Tufts University

COMP116-Introduction to Computer Security

Persistent JavaScript poisoning in web browser's cache

Mentor: Ming Chow

Author: Mengtian Li(Astro)

Mengtian.li@tufts.edu

Dec.12th 2015

Table of Content:

Abstract

Cache is a widely applied component in computer architecture design. By storing some data to cache in advance, one can reduce the serve time when that data is requested in the future. In modern web browser specifically, web cache or HTTP cache refers to temporary storage of web documents to improve performance for both client side and server side. In most of the popular websites, static resources are cached so that user will experience less waiting time per request and server will have less chance to be overwhelmed. However, to this end, the HTTP protocol uses expiration and validation mechanisms to ensure that stale content is updated when necessary. Worse, current browsers lack proper cross-domain cache authentication. [1]

In this paper, I will first explain how web cache works. Then I will discuss the impact of this issue to our community and a recent incident in cache poisoning happened in China in 2014. This paper also discusses in general how how JavaScript poisoning in cache works and how to defense against it from both user's perspective and developer's perspective.

1. Introduction

1.1 Cache

The purpose of the web browser cache is to reduce the loading time of web pages and resources. In order to adapt to the trend of faster web application, cache is a crucial component to accomplish the expectation for responsiveness. Modern web browsers use both memory cache and disk cache to store resources such as HTML, pages, JavaScript, CSS, PDF. [2] There are two types of cache that are widely applied in the industry: web cache and HTML5 AppCache. While web cache is shared across all sites and is active for all web resources, HTML5 AppCache is dedicated per site and is a HTML5 feature that require activation by web application. In this paper, we will mainly focus on web cache.

1.2 Browser Cache Poisoning

While cache can hugely improve user experience and relieve server load, cache itself has many vulnerabilities including timing attacks on browser cache to sniff users' browsing histories and steal private information, [3] geolocations leakage [4] etc. Among these vulnerabilities, browser cache poisoning is the most dangerous one and can lead to crucial important information leak (credit card numbers, for example) about the victims. By man-in-the-middle (MITM) attacks, attackers can

change original web application resources with malicious JavaScript. With proper cache headers settings, the malicious JavaScript can be persistent in the browser's cache. Even worse, these resources in web cache are shared across different sites that the browser visited, affecting future browser sessions.

Consider the scenario where Adam opening a page from site A. If an attacker has a man-in-the-middle position, he or she can prepend a malicious payload to the cacheable response. As long as the header of the cache is valid and the cache is not cleared, the malicious payload will persist in Adam's browser's cache for a long time.

2. To the community

Internet has become accessible to more and more people today. With the versatility of internet, many traditional services have been moved to the internet including online banking, online video streaming, etc. To cope with the large scale of demand in internet requests, computer scientists have come up with many different techniques to improve the user experience of internet services. Cache is one of those genius techniques. By avoiding re-fetching static data (pictures, for example) from remote servers, user can spend significantly short amount of waiting time for a website to be loaded.

However, as a trade off for faster responsiveness, what complements the great performance of web caching is the increasing severity of security risks in browser cache poisoning. An incident happened in China in early 2014 left Chinese netizens unable to access millions of domains. According to theregister.co.uk, two-thirds of China's DNS (Domain Name System) infrastructure was blighted by the incident, which stemmed from a cache poisoning attack. In addition to this incident, with proper methods and JavaScript code, an attacker can gain sensitive information from user by cache poisoning attack. Extensive research has been done in the professional field to exploit these vulnerabilities in browser caching and defense mechanism against such vulnerabilities. Nevertheless, a recent study has shown that the

majority of desktop browsers and popular mobile browsers are affected by browser cache poisoning. And only 1.63% sites of Alexa Top 1,000,000 have proper protection against comprehensive BCP attacks. [5] There are still a lot of things remains to be done.

Also, this paper strives to raise common user's awareness of the enormous security concern behind web cache. Crackers have recognized that the weakest vector in a network system is the human factor. In other words, without the idea of information security in mind, users will suffer from attacks that permitted by themselves in the first place. For example, recent work has measured the click-through rates for SSL warnings, indicating that more than 50% users click through SSL warnings [6] and exposed themselves to the potential cache poisoning attacks. Despite all the defense techniques against browser cache poisoning exploited in the professional world, users can protect themselves via simple actions: avoid using untrusted networks, clear all the cache from time to time.

3. Defenses

3.1 How to persistently poison browser cache

Model

In the threat model, we consider attacker as our adversary. By one-time MITM attack (ARP poisoning, for example), any traffic from victims will be re-route to attacker specified IP address which contains malicious JavaScript cache poisoning code. Once one-time attack is completed, the malicious JavaScript will stay in victims' browser and the attacker no longer need to intercept the traffic again.

There are 3 ways to perform browser cache poisoning: same-origin, cross-origin, extension-assisted. Here, we elaborate on cross-origin attack where when victim visits site A over HTTP, the attacker can intercept the connection and injects subresources such as JavaScript from site B to site A as external JavaScript.

Scenario

Consider the scenario where a victim requests a page from site A which contains a link to some JavaScript from server S which means the victim establish a new connection to S and download the linked script. [1] If the victim connects to an unsecure honey pot network (free wifi hotspot hosted by the attacker) or even an ARP poisoned router, the attacker can prepend a malicious

JavaScript to the original script returned by server S and send it back to the victim. Since normal JavaScript can be part of a widely applied JavaScript library, these scripts tend to be static and server S will return Cache-Control header in the response to store them in the cache. Now the malicious JavaScript payload resides in the victim's cache. After the one-time MITM attack, as long as the cache entry is fresh, the malicious payload will not be updated and can persist for a long time until the victim clears it. From now on, each time the victim visits web pages that contains the same script, the payload will be triggered. An example of the malicious payload is a keystroke logger.

3.2 Defense against browser cache poisoning

User side

As the weakest vector in the network system, users are easily targeted. In order to defend against such attack, awareness of computer security need to be raised. One simple solution is to avoid connecting to unsecure networks that can be potentially corrupted. However, this is not realistic since there are many free wifi provided by airport, coffee shop or even city wise wifi which are meant for faster web browsing. For this type of attack specifically, manually clear web browser cache from time to time is by far the best alternative solution. Major web browsers support various options for users to clear browsing history and cache on their own. (figure 1 serves as an example

in Chrome) In some minor web browsers such as Next, Baidu etc., AppCache is not cleared even after user follows settings to clear cache unless they reinstall the web browser. [5] It is recommended for users to use major web browsers such as Chrome or Firefox.

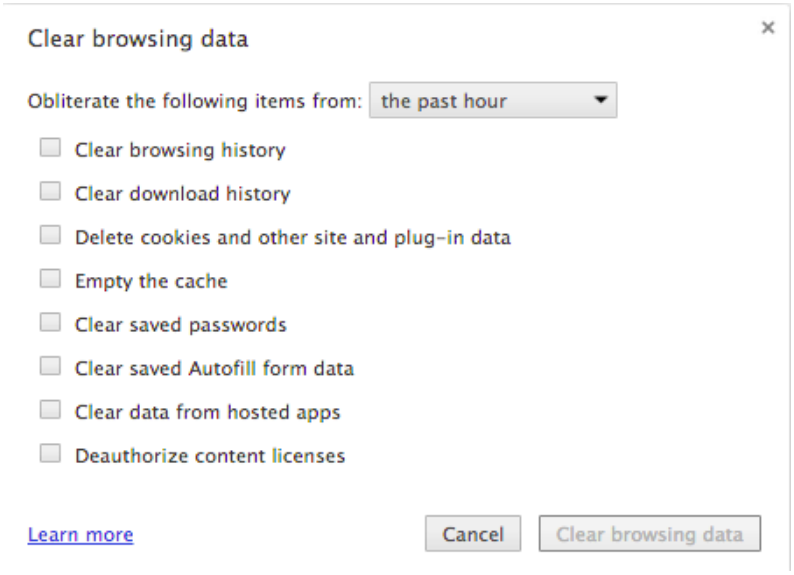

Figure 1 - By changing the tab “the past hour” to “entire”, and clicking on “Empty the cache” (especially after connecting to unsecure network), users can clear all cache and protect themselves from cache poisoning. A little sacrifice in speed will better secure sensitive information.

Developer side

For developers who are developing web browsers, to stay intact from browser cache poisoning, one idea is to prevent HTTPS sites from loading resources

over HTTP by default. Since most sensitive information are in HTTPS session, by avoiding HTTP cache which has much higher possibility of being poisoned, web browser can protect users from browser cache poisoning without user side and server side modification.

For developers who are developing web applications, since web browsers are not all securely implemented, developers need to be aware of the risk in caching content and are responsible for protecting users from cache poisoning. There has been some research in how to mitigate this issue, such as segregating browser cache, [7] randomization of resources' URLs [8] etc. Another solution is to implement a server side integrity check for subresources. In order to prevent malicious JavaScript from running in browser's cache, the server generate randomized hash strings to compare the JavaScript in cache with JavaScript on the server. If the integrity checks failed, the poisoned cache will not be loaded to the target website. Therefore, the browser never loads the poisoned subresources to the site's page and users can remain in a safer state.

4. Conclusion

In this paper we discussed the feasibility and general defense methods for users and developers against persistent browser cache poisoning. The corresponding impact of such attack is gigantic. While the security measures are improving rapidly and more extensive research are undergoing in this field, the increasingly sophisticated counter attacks are still out there being exploited by unknown hacking groups. This is not the end. There are no 100% guarantee in security measurement, but it is important to raise the awareness of this issue not only in the professional field, but also in the commons. As discussed previously, human is the weakest vector in the network system. One simple operation completed on the user side can either destroy all security measurements in vein or can save enormous efforts done by the developers. Thus it is our responsibility as developers to raise their awareness of security and protect them from attacks.

5. Supporting material

A short demo video in how to perform and against persistent JavaScript cache poisoning is available on YouTube. [9]

https://youtu.be/1uQ7EPSlfDs

(There are two corrections under “video detail”)

6. References

[1] Vallentin M, Ben-David Y, Persistent Browser Cache Poisoning, https://www.eecs.berkeley.edu/~yahel/papers/Browser-Cache-Poisoning.Song.Spring10.attack-project.pdf

[2]G. Developers, Leverage browser caching, 2015, https://developers.google.com/speed/docs/insights/LeverageBrowserCaching

[3] Bortz A, Boneh D, Exposing private information by timing web applications, In: Proceedings of the 16th international conference on world web. 2007, p621-8

[4] Wondracek G, Holz T, Kirda E, Kruegel C, A practical attack to deanonymize social network users. In: 2010 IEEE symposium on security and privacy. 2010.p. 223-38

[5] Jia Y, Chen Y, Dong X, Man-in-the-browser-cache: Persisting HTTPS attacks via browser cache poisoning, 2015, http://www.sciencedirect.com/science/article/pii/S0167404815001121

[6] Akhawe D, Felt AP, Alice in warningland: a large-scale field study of browser security warning effectiveness. In:USENIX security symposium, 2013. P. 257-72

[7] Jackson C, Bortz A, Boneh D, Mitchell JC. Protecting browser state from web privacy attacks. In: Proceedings of the 15th international conference on world wide web. 2006, p. 737–44.

[8] Jakobsson M, Stamm S. Invasive browser sniffing and countermeasures. In: Proceedings of the 15th international conference on world wide web. 2006. p. 523–32.

[9] EtherDream, http://github.com/EtherDream/mitm-http-cache-poisoning