Poster: Man-in-the-Browser-Cache: Persisting HTTPS Attacks via Browser Cache Poisoning

Yaoqi Jia*, Yue Chen‡, Xinshu Dong†, Prateek Saxena*, Jian Mao‡, Zhenkai Liang*
*Department of Computer Science, National University of Singapore
‡Beihang University
†Advanced Digital Sciences Center, Singapore

I. Introduction

When browsing the web using HTTPS, if a user Alice ignores, or *clicks through*, the browser's SSL warnings of an invalid SSL certificate, she exposes her browser sessions to a Man-in-the-middle (MITM) attack, allowing attackers to intercept communication in the SSL channel. Recent work has measured the click-through rates for SSL warnings, indicating that more than 50% users click through SSL warnings [1]. A typical solution is to improve warnings of invalid SSL certificates. However, in certain settings, users do not have a choice but to click through the warnings, e.g., when in a hotel WiFi with a malicious proxy that intercepts web sessions.

In this paper, we study the consequence of clicking-through of SSL warnings, focusing on the impact on browser cache. If Alice clicks through one SSL warning, network attackers can launch an MITM attack over a target site protected through HTTPS and substitute original web application resources, such as JavaScript code and images, with malicious resources for the original ones in the targeted site. By setting long-lived cache headers, the malicious copies will be cached in Alice's browser for a long time, and these poisoned resources persistently compromise all future sessions using that browser even over correct HTTPS. We call this class of attacks *browser cache poisoning (BCP)* attacks. Poisoned resources in web cache are shared across all sites visited by the same browser. The implication of these attacks is that if Alice clicks through one SSL warning on any site, all her future sessions on that browser may be compromised until she clears the cache. In our study, we classify BCP attack vectors into three types: *same-origin*, *cross-origin* and *extension-assisted*.

- **Same-Origin.** Suppose the targeted site over HTTPS is an online banking website. If Alice clicks through an SSL warning on the banking site, the attacker can impersonate as the site, and replace the page and the targeted subresources with his malicious ones. By setting long-lived cache headers, the attacker instructs Alice's browser to store the malicious copies for a long time. Alternatively, the attacker can utilize HTML5 AppCache to instruct Alice's browser to store the malicious page, manifest, and subresources in the dedicated storage for the site for one year or longer. Regardless of whether Alice is online or offline, when she revisits the banking site, her browser will directly load the whole page from AppCache without issuing any requests. Since the SSL warning occurs on the banking site, and this attack only affects the same site, we term such attack as *same-origin BCP* attack.
- **Cross-Origin.** In a cross-origin attack, the attacker first injects the banking site's HTTPS subresource into an HTTP response for another site, such as a news site When Alice's browser sends a request for the subresource, the attacker substitutes his malicious resource for the original one. If Alice clicks through the SSL warning, the damage is that the banking site over HTTPS is compromised by the news site over HTTP. Since the SSL warning shown on the news site is for the poisoned subresource, which is a cross-origin resource loaded in the banking site, we term such attack as *cross-origin BCP* attack.
- **Extension-Assisted.** Targets for compromise can be further amplified by browser extensions. Many desktop browser extensions inject resources, e.g., scripts and CSS files, into every page. If Alice clicks through an SSL warning for one extension's injected resource on any site over either HTTP or HTTPS, the attacker can poison it as explained above. The consequence is more devastating than previous two scenarios. Whenever Alice visits any site e.g., the banking site, the poisoned resource will be loaded into each visiting page. Since this attack is based on poisoning the extension's injected scripts, we term it as *extension-assisted BCP* attack.

II. Browser Measurement & Susceptibility of Browsers and Websites

A. Threat Model

The adversary is a one-time MITM attacker against HTTPS, who intercepts HTTPS connections between Alice's browser and the targeted site's server only once. The MITM attacker can utilize a host of well-known MITM techniques (e.g., ARP poisoning and DNS pharming attacks) to re-route all the traffic of Alice to himself. To avoid either suspicion, or subsequently being blocked by additional security mechanisms, once the attacker completes the one-time MITM attack, he no longer intercepts the traffic from/to Alice. We assume that when under a one-time MITM attack, Alice clicks through one SSL warning on a site over either HTTP or HTTPS. In reality, the majority of web users are inclined to click through SSL warnings on various scenarios [1].

B. Preliminary Results

Browsers vary substantially on how they display warnings about invalid HTTPS connections and in their caching policies.

We find several serious vulnerabilities in how browsers handle SSL warnings. For example, we find that CM browser, which has 10 million users, does not check the validity of sites' certificates and never shows SSL warnings. Further, the majority of mobile browsers prompt users with incomplete information in SSL warnings, making it difficult for security-conscious users to make informed decisions. Meanwhile, we find that for click-through HTTPS connections, all 20 evaluated browsers but Safari (desktop version) cache poisoned resources in web cache or in HTML5 AppCache.

We measure BCP attacks on five mainstream desktop browsers and 16 popular mobile browsers. From our experiments, we figure out the inconsistency of SSL warnings in these browsers, e.g., incomplete SSL warnings. Meanwhile, we also find the incoherence of browser caching policies in all evaluated browsers, e.g., caching resources over broken HTTPS in web cache or HTML5 application cache and no pop-up warnings for loading hijacked resources from browser cache.

Susceptibility of Evaluated Browsers. All the evaluated browsers are susceptible to at least one category of BCP attacks. For desktop browsers, Safari is only affected by extension-assisted BCP attacks, Chrome, Firefox and Opera are vulnerable to same-origin and extension-assisted BCP attacks, IE is affected by all the three series of attacks. Since all the evaluated mobile browsers do not support extensions / add-ons, they are not susceptible to extension-assisted BCP attacks. For mobile browsers, Firefox, Chrome, Safari, Opera, IE and UC are only vulnerable to same-origin BCP attacks, and the other browsers are affected by both same-origin and cross-origin attacks.

Susceptible Websites in Alexa Top 1,000,000. We send HTTPS requests to 31,000 SSL-enabled sites in Alexa Top 1 million sites. By analyzing the response headers, we find that 1.63% sites enforce HSTS headers, 1.20% sites set cache-control headers, and only 0.14% sites enable CSP. The majority of HTTPS websites do not have any protection against BCP attacks.

III. Our Defense Techniques

Various existing defenses against HTTPS attacks or attacks via browser cache can help defend against BCP attacks. However, they are not sufficient. CSP [2], Channel IDs [3], SISCA [4], DANE [5], CAA [6], and private browsing mode cannot thwart any series of BCP attacks; HSTS [7] and HPKP [8] can mitigate same-origin BCP attacks; Web Cryptography API [9], Subresource Integrity [10] and randomization of resources' URLs [11] prevent cross-origin BCP attacks; segregating browser cache [12] protects users from cross-origin and extension-assisted BCP attacks. Therefore, none of these techniques provide comprehensive protection against BCP attacks. For instance, HSTS [7] is the successor of ForceHTTPS, which is proposed to mitigate SSL stripping attacks. It provides an HTTP response header for a website to force browsers to make SSL connections mandatory for all subresources on this site. Once HSTS is set in the HTTP header, none of HSTS-compliant browsers give users the option to ignore SSL errors. However, for HSTS, browsers must first connect to the legitimate websites securely to fetch the authorized certificates before connecting to untrusted networks. Thus if the BCP attack occurs before the victim connects to the legitimate site, the attacker can still poison the targeted site's resources. After testing four sites that enable HSTS headers on Firefox, i.e., facebook.com, github.com, paypal.com and alipay.com, we find that the HSTS headers can be stripped by the attacker if it is the user's first visit, and after that the sites are not protected by HSTS.

Guidelines for Users & Browser Vendors. Users should not click through SSL warnings on any site in normal browsing mode. As a precaution, they should also clear browser cache, i.e., web cache and HTML5 AppCache, before visiting a site requesting credentials, especially after an SSL warning is clicked. From the perspective of a browser vendor, to defeat BCP attacks, there are two requirements that suffice: (1) No caching for resources over broken HTTPS in either web cache or AppCache; (2) Default blocking sites over HTTPS from loading HTTP resources.

Acknowledgements

This work is supported in part by Singapore Ministry of Education under NUS Grant No. R-252-000-519-112, the research grant for the Human-Centered Cyber-physical Systems Programme at the Advanced Digital Sciences Center from Singapore's Agency for Science, Technology and Research (A*STAR), and the National Natural Science Foundation of China (No. 61402029), the National Key Basic Research Program (NKBRP) (973 Program) (No. 2012CB315905), Beijing Natural Science Foundation (No. 4132056), and the National Natural Science Foundation of China (No. 61370190).

References

[1] D. Akhawe and A. P. Felt, "Alice in warningland: A large-scale field study of browser security warning effectiveness." in *USENIX Security Symposium*, 2013, pp. 257–272.

[2] "Content security policy," https://w3c.github.io/webappsec/specs/content-security-policy/.

[3] D. Balfanz and R. Hamilton, "Transport layer security (tls) channel ids," http://tools.ietf.org/html/draft-balfanz-tls-channelid-01.

[4] N. Karapanos and S. Capkun, "On the effective prevention of tls man-in-the-middle attacks in web applications," in *USENIX Security Symposium*, 2014.

[5] P. Hoffman and J. Schlyter, "The dns-based authentication of named entities (dane) transport layer security (tls) protocol: Tlsa," RFC 6698, August, Tech. Rep., 2012.

[6] P. Hallam-Baker and R. Stradling, "Dns certification authority authorization (caa) resource record," http://tools.ietf.org/html/rfc6844, 2013.

[7] J. Hodges, C. Jackson, and A. Barth, "Http strict transport security (hsts)," http://tools.ietf.org/html/draft-ietf-websec-strict-transport-sec-04, 2012.

[8] C. Evans and C. Palmer, "Public key pinning extension for http," http://tools.ietf.org/html/draft-ietf-websec-key-pinning-19, 2011.

[9] D. Dahl and R. Sleevi, "Web cryptography api," *W3C Working Draft*, 2013.

[10] F. Braun, D. Akhawe, J. Weinberger, and M. West, "Subresource integrity," *W3C Working Draft*, 2014.

[11] M. Jakobsson and S. Stamm, "Invasive browser sniffing and countermeasures," in *Proceedings of the 15th international conference on World Wide Web*. ACM, 2006, pp. 523–532.

[12] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell, "Protecting browser state from web privacy attacks," in *Proceedings of the 15th international conference on World Wide Web*. ACM, 2006, pp. 737–744.