Poster: Are Trusted-Types the Panacea for XSS?

***Abstract*—Cross-Site Scripting (XSS) is one of the most prevalent vulnerabilities present in modern Web applications. To mitigate the effect of this markup injection vulnerability, the Content Security Policy (CSP) has been introduced. However, research has shown that CSP fails its task. Developers face a plethora of roadblocks during the deployment process, and third parties often mandate the usage of lax and trivially bypassable CSPs. Especially dangerous JavaScript sinks such as *eval*, and HTML creating sinks like *innerHTML* are often used by third-party code. In order to mitigate the threat that is introduced by the usage of those APIs, Trusted Types can now restrict and sanitize the input for those functions. So, the question arises: is Trusted Types the panacea to rid the Web of client-side XSS? To answer that question, we want to conduct a semi-structured interview that includes a coding task where the participants need to create a Trusted Types Policy for a small Web application.**

I. Introduction

As a cornerstone of our modern society, the Web has improved the way we communicate, collaborate, teach, and entertain ourselves and our fellow human beings. Due to this importance, the Web is one of the primary targets of attacks. For more than two decades, Cross-Site Scripting (XSS) has been present in the annually published OWASP Top 10 Web Application Security Risks [10]. In order to mitigate the effect of those attacks, a correctly crafted Content Security Policy (CSP) [20] can be deployed. However, numerous research has shown that the vast majority of all policies in the wild are trivially bypassable [1, 12, 18, 19]. Furthermore, recent work uncovered problems and roadblocks that developers face when dealing with CSP deployment [13], where third-party code was identified as one of the major roadblocks for CSP. They also mentioned that server-side XSS seems to be far more prominent in the minds of developers than the client-side because all participants explained this type of XSS during the drawing task. Also, even CSPs that are considered meaningful can be bypassed via JSONP [18], script-gadgets [8], open redirects [11], or the usage of string-to-code functions like JavaScript's *eval* function. Third parties, however, often require the usage of the *eval* function, or even the usage of *unsafe-inline*, in order to work properly [15]. Allowing dangerous functions like *eval* and other string-to-code conversions, or allowing the usage of event handlers and inline scripts via functions that create DOM Elements (e.g. `innerHTML`) enable client-side XSS attacks. Because research has shown how prevalent those client-side XSS attacks are in the wild[7, 14], Trusted Types have been proposed [6] as a mechanism to sanitize data before passing it to dangerous JavaScript sinks. While Wang et al. [17] have exemplified how Trusted Types could be incorporated into a Web Framework, nobody checked if Trusted Types suffers from the same usability issues as its "older" brother CSP. Especially in a realistic setting that the Web application uses third parties for features, analytics, and monetarization.

Notably, Trusted Types is special since a developer has to face certain technical challenges that occur during Trusted Types deployment but not during the deployment of CSP. For example, same-origin iframes inherit the parent's CSP (which is used to enforce Trusted Types), but they do not inherit the Trusted-Types Policy itself. Also, third parties often programmatically add random URLs (e.g., changing advertisement banners) or evaluate random JavaScript code. Furthermore, Steffens et al. [15] have shown the common practice of using inline scripts and inline event handlers in programmatically added HTML tags of third parties. To specifically allow those, a developer needs to parse the HTML and check if the contained JavaScript is allowed to be executed. However, the usage of the browser's DOMParser API is restricted to only receive a Trusted Type, which results in a circular dependency.

In order to assess if Trusted Types is repeating the problems and design errors of CSP, we want to answer the following research questions by conducting a semi-structured interview that includes a coding task:

1) Do developers know client-side XSS and understand the difference to the server-side problem?
2) What roadblock do the developer face when deploying Trusted Types for a Web application?
3) How do developers resolve problems during the deployment, and what strategies do they follow?

II. Methodology

In order to enable participation from all over the world and to reduce the concerns of possible participants regarding the pandemic situation, we want to conduct the study as a fully online experience. To make this process as comfortable for the participants as possible, we will use whatever video chat software they prefer as long as it allows for screen sharing.

Given that Trusted Types is a relatively new security mechanism (only Chromium supports it, and only since version 83 (May 2020)), not many developers have worked or even heard of Trusted Types before. Thus we want to invite real-world Web developers without prior experience with the mechanism and give them a brief introduction into Trusted Types.

After the introduction into the mechanism, we want to conduct a semi-structured interview with the participants in order to assess their knowledge and perception of XSS and its dimensions (client- vs. server-side), as well as mitigation techniques for those vulnerabilities.

Afterward, we ask the participants to create a functional Trusted Types policy for a small Web application. In contrast to the application used by Roth et al. [13], we want to incorporate real-world third-party services to increase the ecological validity of our results. In order to save time, we give the participants a short introduction to the application structure/code and where to deploy the Trusted Types policy. Details of the Web application, as well as details on the online setup for the coding task, will be explained later (Section II-A).

Finally, during the debriefing, we assess if the participant plans to deploy Trusted Types in their work setting and, if not, what would need to change in order to do so.

To analyze the gathered information, we will first transcribe each interview. Afterward, we unitize [2] the transcript and conducted open coding according to Strauss and Corbin [16] to analyze the data. For the analysis of the coding task, we additionally use the screen recordings to not miss any information. In total, two coders will be involved in the coding process and construction of the codebook. We will iteratively continue with this procedure until we conducted enough interviews to reach saturation of our dataset.

A. The Coding Task:

The application itself will be a Python Django Web application that allows the user to store private notes after a successful registration/login. Notably, it is not required to understand how details of the framework to deploy Trusted Types successfully.

This application uses third-party services that are also used in real-world Web applications such as analytics (Google Analytics [9]), social media integration (Twitter Widget [5]), maps integration (Open Street Maps [4]), comment plug-ins (Disqus [3]), and an ad service (self-crafted to prevent ad-fraud). We choose those services in order to account for widely used third parties from a wide variety of different vendors.

To make participation as comfortable as possible, we offer them several ways to change the code during the coding task. We offer them to download the application's source code to simulate the usual programming behavior as closely as possible. To not force them to run it on their system directly, we offer them to either use an Ubuntu VM that we provide to them, or they can use docker to run the application. If all fail, we also plan to offer the participant to remote control a machine from our institution (e.g., via Teamviewer).

B. Hireing Process:

Roth et al. [13] presented their lessons learned from conducting an online interview study with a coding task. According to their investigations, the most successful way of hiring was to participate as a speaker in non-scientific conferences (e.g., OWASP, BlackHat, RuhrSec) in order to get first-hand contact with real-world Web developers that are interested in Web security. We take those learned lessons and try to do exactly what was proposed by the authors.

III. Conclusion

Trusted Types is a new API that enables Web developers to ensure that only trusted and sanitized input flows into dangerous JavaScript sinks such that effects of client-side XSS are mitigated. The standard is still only an editor's draft, although it is already supported by Google's Chromium Browser and its derivates. From a technical point of view, Trusted Types seems to repeat the problematic decisions of CSP that hindered the wide adoption of the mechanism.

With this work, we want to uncover problems, roadblocks, and deployment strategies of Trusted Types such that we can improve Trusted Types. With an improved and easy-to-use XSS protection mechanism, we can contribute to a more secure Web for every site and every user, not only for the big players.

References

[1] S. Calzavara, A. Rabitti, and M. Bugliesi. Content Security Problems? Evaluating the effectiveness of Content Security Policy in the wild. In *CCS*, 2016.

[2] J. L. Campbell, C. Quincy, J. Osserman, and O. K. Pedersen. Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement. *Sociological methods & research*, 2013.

[3] Disqus. Disqus. Online at https://disqus.com/, 2022.

[4] OpenStreetMap Foundation. OSM Embeddable HTML. Online at https://wiki.openstreetmap.org/wiki/Export#Embeddable_HTML, 2022.

[5] Twitter Inc. What would you like to embed? Online at https://publish.twitter.com/, 2022.

[6] K. Kotowicz and M. West. Trusted Types. *W3C Standard. Online at https://w3c.github.io/webappsec-trusted-types/dist/spec/*, 2021.

[7] S. Lekies, B. Stock, and M. Johns. 25 Million flows later: Large-scale detection of DOM-based XSS. In *CCS*, 2013.

[8] S. Lekies, K. Kotowicz, S. Groß, E. A. Vela Nava, and M. Johns. Code-Reuse attacks for the Web: Breaking Cross-Site Scripting mitigations via Script-Gadgets. In *CCS*, 2017.

[9] Google LLC. Google Analytics. Online at https://analytics.google.com/, 2022.

[10] Open Web Application Security Project. OWASP Top 10 Web Application Security Risks. Online at https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS), 2017.

[11] S. Roth, M Backes, and B Stock. Assessing the Impact of Script Gadgets on CSP at scale. In *AsiaCCS*, 2020.

[12] S. Roth, T. Barron, S. Calzavara, N. Nikiforakis, and B. Stock. Complex Security Policy? A Longitudinal analysis of deployed Content Security Policies. In *NDSS*, 2020.

[13] S. Roth, L. Gröber, M. Backes, K. Krombholz, and B. Stock. 12 Angry Developers - A Qualitative Study on Developers' Struggles with CSP. In *CCS*, 2021.

[14] M. Steffens, C. Rossow, M. Johns, and B. Stock. Don't Trust The Locals: Investigating the Prevalence of Persistent Client-Side Cross-Site Scripting in the Wild. In *NDSS*, 2019.

[15] M. Steffens, M. Musch, M. Johns, and B. Stock. Who's hosting the block party? Studying third-party blockage of CSP and SRI. In *NDSS*, 2021.

[16] A. Strauss and J. M. Corbin. *Grounded theory in practice*. Sage, 1997.

[17] P. Wang, B. A. Gudmundsson, and K. Kotowicz. Adopting trusted types in production web frameworks to prevent dom-based cross-site scripting: A case study. In *EuroS&PW*, 2021.

[18] 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 *CCS*, 2016.

[19] M. Weissbacher, T. Lauinger, and W. Robertson. Why is CSP failing? Trends and challenges in CSP adoption. In *RAID*, 2014.

[20] M. West. CSP Level 3. *W3C Standard. Online at https://www.w3.org/TR/CSP3/*, 2021.