Cache Timing Attacks Revisited: Efficient and Repeatable Browser History, OS and Network Sniffing

Chetan Bansal, Sören Preibusch, Natasa Milic-Frayling

Cache Timing Attacks revisited: efficient and repeatable browser history, OS and network sniffing

Chetan Bansal, Sören Preibusch, Natasa Milic-Frayling

Microsoft Research

chetanb@microsoft.com

Abstract. Cache Timing Attacks (CTAs) have been shown to leak Web browsing history. Until recently, they were deemed a limited threat to individuals' privacy because of their narrow attack surface and vectors, and a lack of robustness and efficiency. Our attack implementation exploits the Web Worker APIs to parallelise cache probing (300 requests/second) and applies time-outs on cache requests to prevent cache pollution. We demonstrate robust cache attacks at the browser, operating system and Web proxy level. Private browsing sessions, HTTPS and corporate intranets are vulnerable. Through case studies of (1) anti-phishing protection in online banking, (2) Web search using the address bar in browsers, (3) publishing of personal images in social media, and (4) use of desktop search, we show that CTAs can seriously compromise privacy and security of individuals and organisations. Options for protection from CTAs are limited. The lack of effective defence, and the ability to mount attacks without cooperation of other websites, makes the improved CTAs serious contenders for cyber-espionage and a broad consumer and corporate surveillance.

Keywords: Privacy, cache timing attacks, cyber-security, cyber-espionage, browser history sniffing.

1 Introduction

Web caching is a common performance enhancing mechanism. It improves Web browsing speed, for the convenience of the consumer, and reduces the network traffic, thus decreasing the cost for the provider and the user. Caching is not limited to the Web browser but also used by the operating system and network proxies to improve the user experience (Fig. 1). This convenience comes at the cost of decreased user privacy since caches store persistent information that can be interrogated by attacker websites.

Since the early Internet days, browsers have adopted the *same-origin principle* to limit the interaction between websites from different domains and to have cookies and JavaScript's from different sites coexist without interfering [1]. However, this principle is not applied to the entire persistent browser state.

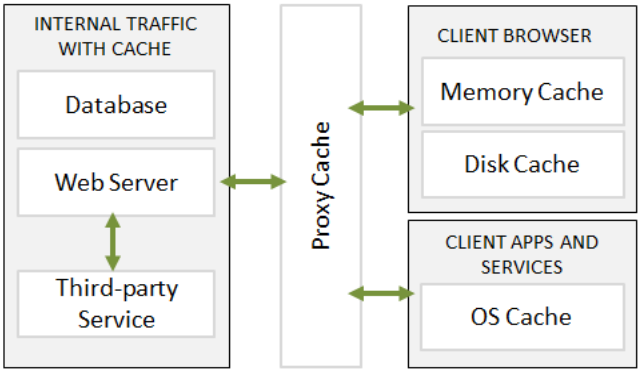

Fig. 1. Interaction among different caching facilities with the user PC and Web network.

The ability for one site to learn about visits of individuals to other sites has fuelled a lucrative Internet economy around advertising services and user tracking across websites [2]. Despite the privacy risks, only rudimentary technical support and policy-based protection have been offered to users to control their exposure. At the same time, browsing history is considered personal information [3], and consumers dislike advertisers using their browsing history even when they are assured to remain anonymous [4].

While cookie-based user tracking is gaining attention of the general public and policy makers, privacy violations due to cache timing attacks (CTAs) are still an obscure matter. Whereas history sniffing through 'coloured hyperlinks' has been fixed by browser manufacturers, CTAs are harder to detect and difficult to prevent [5] [6]. Instead of placing new objects into the cache, CTAs probe the cache for the existence of specific items and make inferences about user activities that leave traces in the cache (e.g., visits to websites). More precisely, the attackers time the browser's response to an item request to determine whether the item was cached (fast response) or fetched from the website (slow response). In the former case, they can conclude that the user has visited the Web resource in the recent past.

Since CTAs require active probing for specific items and careful measurement of the time responses, they have been considered limited in the damage they can cause and the scale they can achieve. A limiting factor in CTAs is the need to lure the victim to the attacker's Web page that hosts the JavaScript for carrying out the attack. Furthermore, a simplistic probing of the cache only allows one-time inquiry since checking for the presence of an item requires reading the corresponding file which is then added to the cache. This side effect contaminates (or 'pollutes') the cache and precludes iterative probing. We revisit the implementation of CTAs and investigate the scope of their effectiveness in the contemporary ecosystem.

New Web paradigms (e.g., HTML5 and AJAX) enable rich, interactive end-user applications. Modern browsers expose comprehensive APIs and position the browser as an 'operating system' that hosts third-party applications (e.g., Chrome OS). At the same time, the scope of Web activities has expanded considerably to include high-

stakes activities such as banking, e-commerce, and e-governance that are attractive targets for attacks. Also, in online social media activities, personal information flows in abundance.

In this paper we demonstrate how CTAs can be implemented to overcome the discussed limitations. We use two specific techniques: (a) time-outs on cache requests and (b) multi-threaded cache probing using Web Worker APIs, to create repeatable and efficient CTAs, making them a serious contender for cyber-espionage and for individual and enterprise-level surveillance by any third party. As a consequence, CTAs become a serious privacy threat both to individuals and enterprises.

Our research makes the following contributions: (1) *extended the scope* of CTAs by including practical attacks on operating systems, browser components, banking and social websites, as well as first real-world, practical attacks on proxy caches that open door for enterprise level tracking and surveillance and (2) improved *efficiency* and *repeatability* of CTAs by eliminating cache 'contamination' during cache probing.

We begin by situating our work within CTA research and follow with the technical description of our approach. We present four case studies that demonstrate the improvements in CTAs. We then discuss remedies to reduce the damage of CTAs before concluding with the summary of our contributions and an outlook on future work.

2 Related work

2.1 Browsing history extraction

The ability of a website to probe whether a hyperlink is displayed as a visited link, through a CSS a:visited rule, was at the heart of the first sniffing attacks on browser history [7]. Fortunately, the vulnerability due to the leak of the visited link colour has now been fixed by all major browser manufacturers, defending from attacks based upon it.

Timing attacks have been well-known for over a decade but remained limited to browser caches and were not considered in detail to achieve robustness and scale. Image-loading times in a browser are a cache-based side channel that can leak information about browsing history. The initial discussion dates back to 2000 by Felten et al. [5] who introduced web timing attacks and showed how they can be used to compromise users' browsing histories. They also discussed timing attacks on DNS and proxy caches.

Considering further the cache timing for DNS and browser, Jackson et al. [6] refined the same-origin policy to also cover the browser cache. They implemented their proposal as a Firefox extension, but the impact on the overall ecosystem remained small. The add-on gained no traction amongst users and browser vendors did not adopt the design [8]. We discuss mitigations in more detail in Section 5.

2.2 Cache timing for inference of private information and identity

More recently, targeted timing attacks have been shown to leak very specific information about the users. Jia et al. demonstrated leakage of users' geo-location through

CTAs [9]. According to them, 62% of the Alexa top 100 websites enable attackers to infer users' geo-location based on cached resources. For instance, websites like Google or Craigslist are personalised by user location (e.g., country or city specific domains and URLs or location-sensitive images). However, due to the lack of empirical studies, user damages from the described leakage of country or city information are unclear. It seems that, unless geo-location attack is combined with other non-trivial attacks, it can only be exploited for targeted phishing attacks. The robustness and accuracy of the attack are questionable, as is the effectiveness with a cache lookup frequency of 5 to 10 URLs per second. In our work, we consider these limitations, as well as the repeatability of the attacks.

Cache timing techniques also misuse the browser cache as a persistent data store, similar to cookies, and thus support identity attacks. Cache cookies can be created by interpreting each cache hit or miss as a single bit [5]. In this manner, a series of URLs can be used to persist several bits of information such as session ids or even unique tracking ids, despite the users' efforts to suppress cookies in their browsers. In the absence of access control for the browser cache, cache fingerprints can be read or written across origins akin to third-party cookies.

3 Attack principles and improvements

3.1 Attack mechanics

Attack principle. Although the same-origin policy [1] restricts cross-origin access to security critical APIs, such as DOM, Cookies, Local Storage, XMLHttpRequest, it allows for shared caches and cross-origin embedding of resources—a fundamental principle of hypermedia. For instance, if the domain A.com embeds a resource A.com/file.ext then the resource is cached by the browser on page load. Subsequently, any other origin can load A.com/file.ext and measure the loading time to infer whether the resource already existed in the cache or not. This can be further used to imply that the targeted user requested A.com/file.ext earlier.

Delivery of the attacks. CTAs can be executed without help from the legitimate site and by any Web-savvy user who is able to implement (or copy) a cache probing JavaScript. Delivery of a malicious JavaScript is not difficult to achieve due to the many ways to distribute URLs, including emails and social media. Advertisements and social networks are the most common channels to distribute JavaScript based malware [10] [11]. Cache probing JavaScripts can be embedded in advertising banners, or delivered through the attacker page by ensuring that it ranks high in search results.

Unlike other JavaScript attacks, like CSR, XSS, etc., timing attacks are based on a polling mechanism. The malicious script has to make a request for each URL and time the response to make an inference. The larger the number of URLs, the longer the script has to run. Thus, cache probing requires a very speed-efficient attack. Otherwise, the victim needs to be exposed to the attack for a prohibitively long time. Finally, previous attacks were limited in their scope; they could only be executed once due to cache contamination with requested URLs.

Attack deployment. Our CTA implementation does not require any custom hardware or software. We have tested the attacks on common browsers, OS, services and real world proxies and banking websites. Our exact setup was a Windows 8.1 PC, on the corporate network of the authors. Attacks are done using JavaScript which can be hosted on any website. The minimum browser requirements for the Web Worker API are Internet Explorer 10, Chrome 4 and Firefox 3.5. Some attacks, like the one described in Section 4.4, require the attacker to host a website within the intranet.

```
function isRequestCached(URI, callback) {
  var timeStamp, timedOut = false;
  var xhr = new XMLHttpRequest();
  xhr.open('GET', URI, true);
  xhr.onreadystatechange = function() {
    setTimeout(function() { callback(!timedOut); }, 2);
  };
  xhr.timeout = 5; // milliseconds
  xhr.ontimeout = function() { timedOut = true; }

  timeStamp = getTimeStamp();
  xhr.send(null);
}
```

Listing 1: JavaScript function snippet for timing queries.

3.2 Improved CTAs

Overall performance. In order to make the timing attacks scalable and avoid cache contamination, we use Web Workers and timeouts on probing requests. Previously, cache contamination was avoided through the same-origin policy and iframes [12]. However, that approach requires DOM access which creates a considerable overhead and leads to poor performance. Furthermore, the DOM is not available to Web Workers due to concurrency issues. As seen in Fig. 2, our parallel implementation of cache attacks is 15 times more efficient than the best-performing approach so far: it takes 0.5 seconds to complete 150 requests, compared to 7.4 sec for the same 150 requests using the CoreDump algorithm [12]. It is also more resilient to cache contamination.

Web Workers. Prior to HTML5, JavaScript did not have any notion of multi-threading. All the scripts on a given web page executed in a single thread. Asynchronous timers using setTimeout and setInterval functions were used to simulate multi-threading. With the introduction of Web Workers, concurrent background tasks can be spawned as OS-level threads. The worker thread can perform tasks without interfering with the user interface. The feature is supported by all major desktop and mobile browsers. Our experimental setup included successful tests with Internet Explorer, Firefox, Chrome, and Opera.

Time-out. We use high resolution timeouts for the XMLHttpRequests to avoid cache contamination during attacks. Listing 1 shows a code snippet where we use very small

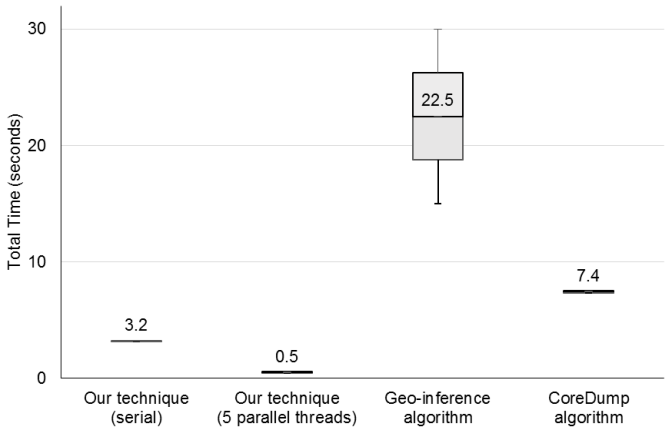

Fig. 2. Total time in seconds to complete 150 requests. Benchmark of our cache probing technique (single-threaded or multi-threaded) against the existing geo-inference algorithm [9] and the CoreDump algorithm [12]. Smaller numbers are better.

timeout values (5ms) to terminate the request before it is completed. This makes sure that the cache does not include the requested URLs due to probing.

The timeout value depends on the type of cache we probe (e.g., a browser or a proxy cache). We extensively tested the timeout technique using Chrome and Opera; similar can be done for other browsers. For attacking the browser cache, the timeout value of 5ms was sufficient while for proxies it was 15ms. The values vary based on the client configuration, bandwidth and network latency. The attacker can pre-compute these values for a specific victim by running benchmarks. We apply a simple method; for browser and proxy caches we make 5 requests, each for cached and un-cached resources, and calculate the mean and max timeout values.

Extended attack surfaces. First, we consider scenarios beyond the browsing history attacks and show that sensitive resources like security images, commonly used on banking website, and XML/JSON based API calls can be targeted to leak user's private information. Second, we exploit Web caches that are not limited to browsers but included in the operating system and network proxies.

4 Case studies

In this section we demonstrate how CTAs render existing anti-phishing defences useless (Section 4.1) and how they can be used to revive and improve previous user deanonymisation attacks (Section 4.2). We also present two novel case studies of CTAs applied to monitoring Web search and OS file-system queries (Sections 4.3 and 4.4).

4.1 Online banking security images

Site-to-user is an anti-phishing product by RSA Security based on the challenge-response model. The product is used by major financial institutions like Virgin Money, HDFC Bank Ltd., and many others. Through Site-to-user a legitimate website can authenticate itself to the user and prove that it is not a phishing site. When a user creates her account, she is asked to choose an image and a text phrase as part of the Site-to-user protection. Now, whenever the user visits the website to log-in, the same Site-to-user image and text are displayed after she enters her customer id and before she enters her password. Thus, the user can verify that the site is legitimate.

The Site-to-user protection is not completely secure. It is vulnerable to man-in-the-middle attacks [13] where a malicious phishing website first asks the user to enter the customer id. Subsequently, serving as a proxy, the attacker makes a request to the real banking website to fetch the image and display it back to the user. Albeit simple to implement, this attack is also easy to detect in real-time because of multiple requests from different clients for the same customer id, in a very short time span.

We describe a new attack against Site-to-user that is difficult to detect by the banking service and equally difficult to prevent. In this instance a phishing site recovers the image directly from the user through a CTA. We confirmed the attack against a real banking website (Fig. 3).

Attack implementation. Since the Site-to-user images are accessible to any individual at the time of subscribing to the service, the attacker has an opportunity to acquire the entire collection of images used by the service. This one-off effort in preparing the resources enables phishing attack across a broad population of customers. Figure 3 illustrates the time (in milliseconds) to get a response from the cache of the selected 283 Site-to-user images.

Responses by manufacturer. RSA Security acknowledged the vulnerability in both the on-premise and hosted versions of RSA Adaptive Authentication. They fixed the vulnerability by setting no-cache headers for the hosted deployment and they also communicated the mitigation to their on-premise customers.

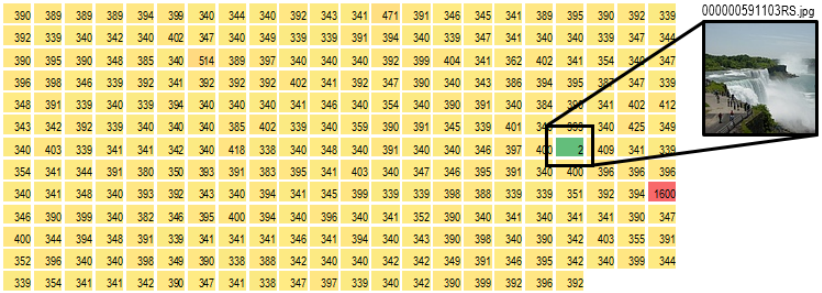

Fig. 3. Request response times in milliseconds for 283 Site-to-user images, colour-coded by magnitude. Green and framed, the correctly identified, cached image (2ms). Red, an outlier that does not impact the success of the attack (1600ms).

4.2 User identification in social networks

Social Networking Sites (SNS) like Facebook, Twitter or LinkedIn are among the most popular destinations on the Web [14]. Facebook has more than 800 million daily active users [15]. SNS store and expose more current and historical personal information about their users and their relations than any other public websites. Personal data on SNS is not limited to personal information, like demographics (age, name, sex, etc.) and profile pictures, but often includes real-time location and status of offline/online friends [16].

Social networking services used in the workplace include data about the organisation, internal team documents, and internal communication. They may also include more personal matter, such as job applications and inquiries of the team members. Linking and exposing such information could be incriminatory. Consequently, security and privacy attacks on these sites could have detrimental effects for their users.

Past SNS attacks involved methods for browser history sniffing based on CSS visited link vulnerability [7], aiming to de-anonymise a person within a set of known users. While the CSS vulnerability has been fixed, alternative attacks resorted to the analysis of the Transport Layer Security (TLS) traffic. Because TLS does not hide the length of the response, user identification can be done based on the pre-computed size of the users' profile images [17].

Attack scenario. Methods for de-anonymising users can be applied to track audiences beyond SNS. For example, a marketing campaign team posts a link to the new promotion on the corporate Facebook page and wants to know who among the company's followers have clicked on the link. Such data is normally concealed from the company by the SNS platform due to privacy reasons. Similarly, in the case of a professional network like LinkedIn, clicks on job adverts could be traced and linked back to employees' profiles. The attack can also be used to circumvent privacy-enhancing techniques built into SNS, such as anonymous groups.

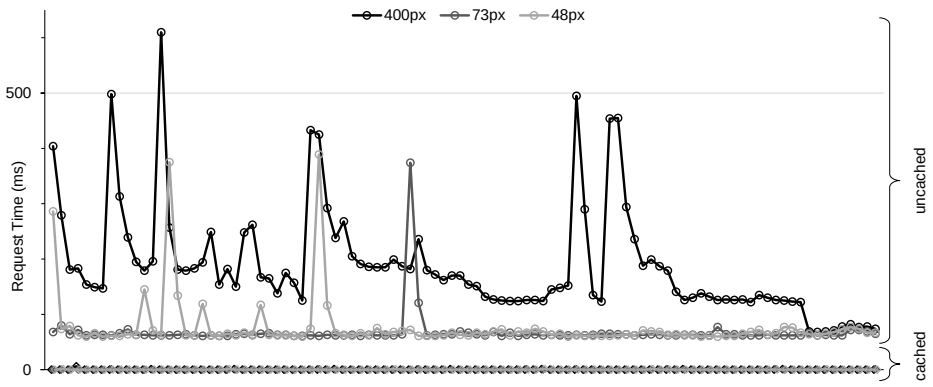

Fig. 4. Request response times in milliseconds for the same Twitter profile picture in the three standard sizes (100 repetitions). For un-cached media, response times vary but are always above response times for cached media, which are all consistently negligible.

Attack implementation. Our attack enables any malicious website to de-anonymise an SNS user by checking if she is within a pre-defined set of users. It exploits the mapping between SNS users and their unique, self-uploaded profile images, a defining feature of SNS [16]. Uploaded pictures are assigned public, stable and unique URLs, cacheable by browsers for long durations. The user's cache is probed for all the images to determine which might be the user's. Figure 4 demonstrates the ability of our attack to consistently measure the difference in response times for cached versus un-cached Twitter profile pictures.

Our attack is easier to execute than previously described de-anonymisation attacks. As noted, history sniffing through differential colouring of visited links has long been fixed and is no longer possible. Furthermore, in contrast to the traffic analysis, our adversary only needs to set up a third-party website; there is no need for network eavesdropping. This increases the effectiveness of the attack because SNS members can be geographically distributed, i.e., not confined to a single physical network.

Mitigation. SNS like Twitter have two options to prevent the attack: first, by making profile pictures un-cacheable; second, by making URLs of profile pictures dependent on the requesting users, so that attackers cannot compile a set of probing URLs.

4.3 Monitoring search queries in real-time

The CTA scenarios covered in the existing literature are limited to targeting image objects. As we noted in Section 3, images are not the only cacheable content type. Other types include XML, JSON, CSV, and HTML. XML and JSON objects are particularly interesting because they typically carry users' personal data in the requested URLs, for example, user identifiers and search keywords.

We demonstrate the threat of XML/JSON caching by constructing attacks that target information entered by the user into the address bar of a Web browser. The implications are serious because the address bars of modern browsers allow users to type in search queries and have them processed directly by a default search engine. Thus, such queries are exposed to malicious websites which can scan the browser cache. As per our tests, these attacks work on most of the commonly used browsers (Internet Explorer, Firefox, and Chrome) and across multiple search providers such as Bing, Yahoo!, Google, and Wikipedia; Ask and AOL do not show the same vulnerability.

Attack scenario. In many instances, browsers are configured to provide real-time suggestions to the users as they type in their queries or Web addresses. Continuous AJAX calls are made to the search provider during this user interaction. For instance, a call made by Chrome to Yahoo! (non-essential parameters omitted):

http://ff.search.yahoo.com/gossip?output=fxjson&command=userquery.

where "userquery" refers to the query typed by the user in the Omnibox. Calls made to Google, Bing and other search engines look similar. The following sequence of AJAX calls are made when the user types in "sec15":

/gossip?output=fxjson&command=s

/gossip?output=fxjson&command=se
/gossip?output=fxjson&command=sec
/gossip?output=fxjson&command=sec1
/gossip?output=fxjson&command=sec15.

The search engine responds to each call with a JSON or XML object that is placed in the browser cache. Because this cache is shared with Web domains, a malicious website can use CTAs to extract the queries using the techniques described in Section 3. Our attack is stronger than the packet size based attacks by Chen et al. [18] in two ways: (1) their adversary model is a network eavesdropper [18], while ours is simply a third party website; (2) their attack is based on the packet sizes of responses [18], while ours is based on cache timing and enables the attacker to leak information at any later time as well. Listing 1 shows our script to carry out the probing.

Mitigation. Ideally, the cache for the address bar, search box and all other browser components should be isolated. Furthermore, one could append a user specific nonce to the requests (Section 5).

Responses by manufacturers. After our report, Bing enabled no-caching headers, effectively preventing CTAs on searches entered into the Browser address bar.

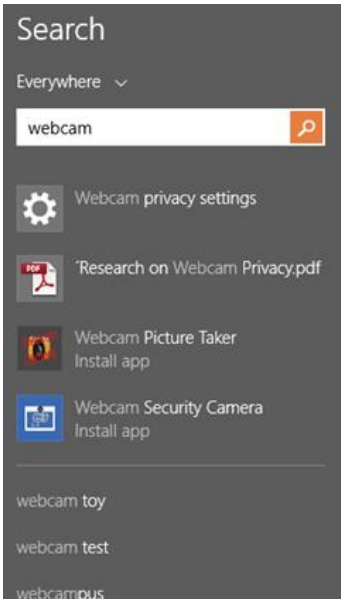

Fig. 5. Windows search provide auto-suggest across PC settings, local and cloud documents, the Web and Windows Store

4.4 Windows operating system caches

Windows 8.1 comes with an integrated search feature based on Bing SmartSearch. Search results are fetched from the local computer and from the Internet, as seen in Fig. 5. To provide auto-suggest, all keystrokes are forwarded to a Web service: https://www.windowssearch.com/suggestions?q=whateveryoutype that returns a cacheable XML response ("Cache-Control: private" response header). The search requests are cached by the INET cache built into Windows. This cache is shared between Internet Explorer, Windows components and even applications such as the Outlook mail client. Hence, SmartSearch and query completion work as side channels which can potentially leak data.

Attack implementation. Any intranet hosted web page opened in Internet Explorer can probe the cache. The intranet requirement stems from the security model of the Internet Explorer (IE) whereby Web content is classified into five security zones, each with a separate cache. However, from our observations, the auto-suggest feature is powered by requests made to Bing by the IE process running in the

local intranet zone. The attacker has to be associated with an intranet site rather than the local network; that is lowering the hurdle.

Impact. The attack is critical because any attacker that can publish in the intranet zone can sniff users' activities in the OS from within the browser. Brute force and dictionary techniques can recover users' searches which might contain software names (e.g., unreleased codenames, confidential intellectual property), personal files or Web searches. The requirement for the attacker to publish a website within an intranet is a hurdle but much lower than for traffic analysis-attacks that require access to the local network.

Mitigation. The cache for each OS component should be isolated, from any Internet or intranet Web pages.

Response by manufacturer. After our report, Bing enabled no-caching headers, effectively preventing CTAs on searches entered into the Windows SmartSearch.

5 Potential defences and their limits

Despite the previous research in this area, browser vendors have not yet taken actions to mitigate against CTAs. With deeper integration of Web services, it is critical that Web caches are carefully designed, by considering the timing side channel. Furthermore, with significant performance improvements of browsers in recent years [19], timing attacks have become more precise. Finally, as described in Section 3, multi-threading in JavaScript makes highly scalable brute force CTAs efficient. Thus, it is necessary to re-evaluate previously proposed mitigations and implement additional ones to address these issues.

5.1 Failed mitigations

Felten et al. [5] proposed various hypothetical solutions such as disabling caches and JavaScript, and altering hit or miss timings. Unfortunately, disabling JavaScript will break most of the modern websites and disabling caching will lead to significant performance degradation for the Internet. Furthermore, altering hit or miss timings is not possible without affecting user experience and slowing Web applications. Solutions in the form of "domain tagging" [5], i.e., tagging each request with the domain that is making a request, could help with stopping attackers but poses a problem when resources need to be shared across multiple domains. That is the case for content delivery networks (CDNs), in use at over 55% of the top Alexa 1000 websites [20]. The same applies to the "domain tagging" method by Jackson et al. [6].

5.2 HTTPS and private browsing don't help

HTTPS responses that are cached in the browser or by the OS are as vulnerable as HTTP responses. Caching is controlled only through the headers; thus, there is no differentiation between HTTPS and HTTP requests [21, 22]. *Private browsing* is a feature

of modern browsers designed to sandbox a session. Data (history, cookies, local storage) pertaining to the private session are discarded when the session is closed. This may seem like a reasonable mitigation; however, the cache is shared between all websites, tabs, and the search-box/address bar opened within the private session. A malicious website can still leak data from any website accessed within the private session. Furthermore, it can still sniff data from the proxy cache.

5.3 Our proposed mitigations

We have demonstrated how various Web caches can be exploited to leak data using CTAs. Any counter-measure which impacts performance or requires Web scale changes cannot be expected to be implemented. Furthermore, there is no single mitigation that can fix all the attacks we have described.

Cache segmentation: browser and OS. As discussed in Section 4.4, browser and OS components currently leak information because they share the cache with Web pages. Unlike proxy caches, even HTTPS requests are stored in browser caches, making this more critical. Segmentation is an easy yet effective fix. Each component, plugin or extension should have a separate cache so that there is no side channel leak to a malicious Web page or a malicious component.

Proxy cache. To prevent CTAs on Web proxies, appropriate no-cache headers should be set on all responses with any private data (e.g., API calls, image requests, etc.). The "Cache-control: private" header can be set to prevent any public proxy caching.

Browser cache. Because of the current design of the browser cache, this is the hardest case of CTAs to fix. To begin with, we recommend that none of the security critical resources (such as Site-to-user images) are cached at all. Website developers should disable caching by setting the appropriate no-cache headers. Eliminating caching altogether is simply not practical; the Web architecture relies on caches.

Issues with the "domain tagging" method can be resolved by using the recently introduced Web standard for "resource timing" API [23]. By default, the cache is shared among all origins. However, for privacy sensitive resources, a list of origins can be provided in the response. These are allowed to load a resource from the cache without having requested it earlier. However, this requires changes in the standards, browsers and also Web servers.

A more pragmatic approach is to append a user or session specific nonce ('number used once') to all URLs with privacy-sensitive data. The CTAs attacker would then need to know or guess the URL in order to probe for it. This mediation does not require Web-scale changes and can be implemented with a minimal effort by developers. Most of the requests by Google and YouTube, for instance, append a nonce.

Probing detection and prevention. As an alternative defence, JavaScript engines could be augmented to detect and then prevent cache probing, thereby blocking the attack vector. Detection could be through the source code [24], or through behavioural

analyses. Such mitigation adopts the route of antivirus software: instead of fixing the vulnerability we hinder its exploitation. Despite its near-term effectiveness, this approach incurs long-term costs and may result in an arms' race between attackers and their targets.

6 Discussion and concluding remarks

6.1 Cyber-espionage of corporate intranets

We have demonstrated how an attacker can de-identify users even on professional social networks (Section 4.2), monitor Web search queries in real-time (Section 4.3), and trace files that users are searching for on their desktop PCs (Section 4.4). An attacker can combine these mechanisms to harvest data from corporate intranets as well. For the exploitation of OS search, the attacker has to be on the intranet (e.g., a rogue employee), but for the rest, any Internet website can deploy the attack.

The caches built into Web proxies (Fig. 1) expand the CTA surface. All major commercial proxy server software have a built-in cache to enable faster loading of frequently accessed resources across users behind the proxy. They are susceptible to the same side channel attack as the browser cache. However, such CTAs are more critical since a malicious Web page can sniff traffic from closed intranets such as hospitals and educational institutions and corporate networks. In some cases, users may not be aware that they share a proxy with others, or cannot change the proxy settings. That particularly applies to the mobile Internet. In our experiments we were able to use a malicious web page to sniff users' queries and traffic from a corporate network.

Taken together, cache attacks at the browser, operating system, and proxy level open door to targeted cyber-espionage. Confidential information can be sniffed from a corporate network. Gathered information can also be used to mount a credible social engineering attack in a second step, for instance, in combination with targeted individuals, identified over Facebook (Section 4.2).

6.2 Policy recommendations and managerial implications

Finding the balance between surveillance and national security is an essential challenge for each society that recognises its citizens' preference for privacy and self-determination. Availability of ad-hoc surveillance "of anyone by anyone", within and outside the workspace, can be easily monetised by a service and, without proper intervention, can become a common practice and an economically viable enterprise. Once such practices take root, they are hard to weed out, as was the case with super cookies [25]. Thus, one would need to act swiftly to prevent the emergence of services that allow attack at scale.

At the corporate level, CIOs and system administrators need to be vigilant in guarding corporate information that can be revealed through employees' activities. This needs to go hand-in-hand with the policy recommendations and work with standardisation bodies to consider adaptations of the cache design.

7 Summary and Future Work

In this paper, we broadened the scope of cache timing attacks by demonstrating CTAs against banking sites, social networks, browser components, and operating systems. Our case studies show that cache timing attacks are applicable to much more critical scenarios than those previously considered. Our technique leverages fine-grained timeouts and HTML5 Web Workers to make the attacks more efficient, without contaminating the cache while probing. The robustness, effectiveness, and the non-cooperative nature of the attack increase the risk to undetectable, widespread attacks as part of the cyber espionage, ad-hoc surveillance of individuals and groups, un-consented identification and de-anonymisation. We discussed potential counter-measures and identified practical mitigations for each scenario that can be easily incorporated in the application design without requiring Web-scale changes.

Our future research will involve empirical studies of attack scenarios to evaluate both the applicability and the effectiveness of the CTAs. We plan a large-scale survey of cache implementations in operating systems across platforms (desktop, mobile, embedded), Web browsers and Web and desktop applications (e.g., mail, document authoring). Our work on systems security will be complemented by a user study to gauge reactions to the unexpected privacy invasions. We foresee outreach efforts to raise awareness amongst consumers and IT professionals.

Acknowledgements. The authors would like to thank Pushkar Chitnis, B. Ashok, Cedric Fournet, Sriram Rajamani and the reviewers for their helpful comments leading to significant improvements to this paper. We would also like to acknowledge the Microsoft and RSA Security teams for prompt and constructive discussions about our attacks.

References

1. Mozilla Developer Network and individual contributors, “Same-origin policy,” 2014. [Online]. https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy.
2. R. Gomer, E. M. Rodrigues, N. Milic-Frayling and M. Schraefel, “Network Analysis of Third Party Tracking: User Exposure to Tracking Cookies through Search,” in IEEE/WIC/ACM Int. J. Conf. on Web Intelligence and Intelligent Agent Tech. 2013.
3. J. P. Carrascal, C. Riederer, V. Erramilli, M. Cherubini and R. de Oliveira, “Your browsing behavior for a big mac: economics of personal information online,” in Proceedings of the 22nd International Conference on World Wide Web (WWW'13), 2013.
4. TRUSTe, “Behavioral Targeting: Not that Bad?! TRUSTe Survey Shows Decline in Concern for Behavioral Targeting,” 4 March 2009. [Online]. Available: http://www. truste. com/about-TRUSTe/press-room/news_truste_behavioral_targeting_survey.
5. E. W. Felten and M. A. Schneider, “Timing attacks on web privacy,” in Proceedings of the 7th ACM conference on Computer and Communications Security, 2000.
6. C. Jackson, A. Bortz, D. Boneh and J. C. Mitchell, “Protecting browser state from web privacy attacks,” in Proc. of the 15th Int. Conf. on World Wide Web (WWW), 2006.

7. G. Wondracek, T. Holz, E. Kirda and C. Kruegel, “A Practical Attack to De-anonymize Social Network Users,” in IEEE Symposium on Security and Privacy (SP), 2010.
8. C. Jackson, “SafeCache: Add-ons for Firefox,” 2006. [Online]. Available: https://addons.mozilla.org/en-US/firefox/addon/safecache/.
9. Y. Jia, X. Dongy, Z. Liang and P. Saxena, “I Know Where You've Been: Geo-Inference Attacks via the Browser Cache,” IEEE Internet Computing, p. forthcoming, 2014.
10. G. Yan, G. Chen, S. Eidenbenz and N. Li, “Malware propagation in online social networks: nature, dynamics, and defense implications,” in Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security (ASIACCS), 2011.
11. N. Provos, D. McNamee, P. Mavrommatis, K. Wang and N. Modadugu, “The Ghost in the Browser: Analysis of Web-based Malware,” in First Workshop on Hot Topics in Understanding Botnets (HotBots), 2007.
12. M. Zalewski, “Chrome & Opera PoC: rapid history extraction through non-destructive cache timing,” December 2011. [Online]. Available: http://lcamtuf.coredump.cx/cachetime/chrome.html.
13. J. Youll, “Fraud vulnerabilities in sitekey security at Bank of America,” 2006. [Online]. Available: www.cr-labs.com/publications/SiteKey-20060718.pdf.
14. Alexa Internet, Inc., “Top Sites in United States,” 2014. [Online]. Available: http://www.alexa.com/topsites/countries/US.
15. Facebook, “Company Info | Facebook Newsroom,” 2014. [Online]. Available: https://newsroom.fb.com/company-info/.
16. J. Bonneau and S. Preibusch, “The Privacy Jungle: On the Market for Data Protection in Social Networks,” in Eighth Workshop on the Economics of Information Security (WEIS 2009), 2009.
17. A. Pironti, P.-Y. Strub and K. Bhargavan, “Identifying Website Users by TLS Traffic Analysis: New Attacks and Effective Countermeasures,” INRIA, 2012.
18. S. Chen, R. Wang, X. Wang and K. Zhang, “Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow,” in IEEE Symposium on Security and Privacy (SP '10), 2010.
19. “The BIG browser benchmark (January 2013 edition),” [Online]. Available: http://www.zdnet.com/the-big-browser-benchmark-january-2013-edition-7000009776/.
20. Datanyze.com, “CDN market share in the Alexa top 1K,” 2014. [Online]. Available: http://www.datanyze.com/market-share/cdn/?selection=3.
21. MSDN, “HTTPS Caching and Internet Explorer - IEInternals,” 2010. [Online]. Available: http://blogs.msdn.com/b/ieinternals/archive/2010/04/21/internet-explorer-may-bypass-cache-for-cross-domain-https-content.aspx.
22. MozillaZine Knowledge base, “Browser.cache.disk cache ssl,” 2014. [Online]. Available: http://kb.mozillazine.org/Browser.cache.disk_cache_ssl.
23. W3C, “Resource Timing,” 2014. [Online]. Available: http://www.w3.org/TR/resource-timing.
24. G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens and B. Preneel, “FPDetective: dusting the web for fingerprinters,” in ACM SIGSAC conference on Computer and communications security (CCS), 2013.
25. M. Holter, “KISSmetrics Settles ETags Tracking Class Action Lawsuit,” Top Class Actions LLC., 22 October 2012. [Online]. Available: http://topclassactions.com/lawsuit-settlements/lawsuit-news/2731-kissmetrics-settles-etags-tracking-class-action-lawsuit/.