Same-Origin Policy definition variations across different problem contexts

Albertus Mitton

Amit509

6547277

INTRODUCTION

"The *Same-Origin Policy* is perhaps the most important security mechanism for protecting web applications," [Schwenk et al, 2017]. "…if the process-based isolation disregards the same-origin policy as one of its goals, then its promise of maintaining the 'web/local system' separation is doubtful," [Jia et al, 2016]. The above quotes say, perhaps, everything that needs to be said about the importance that the same-origin policy has to browser security.

Despite being so important, there is still no formal definition of SOP [Schwenk et al, 2017]. Instead, SOP is defined and applied differently, depending on the context that is being considered.

The three papers that I will be discussing, which are summarised in the section below, target very different areas of concern, where SOP plays an important role. [Schwenk et al, 2017] look at SOP-DOM access control, or the access that scripts from different domains have to the host document's document object model. [Jia et al, 2016] propose an attack, that bypasses Google Chrome's SOP enforcement, to allow access to the user's local system through cloud-services. [Jackson et al, 2006] is an older paper that looks at how improper SOP enforcement for browser state information, can lead to websites from multiple domains being able to track users across sites.

In this paper I will give brief summaries of the papers discussed, and then compare the way that the above authors define SOP, to solve their respective problems.

RELEVANT PAPER SUMMARIES

Protecting browser state from web privacy attacks [Jackson et al, 2006]:

Compared to the other papers I will discuss; this paper takes a different angle on the possible attacks that can take place if "same-origin browsing" is not properly enforced. The area of interest in this paper is browser state information, specifically protecting stored state information from websites that did not store the information originally.

They consider 6 different kinds of tracking that can take place using browser state information, single-session tracking, multiple session tracking, cooperative tracking, semi-cooperative single-site tracking, semi-cooperative multiple-site tracking, and non-cooperative tracking. They claim that the default browser configurations of most modern browsers allow for all the above tracking types to take place.

The authors define a generalised same-origin policy, that applies to browser state information. Their "same-origin principle" is defined as follows: "Only the site that stores some information in the browser may later read or modify that information." This definition is very broad, and the authors apply it to different features of the browser, as separate policies, which they define progressively. The authors propose two browser extensions for Firefox, that applies same-origin policies to limit cache-based tracking and visited link tracking, but these extensions do not help against cooperative tracking.

The Web/Local Boundary Is Fuzzy: A Security Study of Chrome's Process-based Sandboxing [Jia et al, 2016]:

In this paper, the authors found some problems with the security specifications of Google Chrome. Chrome has a process-based sandboxing method, where each tab, or web-instance, is treated as a separate process, and cannot simply share resources with another tab, unless they use procedural calls. The goal of this process-based sandboxing, is to isolate the "local system

from the web." The authors noted, however, that the Chrome team had declared same-origin isolation as one of their "out-of-scope" goals. Chrome had some rudimentary same-origin policy enforcement but, as the authors show, it is easy to circumvent and exploit.

The same-origin policy in Chrome is enforced within the renderer processes themselves, which are the processes that are in control of reading and displaying the http files on the screen, as well as the processes representing the web-instances that the user has open currently. Not dedicating a separate process to enforce same-origin policy improves efficiency, but makes the renderer processes vulnerable to cross-domain attacks.

The authors give an example of how they were able to exploit this security flaw, by using cloud-services (in this case Google Drive) as an intermediary link between the "web," and the "local system." They were only able to perform this attack due to the weak same-origin policy enforcement that the Chrome team had implemented.

They propose certain "lightweight" countermeasures to this attack, which will prevent the authors' exploit from being used if added to the browser's renderer processes. They do say that the Chrome team has admitted that, in order to fully protect against cross domain attacks, a complete refactoring of Chrome would need to be done.

Same-Origin Policy: Evaluation in Modern Browsers [Schwenk et al, 2017]:

This paper focuses on how "SOP-DOM [same-origin policy for document object model] access is implemented in modern browsers". The target area of this paper is different from the two papers above, focusing on the access that scripts from different "execution contexts" have, to document object model elements on the current "execution context." The authors focus on designing a testing framework for SOP-DOM, and they have an in-depth look at the type of access-control that might be best suited for SOP-DOM. They conclude that access rights granted by SOP-DOM depend on web origins, "the type of embedding element, the sandbox, and CORS attributes."

THE PROBLEMS

We can see that SOP applies to many different areas of web security. In this section I will describe the problems that the authors attempted to solve in their respective papers.

[Jackson et al, 2006] look at browser state information, which is stored client-side, at the request of some web origin. It is in the user's interest for this sort of information to be stored locally, because users want speed and efficiency when browsing the web [Jackson et al, 2006]. The problem is that the sort of information that is stored, can be critical in terms of privacy and security. If an attacker gains access to this information, he could track a user across web-origins, observing the user's activities, and possibly steal sensitive information. Cookies and cache information from multiple web-origins are all stored together in the browser, and the authors show that sites that did not store this information have some read (and in some cases, write) access to it.

[Jia et al, 2016] consider the problem of enforcing SOP in the same process as the document being viewed, containing resources from multiple origins. If this method of SOP enforcement can be exploited, the entire process would be compromised, allowing multiple origins to share all the permissions that the current process has. Another aspect of the authors' problem is how cloud-services can be exploited to access and modify the local system. This attack is only possible if SOP can be bypassed in the browser though.

[Schwenk et al, 2017] look at the access rights that scripts from different origins have to the objects open in the host document.

SOP DEFINITIONS

I will now look at the different ways that the authors have defined SOP.

[Jackson et al, 2006] define a generalised version of SOP that they call the "Same-Origin Principle,": "Only the site that stores some information in the browser may later read of modify that information." They call it generalised, because the exact way that SOP is enforced would

depend heavily on the context, or the feature of the web being considered. So, they go on to consider same-origin *policies*, that keep the same general idea of isolating different origins from having access to each other's resources, but allowing context-specific changes to be made. Examples of how the principle would be applied differently, is in, for example, caching vs visited link tracking.

When writing cache entries, two entities could be involved, the site embedding the content, and the host of the content. Same-origin policy should be applied, but if we restrict read access from the site that embeds the content, if we visit that site in the future, it would not be able to access the cached information, making the cached information useless. The host of the content would have full access to the cached information, but if the content was, for example, an image on a news site, the user would not know he has to access the original host of the image in order for the cached information to be read. A concession therefore has to be made in the same-origin principle. On storing the cache entry, the site embedding the content observes that something has been cached, and can access that same entry in the future, if the same content is needed by the user from the same site. If a different site wants to store the same image from the same hosting site, it would not have read access to the cache entry already containing the resource, because that site did not observe the cache entry being stored. This would mean some cache "hits," "miss," but no information would be leaked between sites.

Visited hyperlinks can be a privacy risk if the wrong sites can observe which URLs a user has visited on the current webpage. Two sites should have access to this information, the site where the link is clicked, and the site that the user visits as a result of the click. So, either of these sites should have access to that information if the browser state is persistent. The authors propose two conditions that need to be met for a link to be coloured as "visited," the site of the page where the link is found should be allowed to maintain persistent state, and one of the following should be true: the two webpages should be on the same site, or the user previously visited the second site from a page on the first site.

[Jia et al, 2016] loosely define the same-origin policy to be the restriction of not allowing access from website *x.com* to website *y.com*, unless cross-domain functional calls are used.

Throughout their paper it is also clear that they consider “origin isolation” to be a synonym for SOP. By “origin isolation” they mean the process of sandboxing the resources that one origin has access to from other origins. They define origin to be the combination of protocol, host and port.

[Schwenk et al, 2017] define SOP to be a “complex set of rules.” These rules regulate:

- DOM access – as described in the summary and problem sections above,
- local storage and session storage separation – which locally stored objects can be accessed by JavaScript code in webpages,
- XMLHttpRequest – restrictions on cross-origin HTTP access,
- Pseudo-protocols – SOP enforcement concerning *about:,javascript:, and data: protocols*,
- Plugins – Java, Silverlight, PDF, Flash variants of SOP enforcement,
- Windows/Tabs – cross-window communication, and
- HTTP Cookies – which URLs have access to locally stored cookies.

DISCUSSION

We can see that there is a clear general idea of what SOP should mean, and that is to somehow enforce restrictions on resource access across different web origins. [Schwenk et al, 2017] manage to include the considerations of the other two authors in their list of what SOP applies to, [Jackson et al, 2006] would fall loosely into their Http Cookies category – even though there is a lot more to browser state information than just cookies, and [Jia et al, 2016] would fall into the Window/Tab and Local storage and session storage categories, very loosely. [Jackson et al, 2006] even managed to note that SOP enforcement, even solely in the area of browser state information, should be defined separately, on a case to case basis.

Interestingly though, [Jia et al, 2016] kind of definitively state that SOP **is** origin isolation, or the restriction of not allowing access from *y.com* to *x.com,* unless done through the proper channels. This idea of SOP enforcement is very narrow, and does not include browser state information access, or any of the other categories mentioned by [Schwenk et al, 2017]. Their

definition serves the purpose of helping to solve their problem of interest, but is not general enough to go beyond that.

The authors mention the above areas where SOP is critical, in their attack implementation section, but they make no effort to broaden their definition of SOP. Instead they make statements like the following, “We find that these storage APIs strictly follow SOP, which indicates that only the origin itself can access its data stored in the storage.” The authors clearly have a good idea of what they believe SOP is, but by not clearly stating what exactly they mean by SOP, instead assuming some universal knowledge of the term as shown in the quote above, they fail in highlighting the true importance of SOP enforcement. Their use of the concept SOP is actually quite ironic, since their paper is on the Chrome team’s misuse of the terms “web,” and “local.”

An indefinite, unofficial, albeit good definition of SOP, might just be [Schwenk et al, 2017]’s complex set of rules. SOP enforcement is implemented in many areas of modern browsers and computer systems. As the web gets more complex, the way in which SOP is enforced will also, undoubtedly, get more complex. It is therefore very important that we have a clear current definition of the same-origin policy and what it applies to. [Schwenk et al, 2017] make a good attempt, but they only focused on a singular area in-depth. More work needs to be done to have a more universal definition, but if that can be done, future implementations might be a little bit simpler and more secure.

REFERENCES

Jackson, C., Bortz, A., Boneh, D., & Mitchell, J. C. (2006). Protecting browser state from web privacy attacks. *Proceedings of the 15th international conference on World Wide Web* (pp. 737-744). ACM. Retrieved from https://pdfs.semanticscholar.org/f29a/f7a5f83b5c1d8d65a9a3fb6b99512ff38987.pdf

Jia, Y., Chua, Z. L., Hu, H., Chen, S., Saxena, P., & Liang, Z. (2016). The Web/Local Boundary Is Fuzzy: A Security Study of Chrome's Process-based Sandboxing. *Proceedings of the 2016*

ACM SIGSAC Conference on Computer and Communications Security (pp. 791-804). ACM. Retrieved from https://pdfs.semanticscholar.org/ce42/a9218aaa4a3fc5c069277c19213ca52aeb2b.pdf

Schwenk, J., Niemietz, M., & Mainka, C. (2017). Same-Origin Policy: Evaluation in Modern Browsers. In *26th {USENIX} Security Symposium ({USENIX} Security 17)* (pp. 713-727). {USENIX} Association. Retrieved from https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/schwenk