Usefulness and Workarounds for the Browser Same Origin Policy in the Modern Web

Yotam Bentov
Tufts University

December 15, 2015

Abstract

We discuss the state of the browser same origin policy as it is currently exercised across browsers. We discuss the reasoning behind the policy, as well as scenarios where the policy is useful. We then go on to examine some of the potential issues with the same origin policy. Finally, two potentially malicious workarounds are proposed, both of which utilize browser extensions to tamper with the cross-origin-resource-sharing headers of HTTP requests.

1 Introduction

1.1 State of the Origins

The advent of Web 2.0 in the past decade has pushed the early design of the internet far past it’s initial scope [1]. The modern web is composed of rich user experiences that span over multiple website simultaneously. Websites maintain user state both internally, as well as on user computers, in an effort to tailor rich web-app experiences to each individual user. This rich experience then draws on a number of resources that may exists across the web, such as images, links, and other forms of interactive content. There are, however, many potential vulnerabilities that are brought about by rich user experiences. As a base-rule we can say that a website mustn’t interfere with the workings of another website’s data. Additionally, user information
should only be accessible to websites that have asked for that data, or have otherwise chose to store that data with the user.

The same origin policy was created as a means to limit websites from interacting with each other without user consent. The same origin policy isn’t as much a specifically defined set of rules, as much a set of limitations that are implemented similarly across browsers [2]. As such, there are certain differences in implementation that manifest themselves. That said, the same origin policy ensures in general that resources from different ”origins” cannot interact with each other. Origins are then defined by the following three qualities: protocol, URI, and port.

1.2 Same Origin Policy in Detail

First, the same origin policy limits interactions of origins determined by protocol. ”Protocol” refers to the application-layer protocol used to interpret the information. As such the urls: HTTPS://www.site.com and HTTP://www.site.com will be restricted by the same origin policy. Secondly, an origin is defined by a URI (Uniform Resource Identifier). As an example, the origin site-one.com and site-two.com will be considered different. It’s worth noting that URI’s are defined absolutely, so save for a number of exceptions the origin mail.site1.com and site1.com will be considered of different origins. Finally, origins are defined by their port. As a result the origin HTTP://www.origin.com:200 will be considered separate from HTTP://www.origin.com:201. By default, HTTP is served over port 80 while HTTPs is served over port 443. However, different browsers approach explicit port naming differently, where some restrict explicit port naming versus implicit naming [3].

Another area where same origin policy is important is internet cookies. Cookies are simple key-value pair strings stored on a user’s computer by a website. Cookies define origins a bit differently. First of all, cookies are protocol agnostic, so both secure and non-secure connections are capable of accessing the same cookies. However, it is also possible to limit a cookie’s availability using protocol, path, and subdomains.

Despite the outline described above, same origin policy is highly variant. Different browsers implement any number of the above specifications with difference in restriction. Additionally, older, but popular, versions of browsers may also lack some of the more important implementation details [2]. Even more, the same origin policy doesn’t limit all cross-origin com-
munication. For example an image html element can make a GET HTTP request on behalf of the user, unknowing to the user, which is the motivation behind CSRF attacks [4]. Finally browser-bugs may create vulnerabilities that renders the above policies mute since attackers can easily circumvent the limitations [8].

2 Community

Same origin policy matters because browser security matters. Browser security is important, then, since the browser is one of the primary interface with which people connect to the internet. At the time of the writing of this paper, 51% of adults in the U.S. use browser-centric online banking [5]. Additionally, there are roughly 3.2 billion internet users, and browser usage makes a significant part of this usage [6]. It is thus paramount to secure the user-interactions that occur on the browser.

As a motivating example, take a user that is browsing two websites simultaneously in a "tabbed" browser. Imagine one is their facebook page (facebook boasts a user base of 1 billion people %33 of the aforementioned 3.2 billion internet users [7]). The other website could be any malicious stand-in running javascript. In a world without same-origin policy the malicious site could access the user’s Facebook page and start copying data. Now, take the same example and replace Facebook with a banking site, and imagine the malicious site making withdraws on behalf of the user.

The conclusion is that lack of restrictions on cross-origin communication could be a doorway to do anything from stealing user information to creating serious and lasting damage. But, the importance of cross-origin security policies are not understood viscerally enough to lead to major changes. The variance in browsers and existence of major security holes and workarounds are ample evidence that stricker and more uniform restrictions need to be enforced.

3 Workarounds and Issues

3.1 Approach

The remainder of this paper is dedicated to describing the existing flaws and workarounds with the same origin policy. We describe two theoretical
approaches to circumventing the policy in Google Chrome using chrome extensions and a proxy. Although the variance in implementation has been discussed, it will not be explored further. The solution in this case is simple: all browsers need to implement the same explicitly defined policy. Therefore, our goal is to rather showcase workarounds to the commonly implemented policy.

The main approach explored in this paper uses the `Access-Control-Allow-Origin` headers in order to permit HTTP requests that would have otherwise been restricted by the same origin policy. This HTTP header, commonly referred to as CORS header (cross-origin resource sharing header) is used by browsers to permit cross-origin communication through HTTP requests. The key insight is that if an attacker could maliciously install any of the described methods on a target’s computer than the target’s browser would become exposed to a host of potential attacks that utilize cross-origin communication.

### 3.2 Incoming Traffic Method

The first approach, dubbed "incoming traffic approach" is made possible by the lax security restrictions imposed by Google Chrome’s extension api. The extension api allows chrome extensions to monitor all traffic that goes both in and out of all webpages in chrome. It is also noteworthy that these extensions are given implicit permissions to monitor and modify all requests post-download. If an extension is installed on the browser, it can can achieve the objectives we describe.
figure 1. Visual representation of incoming traffic cross-origin resource sharing approach.

The fundamental approach, illustrated in figure 1, is that all HTTP requests would remain untouched until the response arrives. Then, the extension would catch the request and modify it by adding the CORS header. The browser would then interpret this request as permitting cross-origin communication.

The extension requires few permissions, and could be easily packaged as part of a friendlier chrome extension. Even more, the extension exists in the background so it wouldn’t produce any discernible output. The implications of this is that any website the user uses, malicious or not, could then communicate with every other website.

3.3 Outgoing Traffic Method

The second approach is a bit more involved but could be far more destructive. This approach, dubbed “outgoing traffic method” utilizes two key tools. First, this approach involves a chrome extension that catches all outgoing requests and redirects them to a given proxy. This proxy then completes the requests on behalf of the user and returns the result. To circumvent the same origin policy the proxy adds the CORS headers to the response and sends it to the user.
This approach is a bit more complicated since it requires two fundamental end-points through which the traffic is routed. However, the potential here is to not remove the same-origin policy but to also sniff the responses to intelligently decide when to remove the same origin policy, and when to execute malicious requests. For example imagine the proxy detects that someone has logged into their bank account – it could then serve the banking information with CORS headers, while also tampering with the content of another site to make requests to the banking site. All requests would be made by the user, which could make the proxy essentially invisible. The proxy could then cause damage by just selectively adding CORS headers when appropriate.

3.4 Implementation Details and Available Tools

Both approaches are implemented as chrome extensions. The implementation doesn’t function as a fully-functioning working tool but rather as a proof of concept for the potential work around. The proxy for the outgoing traffic
approach is implemented using node.js and utilizes the node.js express.js web framework as well as the request library.

3.5 Other Threats

The methods described would be effective, but would require another exploit for installation of the extension. However, there are still other vulnerabilities. One year ago a blog post was published claiming it has found a vulnerability that allowed Same-origin policy bypassing in Android [8]. This vulnerability could potentially expose many android devices to the consequences of no same-origin checks have been described before. The point is that despite the workarounds described, there could more ways to enable cross-origin resource sharing maliciously on target browsers.

4 Conclusion and Looking Ahead

The Same-Origin policy is an inconsistent, albeit important, tool to ensure browser security. There are however, both bugs as well as workarounds that render the policy moot in most cases. Understanding these weaknesses is then the first step to creating a safer browser experience that doesn’t compromise user data.

Recently work has been done to devise a new set of policies that are more uniform, and could create an overall higher standard for browser content security [9]. Regardless, it’s important to reckon the current state of things in order to work towards improvement.

References


