Rabitti, A., Bugliesi, M.: Semantics-based analysis of Content Security Policy deployment. TWEB **12**(2), 10:1–10:36 (2018)
4. OWASP: XSS prevention cheat sheet (2017), `https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet`
5. Weichselbaum, L., Spagnuolo, M., Lekies, S., Janc, A.: CSP is dead, long live CSP! On the insecurity of whitelists and the future of Content Security Policy. In: CCS. pp. 1376–1387 (2016)
6. Weissbacher, M., Lauinger, T., Robertson, W.K.: Why is CSP failing? Trends and challenges in CSP adoption. In: RAID. pp. 212–233 (2014)