This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.2988369, IEEE Transactions on Dependable and Secure Computing

Website Fingerprinting Through the Cache Occupancy Channel and its Real World Practicality

Anatoly Shusterman, Zohar Avraham, Eliezer Croitoru, Yarden Haskal, Lachlan Kang, Dvir Levi, Yosef Meltser, Prateek Mittal, *Senior Member, IEEE,* , Yossi Oren, *Senior Member, IEEE,* , and Yuval Yarom, *Member, IEEE*

***Abstract*—Website fingerprinting attacks use statistical analysis on network traffic to compromise user privacy. The classical attack model used to evaluate website fingerprinting attacks assumes an *on-path adversary*, who observes traffic traveling between the user's computer and the network.**

In this work we investigate a different attack model, in which the adversary sends JavaScript code to the target user's computer. This code mounts a cache side-channel attack to identify *other* websites being browsed. Using machine learning techniques to classify traces of cache activity, we achieve high classification accuracy in both the open-world and the closed-world models. Our attack is more resistant than network-based fingerprinting to the effects of response caching, and resilient both to network-based defenses and to side-channel countermeasures. We carry out a real-world evaluation of several aspects of our attack, exploring the impact of the changes in websites and browsers over time, as well as of the attacker's ability to guess the software and hardware configuration of the target user's computer.

To protect against cache-based website fingerprinting, new defense mechanisms must be introduced to privacy-sensitive browsers and websites. We investigate one such mechanism, and show that it reduces the effectiveness of the attack and completely eliminates it when used in the Tor Browser.

I. Introduction

Over the last decades the World Wide Web has grown from an academic exercise to a communication tool that encompasses all aspects of modern life. Users use the web to acquire information, manage their finances, conduct their social life, and more. This shift to the so called virtual life has resulted in new challenges to users' privacy. Monitoring the online behavior of users may reveal personal or sensitive information about them, including information such as sexual orientation or political beliefs and affiliations.

Several tools have been developed to protect the online privacy of users and hide information about the websites they visit [21, 23, 81]. Prime amongst these is the Tor network [23], an overlay network of collaborating servers, called *relays*, that anonymously forward Internet traffic between users and web servers. Tor encrypts the network traffic of all of the users, and transmits it between relays in a way that prevents external observers from identifying the traffic of specific users. The Tor Project also provides the *Tor Browser* [97], a modified version of the Firefox web browser, that further protects users by disabling features that may be used for tracking users.

A. Shusterman, Z. Avraham, E. Croitoru, Y. Haskal, D. Levi, Y. Meltser, and Y. Oren are with the Ben-Gurion University of the Negev

L. Kang performed this work while at the University of Adelaide

P. Mittal is with Princeton University

Y. Yarom is with the University of Adelaide and Data61

Past research has demonstrated that encrypting traffic is not sufficient for protecting the privacy of the users [12, 33, 39, 41, 42, 50, 51, 61, 69, 76, 77, 83, 103, 104, 109]. Observable patterns in the metadata of encrypted traffic, specifically, the size of the transmitted data, its direction, and its timing, may reveal the web page that the user is visiting. Applying such *website fingerprinting* techniques to Tor traffic results in a success rate of over 90% in identifying the websites that a user visits over Tor [83].[1]

In this paper, we focus on an alternative attack model of exploiting micro-architectural side-channels, a less explored option for website fingerprinting. The attack model assumes a victim that visits a website under the attacker's control. The website monitors the state of the victim computer's cache, and uses that information to infer the victim's web activity in other tabs of the same browser or even in other browsers.

Because the attack observes the internal state of the target PC, rather than the network traffic, it offers the potential of overcoming traffic shaping, often proposed as a defense for website fingerprinting [13, 14, 18, 73, 105]. Similarly, the attack may be applicable in scenarios where network-based fingerprinting is known to be less effective, such as when the browser caches the contents of the website [41].

We note that the malicious website does not need to be fully under the control of the attacker. The attacker only needs to be able to inject JavaScript code via the website to the victim's browser. This can be done, for example, through a malicious advertisement or a pop-up window. Alternatively, documents released by former NSA contractor Edward Snowden indicate that some nation-state agencies have the operational capability to exploit this vector on a wide scale. In March 2013 the German magazine Der Spiegel reported on the existence of a tool called QUANTUMINSERT, which the GCHQ and the NSA could use to inject malicious code to any website [92]. The Der Spiegel claims that the tool has been used to attack the computers of employees at the partly-government-held Belgian telecommunications company Belgacom and to target high-ranking members of the Organization of the Petroleum Exporting Countries (OPEC) at the organization's Vienna headquarters. Finally, malicious advertisements are a viable option for injecting cache side-channel attacks to browsers [32].

For a small number of websites, under the closed-world model, Oren et al. [74] show the possibility of fingerprinting via malicious JavaScript code. However, beyond showing the

[1] *Website fingerprinting* is a misnomer. Fingerprinting identifies individual web pages rather than sites. Following this misnomer, in this work we use the term *website* to refer to specific pages, typically the homepage of the site.

ability to distinguish between a handful of websites, their work does not provide an analysis of the effectiveness of the technique. Furthermore, following the disclosure of the Spectre and the Meltdown attacks, which can also be potentially delivered via malicious JavaScript injection [54, 65], major vendors deployed defenses against browser-borne side-channel attacks. In particular, all modern browsers have reduced the resolution of the JavaScript time function, `performance.now()`, by several orders of magnitude [79, 102]. Traditionally, cache attacks require high-resolution timers, and while mechanisms to generate such timers in web browsers have been published [35, 55, 86], it is not clear that these can be used for website fingerprinting.

Thus, in this paper we ask: *Are cache-based attacks a viable option for website fingerprinting?*

Our Contribution

We answer this question in the affirmative. We design and implement a cache-based website fingerprinting attack and evaluate it in both the closed-world and the open-world models. We show that in both models our JavaScript-based attacker achieves high fingerprinting accuracy even when the attack is carried out on modern mainstream browsers that include all recently introduced countermeasures for side-channel (Spectre) attacks. Even when taking these countermeasures to the extreme, as is done in the Tor Browser, our attack remains effective, although with a drop in accuracy.

Our attack consists of collecting traces of cache *occupancy* while the browser downloads and renders websites. Adapting the techniques of Rimmer et al. [83], we use deep neural networks to analyze and to classify the collected traces. By focusing on cache occupancy rather than on activity within specific cache sets, our attack avoids the need for high resolution timers required by prior cache-based attacks. Furthermore, because our technique does not depend on the layout of the cache, it can overcome proposed countermeasures that randomize the cache layout [66, 80, 106].

We investigate the source of the information in the cache occupancy traces and show that they contain information from both the networking activity and the rendering activity of the browser. Using information from the rendering activity allows our attack to remain effective even in scenarios that thwart network-based fingerprinting, such as when the browser retrieves data from its response cache and not from the network, or when the network traffic is shaped.

We implement a potential countermeasure that introduces a high level of activity into the last level cache. We show that the countermeasure reduces the success rate of the attack. In particular, the noise completely masks the activity of the Tor Browser, reducing the attack accuracy to that of a random guess. This countermeasure results in a mean slowdown of 5% for CPU benchmarks, which we consider reasonable when visiting privacy-sensitive web sites.

Finally, we investigate several aspects that affect the real-world applicability of the attack. We show that changes in websites over time result in a gradual drop in the accuracy of the attack, whereas the mere act of updating the browser may result in significant drop in the ability to accurately detect websites. We evaluate the attacker's ability to probe the hardware configuration of the target host, and show that the attack can be resilient to incorrect guess of the hardware and software configuration of the target.

More specifically, we make the following contributions:

- We design and implement the cache occupancy attack, a cache-based side-channel attack technique which can operate with the low timer resolution supported in modern JavaScript engines. Our attacks only require a sampling rate six orders of magnitude *lower* than required for the prior attacks of Oren et al. [74] (Section IV).
- We evaluate the use of two machine learning techniques, CNN and LSTM, for fingerprinting websites based on the cache activity traces collected while loaded by the browsers (Section V).
- We show that cache-based fingerprinting has high accuracy in both the closed- and the open-world models, under a variety of operating systems and browsers (Section VI).
- We evaluate network-based and cache-based fingerprinting with the browser response cache enabled, and show that while the accuracy of network-based fingerprinting drops significantly, the accuracy of cache-based fingerprinting is not affected (Section VII-C).
- We show that cache-based fingerprints contain information both from the network activity and from the rendering activity of the target device. Therefore, cache-based fingerprinting maintains a high accuracy even in the presence of traffic molding countermeasures which force a constant bit rate on network traffic (Section VII-D).
- We explore real-world implications, including evaluating the effects of concept drift and the importance of using correct browser and cache size estimation on the fingerprinting accuracy, as well as evaluating a technique for automatically determining the cache size (Section VIII).
- We design and evaluate a countermeasure that introduces noise in the cache. The countermeasure is applicable from both native code and from JavaScript and completely blocks the attack on the Tor Browser, with a small performance degradation. (Section IX).

II. Background

A. Tor

Tor [23], is a collection of collaborating servers called *relays*, designed to provide privacy for network communication. Tor aims to protect users from *on-path* adversaries that can observe the network traffic. In this scenario, a user uses a PC to browse the web, and an adversary positioned between the user's PC and the destination web server captures the information that the user exchanges with the web server.

A common protection for such an attack model is to use encryption, e.g., using protocols such as TLS [22] which underlies the security of the HTTPS scheme [82]. However, this solution only protects the contents of the communication, leaving the identity of the communicating parties exposed to the adversary. Merely knowing that users connected to a

certain sensitive website may be enough to incriminate them, even if the actual data exchanged over the secure connection is not known. This risk became a reality in 2016, as tens of thousands of individuals were persecuted by the Turkish government for accessing the domain `bylock.net` [56].

The main aim of Tor is thus to protect the identity of the communicating parties. Tor achieves this protection by forwarding the users' communication through a *circuit* consisting of typically three Tor relays. The user encrypts the network traffic with multiple layers of encryption, and each relay in the circuit decrypts a successive layer to find out where to forward the traffic. See Dingledine et al. [23] for further information.

B. Website Fingerprinting Attacks and Defences

In the conventional attack model of a network-level attacker, much previous work has demonstrated the ability of an adversary to make probabilistic inferences about users' communications via statistical analysis, even if these communications are in their encrypted form. These works have investigated both the selection of features (such as packet sizes, packet timings, direction of communication), as well as the design of classifiers (such as Support Vector Machines, Random Forests, Naive Bayes) to make accurate predictions [12, 33, 39, 41, 42, 50, 51, 61, 69, 76, 77, 83, 103, 104, 109]. In response, several defense mechanisms have been proposed in the literature [5, 13, 14, 18, 73, 105]. The common idea behind these defenses is to inject random delays and spurious cover traffic to perturb the traffic features and therefore obfuscate users' communications. A common point of all of these defenses is a typical trade-off between latency/bandwidth and privacy, and thus they face deployment hurdles. Rimmer et al. [83] have recently proposed a family of classifiers based on deep learning algorithms such as SDAE, CNN and LSTM, which operate on the raw network traces, and are therefore less sensitive to ad-hoc defenses against particular traffic features. Following this work, Sirinam et al. [90] proposed different CNN architectures that outperform previous attacks on Tor Browser, and that can withstand the WTF-PAD [52] countermeasure that modifies traffic characteristics.

One of the drawbacks of previous works is the high number of traces required per website. To address this, Bhat et al. [7] propose a neural network with a ResNet [40] architecture, for high accuracy classification using a small amount of data of packet timing information. Their experiments use only 100 traces per website, achieving results comparable to previous works which use thousands of traces. Sirinam et al. [91] suggest another approach, using N-Shot learning [60]. Their approach compares pairs of feature traces for same and different websites. It outputs a feature vector which is matched against other output vectors of traces with known labels, and classified to the most similar traces.

Another limitation of many website fingerprinting works is the single-tab surfing assumptions. To overcome this limitation, Xu et al. [108] proposed a classifier, combining the Balance-Cascade [68] method and the XGBoost [16] classifier, which finds the split point at which a second webpage is loaded in another tab, and then classifies the first website. A different approach, proposed by Zhuo et al. [114], uses a Profile Hidden Markov Model (PHMM) [57]. This method builds a profile out of a network trace of visiting a home-page of a website, followed by deeper pages of the same site. It then calculates the probabilities of a label given a sequence of network traces, while treating the probabilistic noise inside each network trace.

C. Cache Side-Channel Attacks

When programs execute on a processor, they share the use of micro-architectural components such as the cache. This sharing may result in unintended communication channels, often called *side channels*, between programs [31, 44], which may be used to leak secret information. In particular, cache-based attacks, which exploit contention on one of the processor's caches, can leak secrets such as cryptographic keys [4, 30, 75, 78, 98], keystrokes [36], address layout [27, 35, 37], etc.

Cache Operation. Caches bridge the speed gap between the faster processor and the slower memory. The cache is a small bank of memory, which stores the contents of recently accessed memory locations. Most caches in modern processors are *set associative*. The cache is divided into partitions called *sets*. Each memory location maps to a single set and can only be cached in the set it maps to. When the processor needs to access a specific memory location, it successively searches in a hierarchy of caches. In a *cache hit*, when the contents of the required address is found in the cache, access is performed on the cached contents. Otherwise, in a *cache miss*, the process repeats on the next cache level. A miss on the last-level cache (LLC) results in a time-consuming access to the RAM.

The Prime+Probe Technique. Past cache-based attacks from web browsers [32, 74] employ the *Prime+Probe* technique [75, 78], which exploits the set-associative structure. Each round of attack consists of three steps. In the first step, the cache is *primed*, i.e., the attacker completely fills some of the cache sets with their own data. The attacker then waits some time to allow the victim to execute. Finally, the attacker *probes* the cache by measuring the time it takes to access the previously-cached data in each of the sets. If the victim accesses memory locations that map to a monitored cache set, the victim's memory contents will replace the attacker contents in the cache. Hence, the attacker will need to retrieve the data from lower levels in the hierarchy, increasing the access time to its data. Prime+Probe has been used for attacks on data [75, 78] and instruction [3, 4] caches, as well as for attacks on the LLC [48, 67]. It has been shown practical in multiple settings, including across different virtual machines in cloud environments [45] and from mobile code [32, 74].

Countermeasures in JavaScript. The time difference between the latencies of a memory access and cache access is on the order of 0.1 μs. To distinguish between cache hits and misses, cache attacks typically require a high resolution timer. Following the first demonstration of a cache attack in JavaScript [74], some browsers reduced the resolution of the timers they provide. This approach had become widespread after the disclosure of the Spectre attack [54], and now all mainstream browsers incorporate this countermeasure. Furthermore, while non-traditional timers in browsers have

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.2988369, IEEE Transactions on Dependable and Secure Computing

been identified [29, 55, 86], browsers and extensions have since disabled many of the features that allow sub-microsecond resolution [71, 79, 87]. In particular, the Tor Browser restricts the timer resolution to 100 ms, or 10 Hz.

Several of the previously discovered timers rely on browser features that are accessible from JavaScript. These are not accessible in environments such as Cloudflare Workers [9], which rely on the absence of high-resolution timers to protect against timing attacks [100].

D. Related Work

Several past works have looked at the possibility of performing website fingerprinting based on local side-channel information. In all of these works, which we survey in Table I, the adversary observes some property of the system while the victim browser is rendering a webpage. The adversary then applies a machine learning classifier to the observed side-channel trace to identify the rendered website.[2] Some of these works assume that the adversary has malicious control over a hardware component or peripheral [19, 64, 110]. Others assume that the adversary can execute arbitrary native code on the target hardware [38, 49, 58, 70, 94]. Yet others only assume that the adversary can induce the victim to render a webpage containing malicious JavaScript code [10, 53, 70, 74, 101]. We mainly investigate the last model.

Kim et al. [53] abuse a data leak in the Chrome implementation of the Quota Management API, which has been since fixed. Our attack, in contrast, is based on a fundamental property of the CPU running the browser application, which is far less trivial to fix (see Section IX). Moreover, the mitigations put in place as part of the response to the Spectre and Meltdown disclosures make the high sampling rates exploited thus far [74, 101] unattainable in modern secure browsers. Our attack, in contrast, achieves high accuracy at drastically lower sampling rates and is capable of classifying a significant number of websites at sampling rates as low as 10 Hz. To the best of our knowledge, no cache attack that uses such low clock resolutions has been demonstrated.

In addition, Oren et al. [74] only recorded a small number of traces from a few popular websites, and did not investigate the effectiveness of cache-based fingerprinting in open-world contexts, or in scenarios where various anti-fingerprinting measures are in place. We address all of these shortcomings in this work. Furthermore, while Oren et al. [74] do target the Tor Browser, the attack code executes in a different mainstream browser. Unlike our work, they do not demonstrate an attack from JavaScript code running within the Tor Browser.

Booth [10] is able to classify a moderate amount of websites using a non-cache-based method with a millisecond clock. Their attack, however, saturates all of the victim's CPU cores with math-intensive worker threads, making it highly noticeable and easy to detect by the victim.

Cock et al. [20] implement a covert channel using an L1 cache occupancy channel. Ristenpart et al. [84] show that a cache occupancy channel can detect keystroke timing and network load in co-located virtual machines on cloud servers. Both use the technique with high resolution (sub nanosecond) timers. We are not aware of any prior use of the cache

[2] A different but closely related class of attacks are "history sniffing" attacks, such as [62, 107], in which the attacker wishes to learn which websites the victim has visited in the **past**.

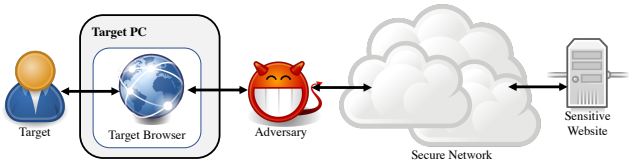


Figure 1: The classical website fingerprinting attack model. The (passive) adversary monitors the traffic between the target user and the secure network.

The classical attack model used to evaluate website fingerprinting attacks is presented in Figure 1. Here, the victim uses a web browser to display a sensitive website. To protect their privacy, the victim does not connect to the website directly, but instead uses a secure network, such as the Tor network, for the connection. The attacker is typically modeled as an *on-path adversary*, who is capable of observing all traffic entering and leaving the Tor network in the direction of the target user. The adversary cannot understand the contents of the network traffic since it is encrypted when it enters the Tor network. The adversary is furthermore unable to directly determine the ultimate destination of the communications after it exits the Tor network, thanks to Tor's routing protocol. Finally, due to the encryption and the validation of the Tor network, the attacker is unable to modify the traffic without terminating the connection. An important thread of research on the security of Tor has investigated the ability of such an adversary to perform statistical traffic analysis of encrypted traffic, and then to make probabilistic inferences about the victim's communications [12, 39, 41, 42, 50, 51, 61, 69, 76, 77, 83, 103, 104, 109].

Gong et al. [33] suggest a variation on this scheme, in which the attacker remotely probes routers to estimate the load of the network traffic they process and performs the statistical analysis based on this estimate. Jansen et al. [50] suggest another variation in which the attacker monitors the traffic inside the Tor network, rather then at the network's edge.

In this work we discuss a different attack model, presented in Figure 2. In this model, the target user has two concurrent browsing sessions. In one session, the user browses to an adversary-controlled site, which contains some malicious JavaScript code. In the other session, the user browses to some sensitive website. Due to architectural boundaries, such as sandboxing or process isolation, the malicious code cannot directly observe the internal state of the sensitive session. Hence, the adversary cannot directly determine the destination of *any* communication issued from the sensitive session, even when the sensitive session is using a direct unencrypted connection to the remote server. The malicious code can, however, observe the micro-architectural state of the processor, and use this information to spy on the sensitive session.

Table I: Related work on website fingerprinting based on local side channels.

Work	Target	Side Channel	Attack Model	Sampling rate [Hz]
Clark et al., 2013 [19]	Chrome (Mac, Win, Linux)	Power consumption	Hardware	250000
Yang et al., 2017 [110]	Multiple smartphones	Power consumption	Hardware	200000
Lifshits et al., 2018 [64]	Android Browser, Chrome Android	Power consumption	Hardware	1000
Jana and Shmatikov, 2012 [49]	Chrome Linux, Firefox Linux, Android Browser (VM)	App memory footprint	Native code	100000
Lee et al., 2014 [58]	Chromium Linux, Firefox Linux	GPU memory leaks	Native code	N/A
Spreitzer et al., 2016 [94]	Chrome Android, Android Browser, Tor Android	Data-Usage Statistics	Native code	20–50
Gülmezoglu et al., 2017 [38]	Chrome Linux (Intel and ARM), Tor Linux	Performance counters	Native code	10000
Matyunin et al., 2019 [70]	Multiple smartphones	Magnetometer	Native code and JavaScript	10–100
Oren et al., 2015 [74]	Safari MacOS, Tor MacOS	Last-level cache	JavaScript	10^8
Booth, 2015 [10]	Chrome (Mac, Win, Linux), Firefox Linux	CPU activity	JavaScript	1000
Kim et al., 2016 [53]	Chromium Linux, Chrome (Win, Android)	Quota Management API	JavaScript	N/A
Vila and Köpf, 2017 [101]	Chromium Linux, Chrome Mac	Shared event loop	JavaScript	40000
This work	**Chrome (Win, Linux), Firefox (Win, Linux), Safari MacOS, Tor Linux**	**Last-level cache**	**JavaScript**	**10–500**

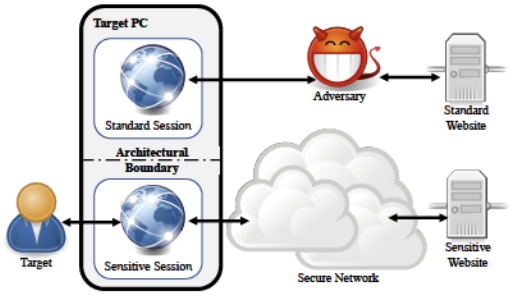


Figure 2: Remote cache-based website fingerprinting attack model. The remote attacker injects malicious JavaScript code into a browser running on the target machine.

Our attack can therefore be considered in two scenarios:

- A *cross-tab scenario*, where a user is made to visit an attacker-controlled website containing malicious JavaScript, and this website tries to learn what other sensitive sites the user is visiting at the same time. These attacker-controlled and sensitive browsing sessions can be carried out on the same browser, on two different browsers belonging to the same user, or even on two browsers residing in two completely isolated virtual machines which share the same underlying hardware [85]. One possible way of causing the user to browse to such an attacker-controlled site is through a phishing attack, where the attacker sends fraudulent messages, purporting to be from a benign source, that induces the victim to click on a link to a malicious website. Alternatively, the attacker may pay an advertisement service to display a (malicious) advertisement when the user visits a third-party website [32].
- A *cross-network scenario*, where the attacker is an active on-path adversary capable of injecting JavaScript into any non-encrypted page. The attacker would like to leverage that access to try to learn about the user's sensitive activity, even though the attacker cannot manipulate or access this traffic directly. For example, the user may simultaneously run one browsing session over an unsecured connection for mundane tasks, and another browsing session over a second, secured connection for sensitive tasks. An attacker capable of modifying traffic on the standard link can learn about activity carried out over the secured link, whether this secure connection made through a VPN, through the Tor network, or even through a separate network adapter which the attacker cannot see.

The main challenge of our attack model is the extremely restricted JavaScript runtime, which requires the attacker code to be written in a particular way, as described in Section IV.

Regardless of the delivery vector, cache-based fingerprinting has a strong potential advantage over network-based fingerprinting, since it can indirectly observe both the computer's network activity and the browser's rendering process. As we demonstrate in Section VII-D, both of these elements contribute to the accuracy of our classifier.

IV. Data Collection

A. Creating memorygrams

The raw data trace for network-based attacks takes the form of a *network trace*, commonly in the `pcap` file format, which contains a timestamped sequence of all traffic observed on a certain network link. The corresponding data trace in the case of cache attacks is the *memorygram* [74]—measured at a constant sampling rate over a given time period. The memorygrams of Oren et al. [74] describe the latency of multiple individual sets or groups of sets at each point in time, resulting in a two-dimensional array. In contrast, in this work we use a simplified, one-dimensional memorygram form. The contents of each entry in our memorygrams is a proxy for the occupancy of the cache at the specific time period. We collect memorygrams while the browser loads and displays websites, and use the data as fingerprints for website classification.

The Cache Occupancy Channel. Unlike prior works [32, 74], which use the Prime+Probe side-channel attack from JavaScript, we use a cache occupancy channel. The main difference is that the Prime+Probe attack measures contentions in specific cache sets, whereas our attack measures contention

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.2988369, IEEE Transactions on Dependable and Secure Computing

over the whole cache. Specifically, our JavaScript attack allocates an LLC-sized buffer and measures the time to access the entire buffer. The victim's access to memory evicts the contents of our buffer from the cache, introducing delays for our access. Thus, the time to access our buffer is roughly proportional to the number of cache lines that the victim uses. Cache occupancy has previously been implemented in native code and used for covert channels and for measuring co-resident activity [20, 84]. Both of these implementations rely on high resolution timers. We are not aware of any prior use of the cache occupancy channel with a low resolution timer.

Native-code and JavaScript Memorygrammers. The results in this paper compare two different memorygramming methods – a native-code memorygrammer based on the Mastik toolkit [111], which is written in C, and a portable code memorygrammer, which is written in JavaScript. While both the native-code memorygrammer and the JavaScript memorygrammer run without super-user permissions, the native-code memorygrammer offers several advantages to the attacker. First and foremost, the native-code memorygrammer has access to high-resolution timers, on the order of nanoseconds, and is also able to query the CPU's internal performance monitoring counters. The JavaScript memorygrammer, in contrast, has more limited timer access. Another advantage of the native-code memorygrammer is its direct access to memory. While the native-code memorygrammer, which is running in user mode, cannot completely map between its virtual address space and physical memory, it can still determine the LLC cache set responsible for each memory location in its own address space. This is due to the use of the "huge pages" memory mapping mode, in which the lowest 21 bits of the virtual address are equal to those of the physical address. The JavaScript memorygrammer, in contrast, is unable to directly access memory, neither virtual nor physical, and relies on accesses to JavaScript array objects, whose base address is completely unknown to the attacker. We therefore consider the native-code results to be a form of upper bound on the performance of the cache occupancy channel, against which the JavaScript results can be compared.

Overcoming Hardware Prefetchers. Ideally, we would like to collect information across the whole cache. Intel processors, however, try to optimize memory accesses by prefetching memory locations that the processor predicts will be accessed in the future. Because prefetching changes the cache state, we need to fool the prefetchers. To fool the spatial prefetcher [47], we use the technique of Yarom and Benger [112] and do not probe adjacent cache sets. To fool the streaming prefetcher, which tries to identify sequences of cache accesses, we use a common approach of masking access patterns by randomizing the order of the memory accesses we perform [67, 75].

Spatial Information. Compared with the Prime+Probe attack, the cache occupancy channel does not provide any spatial information. That is, the adversary does not learn any information on the addresses that the victim accesses. While this is a clear disadvantage of the cache occupancy channel, our attack does not require spatial information. The main reason is that modern browsers have complex memory allocation patterns. Consequently, the location at which data is allocated changes each time a page is downloaded, and the location carries little information on the downloaded page. In practice, not having spatial information is also an advantage. Without it, there is no need to build eviction sets for cache sets, a process that can take significant time [32].

Website Memorygrams. We capture memorygrams when the browser navigates to websites and displays them. We use a JavaScript-based memorygrammer to probe the cache at a fixed rate of one sample every 2 ms. We continue the probe for 30 seconds, resulting in a vector of length 15,000. When a probe takes longer than 2 ms, we miss the slot of the next probe. We use a special value to indicate this case. We use this collection method for all mainstream browsers other than the Tor Browser.

When the attack code is launched from within the Tor Browser, where the timer resolution is limited to 100 ms, we do not measure how long a sweep over the cache takes, but instead count how many sweeps over the entire cache fit into a single 100 ms time slot. In addition, we do not probe for 30 seconds in this setting, but rather for 50 seconds, to account for the slower response time over the Tor network. Hence, Tor memorygrams contain 500 measurements over the entire 50 second measurement time period.

The native-code memorygrammer used for the evaluations in Section VII does not suffer from a reduced timing resolution when measuring the Tor Browser. Therefore, on mainstream browsers it runs for 30 seconds and produces 15,000 entries, and on the Tor Browser it runs for 50 seconds and produces 25,000 entries.

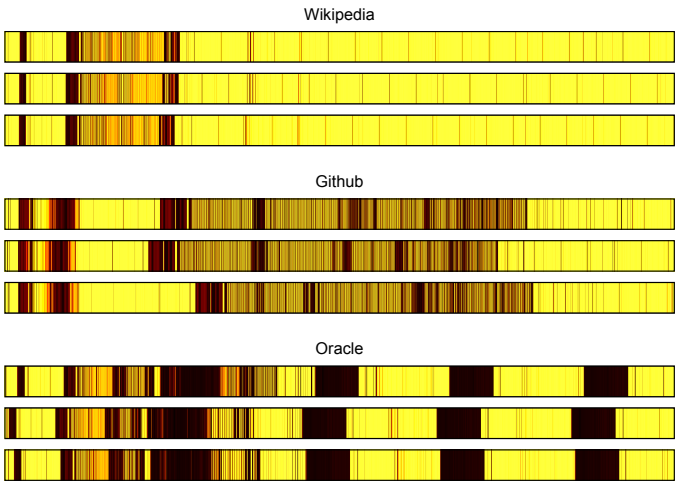


Figure 3: Examples of memorygrams. Time progresses from left to right, darker shades correspond to more evictions.

Sanity Check. Before proceeding, we want to verify that memorygrams can be used for fingerprinting. Indeed, Figure 3 shows graphical representations of memorygrams of three sites: Wikipedia (https://www.wikipedia.com), Github (https://www.github.com), and Oracle (https://www.oracle.com), collected through the native-code memorygrammer. Each memorygram is displayed as a colored strip, where time goes from

left to right and the shade corresponds to cache activity (darker shades correspond to more evictions). We see that the three memorygrams of each site, while not identical, are similar to each other. The memorygrams of different websites are, however, very different from each other. This indicates that memorygrams may be used for identifying websites.

B. Datasets

Closed World Datasets. We evaluate our cache-based fingerprinting on six different combinations of browsers and operating systems, summarized in Table II. Many early works on website fingerprinting operated under a *closed world assumption*, where the attacker's aim is to distinguish among accesses to a relatively small list of websites. Our closed world datasets follow this line of work. These datasets consist of 100 traces each for a set of 100 websites, to a total of 10,000 memorygrams. We use the same list of 100 websites that Rimmer et al. [83] selected from the top Alexa sites. Similar to previous works, no traffic molding is applied and only one tab is opened at a time.

Open World Datasets. One common criticism of the closed world assumption is that it requires the attacker to know the complete set of websites the victim is planning to visit, allowing the attacker to prepare and train classifiers for each of these websites. This assumption was challenged by many authors, for example Juárez et al. [51]. To address this criticism, website fingerprinting methods are often evaluated in an open-world setting. In this setting, the attacker wishes to monitor access to a set of sensitive websites, and is expected to classify them with high accuracy. Additionally, there is a large set of non-sensitive web pages, all of which the attacker is expected to generally label as "non-sensitive".

To evaluate our fingerprinting method in the open-world settings, we augment the closed-world datasets with additional 5,000 traces, each collected for a single unique website, again using the list of websites provided by Rimmer et al. [83]. The base rate for this setting is 33.3%, since a trivial classifier can simply decide that all pages are non-sensitive.

C. Limiting Assumptions

As noted by Juárez et al. [51], many academic works that deal with website fingerprinting make assumptions on the conditions of the attacker and the system under attack that are different from those encountered outside the lab. Our threat model and data collection protocol also makes several assumptions, as listed below.

Synchronization. Each trace in our data set contains a single web browsing session, from beginning to end. A real-world attacker would be faced with a continuous trace where the beginning and end of browsing sessions is not clearly marked, and in which multiple browsing sessions may overlap. In the network-based website fingerprinting scenario, little to no traffic travels through the network unless the user is actively fetching a webpage. This makes the task of synchronization relatively easy. In the cache-based scenario, however, the cache is always active to a degree, even before the browser starts to receive and render the webpage. Recognizing the start of a trace may therefore be more difficult in the cache-based setting than in the network-based setting, especially in the case of a real attack. Our framework implicitly synchronizes the trace with the start of the download. Due to varying network conditions, we see differences of up to six seconds between trace start and render start. As such, we believe that our technique can identify websites even without the synchronization. Further experimentation is required, however, to verify this fact. We also note that if the machine is otherwise idle, cache activity can serve as a (slightly noisy) indicator of the start of the trace.

Hardware Diversity. Despite the diversity of CPU generations and configurations evaluated in this work, we only used Intel CPUs. While in principle the cache contention attack is agnostic of the specific structure of the cache, more experiments are needed to verify its effectiveness on other CPU architectures, such as Arm and AMD.

Full Cache Eviction. Our JavaScript code allocates a buffer of the size of the victim's LLC, and repeatedly accesses it to observe cache occupancy. In Section VIII-D we evaluate the adversary's ability to correctly estimate the size of the cache. The question remains, however, as to whether a single pass over this buffer will cause it to fill up the whole LLC, evicting all other entries. Unless the cache uses a true LRU policy, some entries may not be replaced in a single pass over the buffer. Worse, the mapping from virtual addresses used by the program to physical addresses used to index the cache is unlikely to be uniform. In Section VIII-C we show that the attacker does not need to have an exact coverage of the cache for the attack, hence we believe that full cache eviction is not necessary for the attack.

V. Machine Learning

A. Problem Formulation

Website fingerprinting is generally formulated as a supervised learning problem, consisting of a template building step and an attack step. In the template building step, the adversary visits each target website multiple times and collects a set of labeled traces (either network traces or memorygrams), each corresponding to a visit to a certain website. Next, the adversary trains a classifier on these labeled traces, using either classical machine learning methods or deep learning methods.

In the attack step, the adversary is presented with a set of unlabeled traces, each one corresponding to a visit to an unknown website. The adversary then applies the previously trained classifier to each of these traces and outputs a guess for each trace. The accuracy of the classifier is finally calculated as the percentage of the correctly assigned labels.

B. Deep Learning Models

Early works on website fingerprinting, starting from Cheng and Avnur [17], used classical machine learning methods such as Naive Bayes, Support Vector Machine (SVM) and k-Nearest Neighbors (KNN). As a prerequisite step to running these classical machine learning methods, the adversary needs to

apply an additional feature extraction step which transforms the raw trace into a more succinct representation. Since these features were chosen through human insight into the nature of network traffic, there was no immediate way of directly applying them to memorygram analysis.

Abe and Goto [2] and later Rimmer et al. [83] suggest using deep learning for website fingerprinting. Deep learning performs automatic feature learning from the raw data, reducing the reliance on human insight at the cost of a larger required training set. Rimmer et al. [83] show that, given a large enough training set, deep-learning website-fingerprinting approaches are as effective as earlier methods. An advantage of this approach is that it allows us to compare network-based and cache-based fingerprinting operating directly on the raw data, rather than on a specific choice of features.

Deep Neural Network Configuration. A deep neural network (DNN) is typically configured as a sequence of non-linear layers which transform the raw data, first extracting salient features and then selecting the appropriate ones [34]. Every layer in a DNN consists of a set of artificial neurons, each connected to a set of outputs from the previous layers. At the forward propagation stage, the activation function is applied to the product of the each neuron's input and its weight value, and then forwarded to the next layer. As a last layer, we use a softmax layer, which outputs a vector containing a-posteriori probabilities for each of the classes.

The process of training the neural network uses back-propagation to update the weights of each neuron to achieve a minimum loss at the output. First, the model calculates the cost between the true classification of the measurement and the predicted value using a loss function. Next, the model updates the weights of the each neuron based on the calculated loss. Every round of forward propagation and back-propagation is called an epoch. A neural network model runs multiple epochs to learn the weights for accurate classification.

We evaluate deep learning using two classifier models, Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks [43]. A CNN uses a sequence of feature mapping layers alternating between convolutions and max-pooling. Each of the layers sub-samples the previous layer, iteratively reducing the size of the input to a more succinct representation, while preserving the information they encode. Each convolutional layer is a neural network specialized for detecting complex patterns in its input. The convolution layer applies several filters to the input vector, each of which is designed to identify an abstract pattern in a sequence of input elements it is provided with. The max-pooling layers reduce the dimensionality of the data by subsampling the filters, choosing the maximum value from adjacent groups of neurons applied by the filters. This alternating sequence of layers extracts complicated features from the input and produces vectors short enough for the classifiers. The feature mapping layers are followed by a *dense* layer, in which every neuron is connected to every output of the feature extraction phase. The LSTM-based network has an initial feature selection step similar to the CNN, but then adds a layer in which each neuron has a memory cell, with the output of this neuron determined both by its inputs and by the value of this memory cell. This allows the classifier to identify patterns in time-based data.

Hyperparameter Selection. *Hyperparameters* describe the overall structure of the DNN and of each layer. The choice of hyperparameters depends on the specific classification problem. For network-based fingerprinting, we replicated the parameters specified in the dataset provided by Rimmer et al. [83]. For cache-based fingerprinting, we manually evaluated several choices for each hyperparameter.

To prevent overfitting, we use 10-fold cross validation. We split each dataset into 10 folds of equal size, and select one fold, as a *test set*. The remaining 90% of the traces are used for training the classifier, with 81% serving as the *training set* and 9% as the *validation set*. The model trains on the training set and the evaluation is done on the test set. The number of epochs is regulated with an Early-Stop function which stops the epochs when the accuracy of the validation set no longer increases over successive iterations.

For the CNN classifier we use three pairs of convolution and max pooling layers. For the LSTM classifier we use two. As discussed above, the traces captured by the code running within the Tor Browser contain only 500 measurements, due to the reduced timer resolution. For these shorter traces, we modified the architecture of our LSTM-based classifier. The feature selection of this classifier contains only one convolution layer. We therefore used a pool-size of three for the max-pooling layer to limit the feature reduction before the LSTM layer. In addition, because of the small amount of features, we could increase the number of LSTM units to 128 and learn more complex patterns. The full hyperparameter tuning space is described in Shusterman et al. [88, Appendix A].

VI. Results

All of the results in this section were obtained by using Keras version 2.1.4, with TensorFlow version 1.7 as the back end, running on two Ubuntu Linux 16.04 servers, one featuring two Xeon E5-2660 v4 processors the other two Xeon E5-2620 v3, both with 128 GB RAM. Our machine learning instances took approximately 40 minutes to run in this configuration.

Table II presents the fingerprinting accuracy we obtain. Recall that in this scenario the JavaScript interpreter of the targeted browser executes the memorygrammer. Considering that all modern browsers reduced their timer resolution and some added jitter as a countermeasure for the Spectre attack [79, 102], the first question we need to address is whether it is even possible to implement cache-based fingerprinting attacks in such an environment.

To answer this question, we measured the latencies of the cache occupancy channel as the browser was rendering a representative webpage, using the native code memorygrammer. The measurement was made on a desktop computer featuring an Intel Core i5-25006 CPU at 3.30 GHz with 6 MB last-level cache, running CentOS 7.2.1511. Figure 4 shows the cumulative distribution function (CDF) of the latencies of the 14,632 samples collected while rendering the Facebook home page (https://facebook.com). The figure also highlights the timer resolutions of three mainstream browsers. (See Table II.)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.2988369, IEEE Transactions on Dependable and Secure Computing

Table II: Accuracy obtained by in-browser memorygrammerdeviation.

Operating System	CPU	LLC Size	Browser	Timer Resolution	Closed World CNN	Closed World LSTM	Open World CNN	Open World LSTM
Linux	i5-2500	6 MB	Firefox 59	2.0 ms	78.5±1.7	80.0±0.6	86.8±0.9	87.4±1.2
Linux	i5-2500	6 MB	Chrome 64	0.1 ms	84.9±0.7	91.4±1.2	84.3±0.7	86.4±0.3
Windows	i5-3470	6 MB	Firefox 59	2.0 ms	86.8±0.7	87.7±0.8	84.3±0.6	87.7±0.3
Windows	i5-3470	6 MB	Chrome 64	0.1 ms	78.2±1.0	80.0±1.6	86.1±0.8	80.6±0.2
Mac OS	i7-6700	8 MB	Safari 11.1	1.0 ms	72.5±0.7	72.6±1.3	80.5±1.0	72.9±0.9
Linux	i5-2500	6 MB	Tor Browser 7.5	100.0 ms	45.4±2.7	46.7±4.1	60.5±2.2	62.9±3.3
Linux	i5-2500	6 MB	Tor Browser 7.5 (top 5)	100.0 ms	71.9±2.1	70.0±1.7	80.4±1.7	82.7±1.8

As we can see, even at the 2 ms resolution of the Firefox 59 timer, it is possible to distinguish between 80% of the probes which take less than 2 ms and the remaining 20%. This is a welcome side-effect of the use of a large buffer which is accessed at every probing step. None of the cache probes we measured, however, took longer than the 100 ms clock period of the Tor Browser. Hence, when running within the Tor Browser, we count the number of probes we can perform within each clock tick. (See Section IV.)

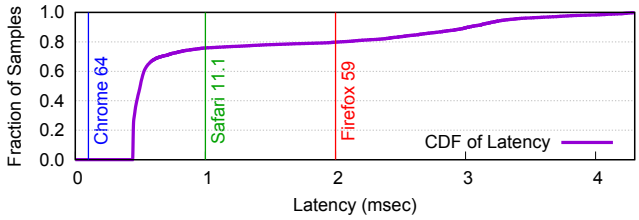


Figure 4: Cache probe latencies vs. browser timer resolutions.

The next question is whether the information we collect with such a low resolution is sufficient for fingerprinting. Indeed, Table II shows that in all of the environments we test our classifier is significantly better than a random guess. Remarkably, as our results show, even the highly restricted Tor Browser can be used for mounting cache attacks, albeit with a significantly lower accuracy than that mainstream browsers.

A. Closed World Results

We first look at the typical closed-world scenario investigated by past works. In mainstream browsers, our JavaScript attack code is consistently able to provide classification accuracies of 70–90%, well over the base rate of 1%. The Tor Browser attack, however, achieves a lower accuracy of 47%. Yet, if we look not only at the top result output by the classifier, but also check whether the correct website is one of the top five detected websites, the accuracy of the Tor Browser attack climbs to 72%, with a base rate of 5%. This method of looking at the few most probable outputs of a classifier was previously used in similar classification problems [15, 72]. With some a-priori information an attacker can deduce which of the top five pages the victim has accessed.

We can compare the accuracy of our cache-based fingerprinting to the one obtained by state-of-the-art network-based methods, as reported by Rimmer et al. [83]. We see that while there are differences between the classification accuracies achieved in each case, the overall accuracy is comparable, assuming both attacks capture the same amount of traces per website. As in the network-based setting, we believe that capturing more than 100 traces per website is likely to increase the accuracy and the stability of our classifier.

B. Open World Results

We next turn into a different scenario of open-world dataset. Recall that in this scenario the classifier needs to distinguish between 101 classes. These include one class for each of the 100 sensitive websites, as well as one generic non-sensitive class for all 5000 websites not included in the sensitive classes. The best strategy for a random classifier in this case would be to always classify all traces as non-sensitive, providing a base accuracy rate of 33%. As seen in Table II, the accuracies the classifiers achieve in this case are 70–90%, slightly better results than in a closed-world scenario. The reason might be that the classifier easily recognizes the non-sensitive class that includes 33% of the traces in the dataset.

If we group all of the sensitive classes into a meta-class of "sensitive websites", the classification between sensitive vs. non-sensitive sites becomes a binary classification problem. We can, therefore, apply standard analysis techniques to this aspect of the results. Using this labelling, we achieved a near perfect classification in all of the open world settings we evaluated, achieving an area under curve (AUC) of more than 99% in all cases, meaning that there is minimal confusion between these two groups.

Table III: Average precision and recall obtained by in-browser memorygrammer for LSTM model in open-world setting.

Operating System	Browser	Precision	Recall
Linux	Firefox 59	87.1±0.3	84.8±0.4
Linux	Chrome 64	94.8±1.4	94.0±1.3
Windows	Firefox 59	92.9±0.4	91.9±4.3
Windows	Chrome 64	91.7±2.4	88.5±0.4
Mac OS	Safari 11.1	80.0±0.5	77.3±0.6
Linux	Tor Browser 7.5	57.8±0.3	55.7±3.6

Another set of metrics we can use are the average precision and recall our classifiers achieve across all 101 classes. Precision for a class is defined as $\frac{TP}{TP+FP}$, where TP is the number of *true positives*, i.e. the number of traces of the class for which the classifier correctly detects the class, and FP is the number of *false positives*, i.e. the number of traces of other classes that the classifier claims belong to the measured class. Recall is defined as $\frac{TP}{TP+FN}$, with FN being the number of *false negatives*—traces of the class

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.2988369, IEEE Transactions on Dependable and Secure Computing

that the classifier misclassifies. We calculate the precision and the recall for each class separately and report the simple, unweighted average, to avoid bias toward the majority class. Table III shows the results for the LSTM classifier, which in our experiments performs better than others. As shown in the table, the classifier achieves recall rates of 77–94% and precisions of 80–95% for mainstream browsers. For the Tor Browser, the precision and recall are 56% and 58% respectively, slightly worse than for mainstream browsers, but still significantly better than the base rate.

VII. Robustness Tests

We now turn our attention to the robustness of our website fingerprinting technique and test its resilience to issues known to affect network-based fingerprinting.

A. Evaluation Setup

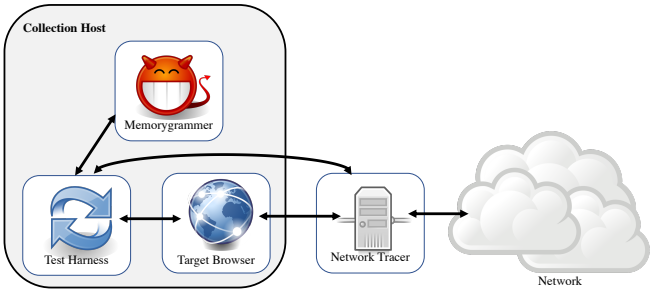


Figure 5: Data Collection Setup for the Robustness Tests.

To compare the results of network fingerprinting with cache-based fingerprinting, we need to modify our data collection setup. The setup, illustrated in Figure 5, consists of two data collection hosts. The *memorygram collection host*, which simulates the victim's machine, runs both the target browser and the memorygrammer software. The *network tracer* sits on-path between the memorygram collection host and the Internet, and collects a record of the network traffic. A test harness written in Perl and Python invokes the memorygrammer, the network tracer and the target browser at the same time, then saves a correlated data record consisting of the memorygram, the network trace in `pcap` format, and a screenshot of the target web page for monitoring purposes. For data collection, we use HP Elite 8300 desktop computers featuring Intel Core i5-2500 CPUs at 3.30 GHz, with a 6 MB last-level cache, running CentOS 7.2.1511 and either Firefox 59 or Tor Browser 7.5.

For the robustness tests we use a native-code memorygrammer, which is based on the Prime+Probe implementation of Mastik, a side-channel toolkit released under the GNU Public License [111]. We apply two modifications to the Mastik code. First, we change the Prime+Probe code to measure cache occupancy rather than activity in specific cache sets. Secondly, we use the processor's performance counters [46] to count the number of cache evictions rather than use the high resolution timer to identify evictions. The use of performance counters for attack purposes has already been proposed and investigated in the past [8, 11, 59, 99]. Every dataset of the following scenarios contains 100 traces for each of the 100 URLs in a closed-world setting, with memorygram traces and associated network traces for comparison.

B. Baseline Scenario

Our baseline scenario replicates the results of our closed-world JavaScript memorygrammer, as well as some of the results of Rimmer et al. [83]. As we can see in Table IV, the native-code memorygrammer gives a slightly better accuracy than the JavaScript memorygrammer on Firefox. When attacking the Tor Browser, the native-code memorygrammer achieves much better results than the in-browser JavaScript code. We believe that the cause of the improvement is the higher probing accuracy afforded by the native-code memorygrammer. In both browsers, we achieve similar results to those achievable with network-based fingerprinting.

C. Enabling the Response Cache

Network-based fingerprinting methods, by definition, must rely on network traffic to perform classification. Typically, due to caching, many web pages are loaded with partial or no network traffic. As specified in RFC 7234 [28], the performance of web browsers is typically improved by the use of response caches. When a web browser client requests a remote resource from a web server, the server can specify that a particular response is cacheable, and the web browser can then store this response locally, either on disk or in memory. When the page is next requested, the web browser can ask the server to send the response only if it has been modified since the last time it was accessed by the client. In the case of a response cache hit, the server only returns a short header instead of the complete remote resource, resulting in a very short network traffic sequence. In some cases, the client can even reuse the cached response without querying the server for a remote copy, resulting in no network traffic at all. Herrmann et al. [41] demonstrate a significant decrease in the accuracy of web fingerprinting when the browser uses the response cache. Indeed, deleting or disabling the browser cache prior to fingerprinting attacks is a common practice [76, 103].

We enable caching of page contents by the browser, and measure the effect on fingerprinting accuracy. In the Firefox browser we simply refrain from clearing the response cache between sessions. For privacy reasons, the response cache in the Tor Browser does not persist across session restarts. Hence, when collecting data on the Tor Browser we "prime" the cache before every recording by opening the web page in another tab, allowing it to load for 15 seconds, then closing the tab.

When we keep the browser's response cache, the advantage of cache-based website fingerprinting emerges. As Table IV shows, the accuracy of the standard network-based methods degrades when caching is enabled. We can see a degradation in accuracy of over 20% in the fingerprinting accuracy. In contrast, the cache-based methods are largely unaffected by the reduction in network traffic, achieving high accuracy rates. This result supports the conclusion that the cache-based detection methods are not simply detecting the CPU activity related to the handling of network traffic, making them

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.2988369, IEEE Transactions on Dependable and Secure Computing

Table IV: Accuracy obtained in robustness tests — Mean (percents) and Standard deviation.

	Firefox Network		**Firefox Cache**		Tor Network		**Tor Cache**	
Test	CNN	LSTM	**CNN**	**LSTM**	CNN	LSTM	**CNN**	**LSTM**
Baseline	86.4±1.0	93.2±0.5	**94.9±0.5**	**94.8±0.5**	77.6±1.6	90.9±0.7	**72.7±0.7**	**80.4±0.5**
Response cache enabled	56.1±1.5	70.6±1.5	**92.2±0.8**	**92.2±0.5**	55.5±1.7	65.9±1.0	**86.1±0.5**	**86.3±0.6**
Render only	–	–	–	–	1.0±0.0	1.0±0.0	**63.3±1.1**	**63.9±1.5**
Network only	–	–	–	–	77.6±1.6	90.9±0.7	**19.9±1.8**	**51.9±2.7**
Concept drift	–	–	–	–	64.5±2.2	81.0±0.6	**68.3±0.5**	**75.6±0.7**

essentially a special case of network-based classifiers, but are rather detecting rendering activities of the browser process.

D. Net-only and Render-only Results

Oren et al. [74] show that cache activity is correlated with network activity, suggesting that cache-based fingerprinting identifies the level of network activity. To rule out this possibility and show that website rendering also contributes to fingerprinting, we separate rendering (or more precisely, data processing) activity from handling of network data.

Render-Only Fingerprinting. To capture the data processing activity, we neutralize the network activity by guaranteeing constant traffic levels. More specifically, we apply molding to the network traffic, ensuring that data flow between the collection host and the network at a fixed bandwidth of 10 KB every 250 ms. To achieve that, we queue data transmitted at a higher rate, or send dummy packets when the transmitted data does not fill the desired bandwidth. These dummy packets are silently dropped by the receiver. The approach is, basically, BuFLO [25], with $\tau = \infty$, i.e., when the data stream continues indefinitely. This approach has a high bandwidth overhead compared to WTF-PAD and WT, however, it is designed to ensure that the network traffic is constant irrespective of the contents of the website. As expected, the raw network captures in this scenario all have the exact same size, which happens to be twice as large as the largest network capture recorded without traffic molding.

Because all the traces are identical, the network-based classifier assigns the same class to all of the traces, and its accuracy is the same as a random guess. The results of cache-based fingerprinting show a drop in accuracy compared with unmolded traffic. However, the accuracy is still significantly better than a random guess. This experiment demonstrates the resilience of cache-based website fingerprinting to mitigation techniques aimed at network-based fingerprinting, and suggests that this privacy threat may require different mitigation techniques, as we explore further in Section IX.

Network-Only Fingerprinting. In a complementing experiment, we aim to capture only the network traffic. To collect this dataset, we first capture traffic data from a real browsing session. We then use a mock setup, that does not involve a browser at all. Instead, we use two `tcpreplay` [1] instances, one at the collection host, and the other at a server, to emulate the network traffic, replaying the data from the `pcap` file.

We find that the cache-based classifier can classify many pages even in the absence of rendering activity. However, the accuracy is significantly lower than in the case that rendering activity does take place. In particular, our CNN classifier only detects the correct website in about 20% of the cases, significantly lower than the 73% we get for the matching closed-world scenario, but still much better than the 1% expected for a random guess. The accuracy of the network-based classifier is the same as for the baseline, simply because the network traffic is replicated.

Combining these two experiments, we conclude that cache-based fingerprinting identifies features both in the network traffic patterns and in the *contents* of the displayed web pages.

VIII. Real World Practicality

Previous sections show that cache-based website fingerprinting attacks can have a high accuracy. However, these attacks are carried out in a lab environment, and may not achieve the same success in a real-world environment, where the attacker does not know the victim's machine configuration or browser version, or where some time has passed between the training phase and the attack phase. In this section we investigate the feasibility of our attack in these more realistic scenarios. Specifically, we look at the effects and implications of concept drift between training and testing, unknown browser, and unknown cache size.

A. Effect of Concept Drift

Juárez et al. [51] note that the accuracy of network-based website fingerprinting declines as time passes between training set collection and the collection of data for performing the website fingerprinting attack. They attribute the decline both to changes to the websites and changes to the version of browser the users use. We now proceed to evaluate the effects of concept drift on the accuracy of our cache-based attack, including both causes of change.

Methodology: We apply the methodology of Section IV to collect multiple closed-world datasets. During a period of 20 weeks, we collect weekly datasets, each dataset containing 100 traces for every 100 closed-world website. Our memorygram collection platform features an Intel Core i5-3470 processor with 6 MB LLC, running CentOS 7.6 operating system. In the first 13 weeks we use Firefox version 60.7. After 13 weeks, we upgrade to version 60.8 using the `yum` system update utility.

We use five of the datasets, collected four weeks apart of each other, in weeks 2, 6, 10, 14, and 18, to train five models, one for each of the datasets. The training uses 10-fold cross validation on the traces from the dataset. We then test how well each of the five models classifies the traces in each of the 20 datasets collected for the experiment.

Results: Figure 6 shows the results of our experiment. Each of the five lines shows the accuracy of one of the models. The

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.2988369, IEEE Transactions on Dependable and Secure Computing

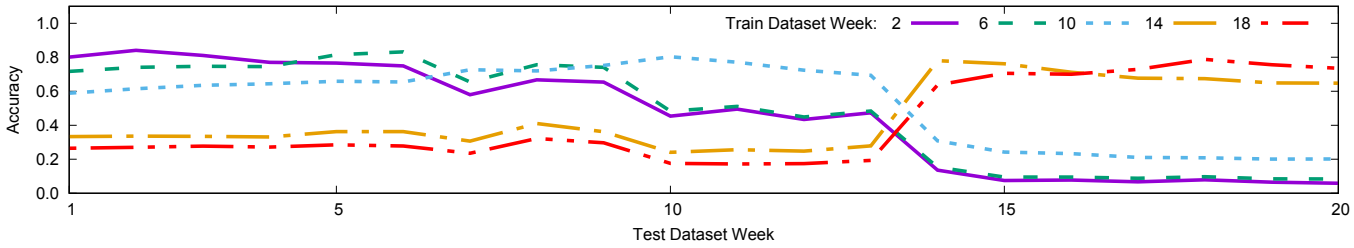


Figure 6: Concept Drift – Effects of time difference between collecting training and test data.

horizontal axis shows the week the test dataset was collected at and the vertical axis is the model's classification accuracy.

As expected, each model achieves its best performance at the week in which it is collected. We further see that, with the exception of week 13, the decline in model accuracy is quite moderate. For example, the model collected in week 2 still achieves over 47% in week 13. (Compared with 84% in week 2 and 1% base rate). This compares well with the results of Juárez et al. [51], who note with network-based fingerprinting the accuracy drops to under 50% within less than ten days. However, the browser upgrade in week 13 has a significant impact on the accuracy of the models. Models based on data collected before the change do poorly on data collected after the change and vice versa. Although we used the Firefox browser as a case study, the concept drift of data traces is influenced by both webpage changes and browser updates. Therefore, we expect the problem to also appear in Google Chrome and in the Tor Browser.

B. Cross-Browser Fingerprinting

Different browsers use different algorithms for downloading and presenting pages. As such, we expect different browsers to produce different signatures for the same website. At the same time, for attack efficiency, it is desired to minimize the number of models created for different scenarios. One option is to create a single training dataset which would learn over all possible types of browser software. We now investigate the relationship between the browsers and the attack efficiency.

Table V: System Setup for the Cross-Browser Experiment.

Browser	OS	CPU	LLC
Firefox 60.8	Linux	i5-2500	6 MB
Chrome 77	Windows	i5-3470	6 MB
Safari 11.1	Mac OS	i7-6700	8 MB

Methodology: We test the sensitivity of classifiers to the browser used for collecting the training sets. We further test whether a single classifier can correctly identify websites irrespectively of the browser used by the victim. We collect traces on three hosts, summarized in Table V. On each host, we collect our closed-world dataset and use the data to build four models. Three of the models are trained on a dataset collected on one of the hosts. The fourth model is trained with the combined data of all three datasets. We use 10-fold cross validation for the evaluation.

Results: The experiment shows the importance of matching the browser used for collecting the training data to the target browser. Figure 7 shows the accuracy of the four models we trained when tested against the data from each browser, and against the combined dataset. The vertical axis specifies the browser used for collecting the training dataset, while the horizontal axis is the browser used for collecting the testing dataset. Accuracy is presented both numerically and by using darker shades for higher accuracy.

The results show that training on a single browser only allows fingerprinting on the same browser. With cross-browser classification, the results are close to random, achieving the base rate accuracy of 1%. Nonetheless, with a training set that includes data collected on all browsers, the accuracy is almost as high as with training on each specific browser. Thus, one model can suffice for cross-browser classification, provided that the model is trained with data from all browsers.

C. Effect of Cache Size Misestimation

The cache occupancy attack we use assumes we know the target computer's cache size. Presumably, using too small a buffer might fail to force cache contention, whereas too large a buffer would cause evictions regardless of the victim's activity. To validate this assumption, we measure the effect of an incorrectly estimated last-level cache size on fingerprinting accuracy, by creating datasets in which the buffer size differs from the actual cache size.

Methodology: We collect traces with incorrectly estimated cache sizes. We use a host with an Intel Core i5-3470 processor, with a 6 MB last-level cache, running CentOS 7.6 and Firefox 60.8. We collect three datasets, each with a different "guess" of a cache size, reflected in the size of the buffer we use for the cache occupancy attack. One guess is the correct cache size of 6 MB. The other guesses are a smaller and a larger cache (4 MB and 8 MB). We train an LSTM model on each of these datasets, using a 10-fold cross validation with 90% of the traces used for training and 10% for testing. We evaluate the models against each of the collected datasets.

Results: Figure 8 shows that as long as we use the same buffer size for both the training and the test sets, the accuracy of the classifier is high. However, if the model is trained with one estimation and tested with a different estimation, the results are close to a random guess. Surprisingly, correctly guessing the cache size is less important than matching the guess between the training and testing sets. That is, the cache

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.2988369, IEEE Transactions on Dependable and Secure Computing

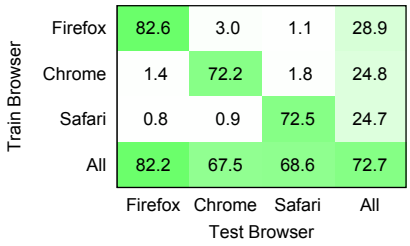


Figure 7: Classifier accuracy with different browser combinations.

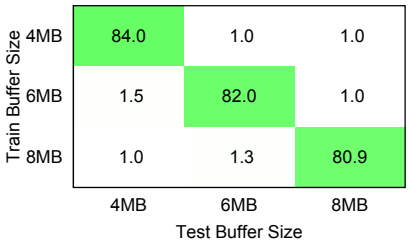


Figure 8: Cache size misestimation classifiers performance.

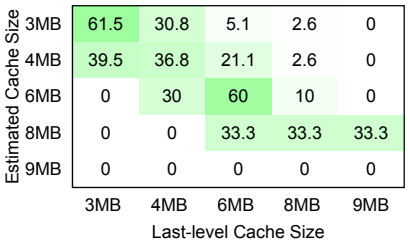


Figure 9: Real-world accuracy of the

occupancy attack is not too sensitive and works well with wrong estimations of the cache size.

D. Cache Size Estimation

In this section we evaluate the adversary's ability to correctly estimate the size of the cache. We start with a lab experiment on machines under our control and follow with a real-world experiment on users' machines.

Initial Lab Experiment: In our initial lab experiment [88], we created a JavaScript program that allocates a 20 MB array in memory and iterates over it in several patterns which should fit in well into different configurations of cache set-counts and associativities. We then recorded the minimum, maximum and mean access time per element, plus the standard deviation, for each of these configurations. We collected 1,350 such measurements from multiple systems with cache sizes of 3 MB, 4 MB, 6 MB, and 8 MB. We then used MATLAB's classification learner tool to apply a variety of machine learning classifiers to the measured data. Using both KNN and SVM classifiers, we were able to correctly classify the configuration of the target's last-level cache with over 99.8% classification accuracy under 5-fold cross validation. Interestingly, even a simple tree-based classifier which compared the minimum iteration time of three different configurations to a predefined threshold was 99.6% accurate. We ported this simple tree-based classifier to JavaScript, creating an LLC cache size detector which we tested and found capable of accurately detecting the cache sizes of 15 different machines with diverse browser, hardware and operating system configurations, taking less than 300 ms to run in all cases. We thus concluded that generic attacks that adapt to the specific hardware configuration seems feasible.

Real-world Cache Size Detection: For our real-world experiment we set up a custom-designed man-in-the-middle (MITM) environment that injects the cache-size estimation code to the users' browsers. In this setup, shown in Figure 10, users connect to the Internet via a wireless access point. Traffic between the access point and the Internet is filtered by a MITM server, implemented as an Internet Content Adaptation Protocol (ICAP) [26] in a Squid-Cache proxy server [96]. The MITM server monitors access to non-encrypted websites, and injects the JavaScript code that performs the cache size estimation to the accessed pages.

Experiment: We conducted the experiment during an undergraduate programming "hackathon". Prior to the experiment,

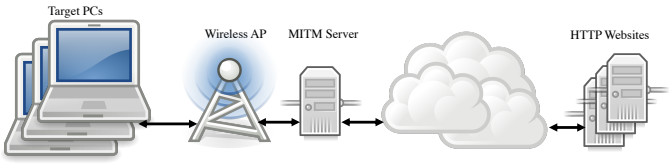


Figure 10: Physical setup of the real-world experiment.

participating students provided the ground-truth hardware configuration of their computers, including the MAC addresses, used for identifying the computer, and the last-level cache size. Because most traffic the students accessed was encrypted, we ended up asking participants to also visit the non-encrypted website http://askcom.me, to ensure we can inject the code.

Ethical Considerations: The experiment design allows us access to participants' web browsing activity. To address potential ethical issues, we made sure to limit the amount of information we record. Specifically, we recorded the URLs of accessed websites, but not their contents. We kept track of the MAC address of the participants' computers, but did not store any personally identifying information that can link specific participants to their computers. Finally, sensitive websites, such as health or banking websites, were not likely to be intercepted, because these are likely to be encrypted.

Participation in the experiment was voluntary. Students were briefed about the procedure and the implications of the experiment, were asked to provide written consent before participating, and were given the option not to participate. Participating students received one bonus point in the final grade of an undergraduate course. Prior to conducting the experiment we sought and received approval from the Ben-Gurion University's Institutional Review Board (IRB).

Table VI: Machine Configurations for Real-World Experiment

Property	Value
Operating System	Windows: 82, Linux: 1, Mac OS: 6, Android: 1
Browser	Chrome: 81, Edge: 1, Firefox: 3, Safari: 1, Unknown: 4
CPU Generation	Gen 2 (Sandy Bridge) to Gen 5 (Broadwell): 21, Gen 6 (Skylake): 42, Gen 7 (Kaby Lake): 18, Gen 8 (Coffee Lake): 9
Last-Level Cache	3 MB: 39, 4 MB: 29, 6 MB: 17, 8 MB: 4, 9 MB: 1

Results: Table VI summarizes the hardware and software configurations of the participating computers. The vast majority of the participants used Chrome browser on Windows, and featured a wide diversity of Intel CPU micro-architectures,

spanning from Generation 2 (Ivy Bridge) all the way to Generation 8 (Coffee Lake).

Figure 9 shows the data segmentation by the cache sizes. From this figure, we can understand besides the accuracy, the true positive rate and the false positive rate of each classification. The x-axis shows the ground truth of the LLC size, as collected from the participants. The y-axis shows the possible classification results from the injected code. The data inside the confusion matrix shows us the probability to get each estimate given the ground-truth size. As the figure shows, the real-world performance of the cache size detection code is considerably worse than under ideal conditions, but it still performs much better than a random guess. We conjecture that the lower accuracy may stem from the difference between the training setup (a standalone web page in the lab setup) and the testing setup (a MITM injected script in the real-world experiment). The accuracy may be increased by training the classifier under more realistic conditions, or by extending the testing time beyond 300 milliseconds.

IX. Countermeasures

We now discuss potential countermeasures to our fingerprinting attack. We first describe a cache masking technique we experimented with. We then follow with a review of other cache attack countermeasures suggested in the literature.

A. Cache Activity Masking

A well-studied mitigation approach from the domain of network-based cache fingerprinting involves creating spurious network activity to mask the actual website traffic [25]. It is possible to adapt this technique to our domain and create activity in the cache to mask the website rendering activity. Our initial experiments show some promise, but further research is needed to assess its effectiveness and its effect on performance and on power consumption.

Masking implementation. Our countermeasure repeatedly evicts the entire last-level cache. More specifically, we allocate a cache-sized buffer and access every cache line in the buffer in a loop. Such masking could be applied in the browser, in the operating system, as a browser plugin, and even incorporated into a security-conscious website in the form of JavaScript delivered to the client. For our initial proof of concept implementation we use a native code application, based on Mastik [111]. This setting allows us to investigate the effectiveness of our countermeasure while leaving deployment complexities for future work.

Evaluation. For evaluation, we use a computer featuring an Intel Core i5-2500, running Centos Linux version 7.6.1810. We enable the countermeasure, then collect website traces both for Firefox (Linux) and for the Tor Browser, using the same mix of traces described in Section IV-B—100 traces of each of 100 websites for the closed-world scenario and one additional trace of each of 5,000 websites for the open-world scenario. As in Section V-B, we use 10-fold cross validation for building and evaluating the models.

We find that the countermeasure completely thwarts the attack when training is done on an unprotected system—the accuracy of our classifier is at or below the base rate of 1% for the closed-world scenario and 33% for the open-world scenario. We also evaluate a scenario in which the adversary is allowed to train on traces with the countermeasure applied. In this more challenging scenario, the countermeasure completely thwarts the attack when the attack code is running from the Tor Browser. On Firefox, however, we only notice a moderate reduction in the effectiveness of the attack. In the closed-world scenario, the attack achieves an accuracy of 73%, and in the open-world, 77%. (Down from 79% and 86%, respectively.)

Performance Impact. To understand the effect that our countermeasure has system performance, we use the industry-standard SPEC CPU benchmark [93], the de-facto standard benchmark for measuring the performance of the CPU and the memory subsystems. Figure 11 shows the results of the SPEC CPU 2006 benchmarks with our countermeasure, relative to no countermeasure. The countermeasure causes a slowdown of around 5% (geometric mean across the benchmarks) with a worst case slowdown of 14% for the bwaves benchmark. These results are from the average of ten executions of the benchmarks for each case. With Tor network performance being as it is, we believe that the performance hit on CPU benchmarks is acceptable for this scenario.

B. Other Countermeasures

Most of the past research into cache attacks has been done in the context of side-channel cryptanalysis. Due to the different scenario, many of the countermeasures typically suggested for cache-based attack are no longer effective. Techniques such as constant-time programming [6] are only applicable to regular code, typically found in implementations of cryptographic primitives. It is hard to see how such techniques can be applied to web browsers. Similarly, as we show, timer-based defenses that reduce the timer frequency or add jitter are not effective.

Cache randomization techniques [66, 80, 106] dissociate victim and adversary cache sets, and prevent the adversary from monitoring victim access to specific addresses. However, our attack measures the overall cache activity rather than looking at specific victim accesses. As such, such techniques are unlikely to be effective against our attack.

Cache partitioning, either using dedicated hardware [24, 106] or via page coloring [63], is a promising approach for mitigating cache attacks. In a nutshell, the approach partitions the cache between security domains, preventing cross-domain contention. Web pages are often rendered within the same browser process. A page-coloring countermeasure will, therefore, need to adapt to the browser scenario. Alternatively, the current shift to strict site isolation [95] as part of the mitigations for Spectre [54], may assist in applying page coloring to protect against our attack. A further limitation of page coloring is that caches support only a handful of colors. Hence, colors need to be shared, particularly when a large number of tabs are open. To provide protection, page coloring will have to be augmented with a solution that prevents concurrent use of the same color by multiple sites.

CacheBar [113] limits the contention caused by each process as a protection for the Prime+Probe attack. Like cache

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.2988369, IEEE Transactions on Dependable and Secure Computing

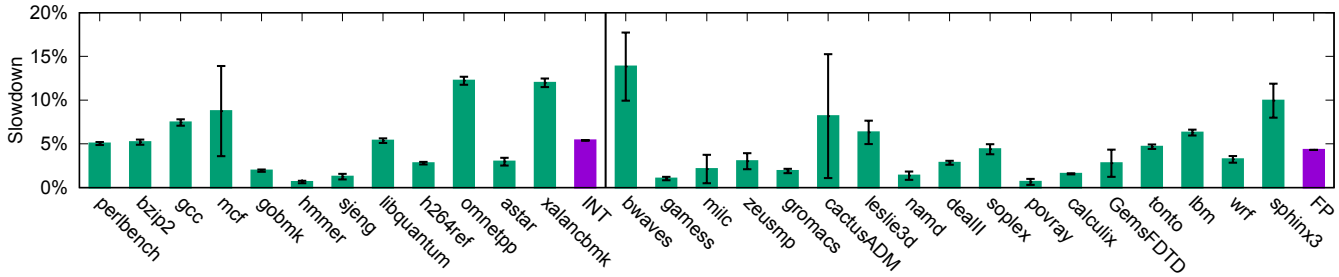


Figure 11: Performance slowdown of our countermeasure on the SPEC benchmark. Error bars indicate one standard deviation. INT and FP show the geometric mean of the SPEC integer and floating point benchmarks, respectively.

partitioning, this approach works at a process resolution and may require adaptions to work in the web browser scenario. Furthermore, unlike past cryptographic attacks that aim to identify specific memory accesses, our technique measures the overall memory use of the victim. Consequently, unless CACHEBAR is configured to partition the cache, some cross-process contention will remain, allowing our attack to work.

X. LIMITATIONS AND FUTURE WORK

Although we demonstrate the feasibility of cache-based website fingerprinting and provide an analysis of the attack, we do leave some areas for further study. Being the first analysis of its kind, the scope of the work does not match similar works on network-based website fingerprinting. In particular, our datasets are significantly smaller than those of Rimmer et al. [83], for example. Providing larger datasets would allow better analysis of the effectiveness of the technique.

For most of our experiments we use identical machine configurations for collecting the training and test datasets. Some of our results, in particular in sections VIII-B and VIII-C, show the potential for using a single classifier that can effectively classify memorygrams collected on multiple configurations. It would also be interesting to improve the accuracy of our cache size detection script, perhaps by training it under more realistic conditions. Similarly, it would be interesting to see whether the classifier can genuinely find commonalities between multiple browsers rendering the same website, or more generally whether a classifier can perform *app classification* and detect which browser is being used to browse to a previously unknown website.

This work further shares many of the limitations of network-based fingerprinting [51]. In particular, websites tend to change over time or based on the identity of the user or the specifications of the computer used for displaying them. Furthermore, our work, like most previous works, assumes that only one website is displayed at each time. Rimmer et al. [83] briefly discuss temporal aspects of website fingerprinting and our work further investigates concept drift over a 20 week period. A followup to our results would be a direct comparison between the concept drift of network-based fingerprinting and cache-based fingerprinting. We believe that while the content elements of a website, such as images and text, may change quickly, the general structure of a website changes much slower. Hence, cache-based traces may be better able to capture this structure and therefore be less sensitive to concept drift. Another followup would be a design of drift-resistant classifiers which can obtain good accuracy results with minimum maintenance over time.

XI. AVAILABILITY

To allow reproduction of our results, we published several of the JavaScript datasets used in this work on the IEEE DataPort website [89]. The `linux_chrome`,`linux_ff59` and `linux_tor` are traces that collected on a Linux machine, using the browsers Chrome, Firefox and Tor correspondingly. The directories `win_chrome` and `win_ff59` are with data collected on Windows 10, using the browsers Chrome and Firefox respectively. Finally, the directory `mac_safari` is data collected on MacOs with the Safari browser. The experiment of the countermeasure is in a folder named `linux_tor_counter`. These directories have subdirectories CW and OW, for the closed-world and open-world scenarios. The closed-world files include up to 100 traces per website, whereas the open-world files contain one trace per each website. All of the data files are in JSON format.

The implementation of the JavaScript memorygrammer is available online at (https://codepen.io/atoliks24/pen/GRRPzQm).

XII. CONCLUSIONS

In this work we investigate the use of cache side channels for website fingerprinting. We implement two memorygrammers, which capture the cache activity of the browser, and show how to use deep learning to identify websites based on the cache activity that displaying them induces.

We show that cache-based website fingerprinting achieves results comparable with the state-of-the-art network-based fingerprinting. We further show that cache-based fingerprinting outperforms network-based fingerprinting when the browser caches objects. Finally, we demonstrate that cache-based fingerprinting is resilient to both traffic molding and to reduced timer resolution. The former being the standard defense for network-based website fingerprinting and the latter the currently implemented countermeasure for mobile-code-based micro-architectural attacks. To the best of our knowledge, this is the first cache-based side-channel attack that works with the 100 ms clock rate of the Tor Browser.

We carried out a real-world evaluation of our attack on a set of computers with diverse hardware and software configurations. Our results show that, while the accuracy of the attack is severely degraded when the precise hardware and software configuration of the victim is not known beforehand, it is still significantly higher than the base rate accuracy of a random guess. Surprisingly, mispredicting the LLC cache size of the victim's computer had only a minor impact on the accuracy of the website fingerprinting attack, as long as the training and testing steps were carried out under the same assumption.

Acknowledgements

We would like to thank Vera Rimmer for her helpful comments and insights. We would also like to thank Roger Dingledine and our shepherd Rob Jansen for reviewing and commenting on the final version of the conference paper.

This research was supported by the ARC Centre of Excellence for Mathematical & Statistical Frontiers, an ARC Discovery Early Career Researcher Award DE200101577, Intel Corporation, Israel Science Foundation grants 702/16 and 703/16, NSF CNS-1409415, and NSF CNS-1704105.

References

[1] "Tcpreplay," https://tcpreplay.appneta.com/.
[2] K. Abe and S. Goto, "Fingerprinting attack on Tor anonymity using deep learning," in *APAN*, 2016.
[3] O. Acıiçmez, "Yet another microarchitectural attack: : exploiting I-Cache," in *CSAW*, 2007.
[4] O. Acıiçmez, B. B. Brumley, and P. Grabher, "New results on instruction cache attacks," in *CHES*, 2010.
[5] K. Al-Naami, A. El Ghamry, M. S. Islam, L. Khan, B. M. Thuraisingham, K. W. Hamlen, M. Alrahmawy, and M. Rashad, "BiMorphing: A bi-directional bursting defense against website fingerprinting attacks," *IEEE Transactions on Dependable and Secure Computing*, 2019.
[6] D. J. Bernstein, T. Lange, and P. Schwabe, "The security impact of a new cryptographic library," in *LATINCRYPT*, 2012.
[7] S. Bhat, D. Lu, A. Kwon, and S. Devadas, "Var-CNN: A data-efficient website fingerprinting attack based on deep learning," *PoPETs*, vol. 2019, no. 4, pp. 292–310, 2019.
[8] S. Bhattacharya and D. Mukhopadhyay, "Who watches the watchmen?: Utilizing performance monitors for compromising keys of RSA on Intel platforms," in *CHES*, 2015.
[9] Z. Bloom, "Cloud computing without containers," https://blog.cloudflare.com/cloud-computing-without-containers/, 2018.
[10] J. M. Booth, "Not so incognito: Exploiting resource-based side channels in JavaScript engines," Bachelor Thesis, Harvard, April 2015.
[11] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A. Sadeghi, "Software grand exposure: SGX cache attacks are practical," in *WOOT*, 2017.
[12] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a distance: website fingerprinting attacks and defenses," in *CCS*, 2012.
[13] X. Cai, R. Nithyanand, and R. Johnson, "Cs-buflo: A congestion sensitive website fingerprinting defense," in *WPES*, 2014.
[14] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg, "A systematic approach to developing and evaluating website fingerprinting defenses," in *CCS*, 2014.
[15] A. Caliskan-Islam, R. Harang, A. Liu, A. Narayanan, C. Voss, F. Yamaguchi, and R. Greenstadt, "De-anonymizing programmers via code stylometry," in *USENIX Sec*, 2015.
[16] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *KDD*, 2016.
[17] H. Cheng and R. Avnur, "Traffic analysis of SSL encrypted web browsing," Project paper, University of Berkeley, 1998.
[18] G. Cherubin, J. Hayes, and M. Juárez, "Website fingerprinting defenses at the application layer," *PoPETs*, vol. 2017, no. 2, pp. 186–203, 2017.
[19] S. S. Clark, H. A. Mustafa, B. Ransford, J. Sorber, K. Fu, and W. Xu, "Current events: Identifying webpages by tapping the electrical outlet," in *ESORICS*, 2013.
[20] D. Cock, Q. Ge, T. C. Murray, and G. Heiser, "The last mile: An empirical study of timing channels on seL4," in *CCS*, 2014.
[21] W. Dai, "PipeNet description," Post to the `cypherpunks` mailing list. https://www.freehaven.net/anonbib/cache/pipenet10.html, 1998.
[22] T. Dierks and E. Rescola, "The transport layer security (TLS) protocol version 1.2," Internet Requests for Comments, RFC 5246, 2008.
[23] R. Dingledine, N. Mathewson, and P. F. Syverson, "Tor: The second-generation onion router," in *USENIX Sec*, 2004.
[24] L. Domnitser, A. Jaleel, J. Loew, N. B. Abu-Ghazaleh, and D. Ponomarev, "Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks," *TACO*, vol. 8, no. 4, pp. 35:1–35:21, 2012.
[25] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-Boo, I still see you: Why efficient traffic analysis countermeasures fail," in *IEEE SP*, 2012.
[26] J. Elson and A. Cerpa, "Internet content adaptation protocol (icap)," Internet Requests for Comments, RFC Editor, RFC 3507, April 2003.
[27] D. Evtyushkin, D. V. Ponomarev, and N. B. Abu-Ghazaleh, "Jump over ASLR: attacking branch predictors to bypass ASLR," in *MICRO*, 2016.
[28] R. Fielding, M. Nottingham, and J. Reschke, "Hypertext transfer protocol (HTTP/1.1): Caching," Internet Requests for Comments, RFC Editor, RFC 7234, June 2014, http://www.rfc-editor.org/rfc/rfc7234.txt.
[29] P. Frigo, C. Giuffrida, H. Bos, and K. Razavi, "Grand pwning unit: Accelerating microarchitectural attacks with the GPU," in *IEEE SP*, 2018.
[30] C. P. García, B. B. Brumley, and Y. Yarom, ""Make sure DSA signing exponentiations really are constant-time"," in *CCS*, 2016.
[31] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," *J. Cryptographic Engineering*, vol. 8, no. 1, pp. 1–27, 2018.
[32] D. Genkin, L. Pachmanov, E. Tromer, and Y. Yarom, "Drive-by key-extraction cache attacks from portable code," in *ACNS*, 2018.
[33] X. Gong, N. Borisov, N. Kiyavash, and N. Schear, "Website detection using remote traffic analysis," in *PET*, 2012.
[34] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning (Adaptive Computation and Machine Learning series)*. The MIT Press, 2016.
[35] B. Gras, K. Razavi, E. Bosman, H. Bos, and C. Giuffrida, "ASLR on the line: Practical cache attacks on the MMU," in *NDSS*, 2017.
[36] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches," in *USENIX Sec*, 2015.
[37] D. Gruss, C. Maurice, A. Fogh, M. Lipp, and S. Mangard, "Prefetch side-channel attacks: Bypassing SMAP and kernel ASLR," in *CCS*, 2016.
[38] B. Gülmezoglu, A. Zankl, T. Eisenbarth, and B. Sunar, "PerfWeb: How to violate web privacy with hardware performance events," in *ESORICS (2)*, 2017.
[39] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *USENIX Sec*, 2016.
[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.
[41] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier," in *CCSW*, 2009.
[42] A. Hintz, "Fingerprinting websites using traffic analysis," in *Privacy Enhancing Technologies*, 2002.
[43] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[44] W. Hu, "Lattice scheduling and covert channels," in *IEEE SP*, 1992.
[45] M. S. Inci, B. Gülmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cache attacks enable bulk key recovery on the cloud," in *CHES*, 2016.
[46] Intel Corp., "Intel 64 and IA-32 architectures software developer's manual volume 3B," Sep. 2016. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf
[47] ——, "Intel 64 and IA-32 architectures optimization reference manual," https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html, Jun. 2016.
[48] G. Irazoqui Apecechea, T. Eisenbarth, and B. Sunar, "S$A: A shared cache attack that works across cores and defies VM sandboxing - and its application to AES," in *IEEE SP*, 2015.
[49] S. Jana and V. Shmatikov, "Memento: Learning secrets from process footprints," in *IEEE SP*, 2012.
[50] R. Jansen, M. Juárez, R. Galvez, T. Elahi, and C. Díaz, "Inside job: Applying traffic analysis to measure Tor from within," in *NDSS*, 2018.
[51] M. Juárez, S. Afroz, G. Acar, C. Díaz, and R. Greenstadt, "A critical evaluation of website fingerprinting attacks," in *CCS*, 2014.

[52] M. Juárez, M. Imani, M. Perry, C. Díaz, and M. Wright, "Toward an efficient website fingerprinting defense," in *ESORICS (1)*, 2016.

[53] H. Kim, S. Lee, and J. Kim, "Inferring browser activity and status through remote monitoring of storage usage," in *ACSAC*, 2016.

[54] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Haburg, M. Lipp, S. Mangard, T. Prescher, M. Schwartz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *IEEE SP*, May 2019.

[55] D. Kohlbrenner and H. Shacham, "Trusted browsers for uncertain times," in *USENIX Sec*, 2016.

[56] N. Köskal, "'Terrifying': How a single line of computer code put thousands of innocent Turks in jail," http://www.cbc.ca/news/world/terrifying-how-a-single-line-of-computer-code-put-thousands-of-innocent-turks-in-jail-1.4495021, Jan. 2018.

[57] A. Krogh, M. Brown, I. S. Mian, K. Sjolander, and D. Haussler, "Hidden Markov models in computational biology. Applications to protein modeling," *Journal of Molecular Biology*, vol. 235, no. 5, pp. 1501–1531, 1994.

[58] S. Lee, Y. Kim, J. Kim, and J. Kim, "Stealing webpages rendered on your browser by exploiting GPU vulnerabilities," in *IEEE SP*, 2014.

[59] S. Lee, M. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, "Inferring fine-grained control flow inside SGX enclaves with branch shadowing," in *USENIX Sec*, 2017.

[60] F. Li, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 4, pp. 594–611, 2006.

[61] S. Li, H. Guo, and N. Hopper, "Measuring information leakage in website fingerprinting attacks and defenses," in *CCS*, 2018.

[62] B. Liang, W. You, L. Liu, W. Shi, and M. Heiderich, "Scriptless timing attacks on web browser privacy," in *DSN*, 2014.

[63] J. Liedtke, H. Härtig, and M. Hohmuth, "OS-controlled cache predictability for real-time systems," in *IEEE RTAS*, 1997.

[64] P. Lifshits, R. Forte, Y. Hoshen, M. Halpern, M. Philipose, M. Tiwari, and M. Silberstein, "Power to peep-all: Inference attacks by malicious batteries on mobile devices," *PoPETs*, vol. 2018, no. 4, pp. 1–1, 2018.

[65] M. Lipp, M. Schwartz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *USENIX Sec*, Aug. 2018.

[66] F. Liu and R. B. Lee, "Random fill cache architecture," in *MICRO*, 2014.

[67] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *IEEE SP*, 2015.

[68] X. Liu, J. Wu, and Z. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Trans. Systems, Man, and Cybernetics, Part B*, vol. 39, no. 2, pp. 539–550, 2009.

[69] L. Lu, E. Chang, and M. C. Chan, "Website fingerprinting and identification using ordered feature sequences," in *ESORICS*, 2010.

[70] N. Matyunin, Y. Wang, T. Arul, J. Szefer, and S. Katzenbeisser, "MagneticSpy: Exploiting magnetometer in mobile devices for website and application fingerprinting," arXiv:1906.11117, 2019.

[71] Mozilla Foundation, "Security advisory 2018-01," https://www.mozilla.org/en-US/security/advisories/mfsa2018-01/, 2018.

[72] A. Narayanan, H. Paskov, N. Z. Gong, J. Bethencourt, E. Stefanov, E. C. R. Shin, and D. Song, "On the feasibility of internet-scale author identification," in *IEEE SP*, 2012.

[73] R. Nithyanand, X. Cai, and R. Johnson, "Glove: A bespoke website fingerprinting defense," in *WPES*, 2014.

[74] Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis, "The spy in the sandbox: Practical cache attacks in JavaScript and their implications," in *CCS*, 2015.

[75] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of AES," in *CT-RSA*, 2006.

[76] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *WPES*, 2011.

[77] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website fingerprinting at internet scale," in *NDSS*, 2016.

[78] C. Percival, "Cache missing for fun and profit," 2005, presented at BSDCan. http://www.daemonology.net/hyperthreading-considered-harmful.

[79] F. Pizlo, "What Spectre and Meltdown mean for WebKit," https://webkit.org/blog/8048/what-spectre-and-meltdown-mean-for-webkit/, Jan. 2018.

[80] M. K. Qureshi, "CEASER: Mitigating conflict-based cache attacks via encrypted-address and remapping," in *MICRO*, 2018.

[81] M. K. Reiter and A. D. Rubin, "Crowds: Anonymity for web transactions," *ACM Trans. Inf. Syst. Secur.*, vol. 1, no. 1, pp. 66–92, 1998.

[82] E. Rescola, "HTTP over TLS," Internet Requests for Comments, RFC Editor, RFC 2818, 2000, https://tools.ietf.org/html/rfc2818.

[83] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen, "Automated website fingerprinting through deep learning," in *NDSS*, 2018.

[84] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *CCS*, 2009.

[85] J. Rutkowska and R. Wojtczuk, "Qubes OS architecture," https://www.qubes-os.org/attachment/wiki/QubesArchitecture/arch-spec-0.3.pdf, Feb. 2010.

[86] M. Schwarz, C. Maurice, D. Gruss, and S. Mangard, "Fantastic timers and where to find them: High-resolution microarchitectural attacks in JavaScript," in *Financial Cryptography*, 2017.

[87] M. Schwarz, M. Lipp, and D. Gruss, "JavaScript zero: Real JavaScript and zero side-channel attacks," in *NDSS*, 2018.

[88] A. Shusterman, L. Kang, Y. Haskal, Y. Meltser, P. Mittal, Y. Oren, and Y. Yarom, "Robust website fingerprinting through the cache occupancy channel," in *USENIX Sec*, 2019.

[89] A. Shusterman, L. Kang, Y. Haskal, Y. Meltzer, P. Mittal, Y. Oren, and Y. Yarom, "Website fingerprinting - last level cache contention traces," 2019. [Online]. Available: http://dx.doi.org/10.21227/a33s-cf63

[90] P. Sirinam, M. Imani, M. Juárez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," in *CCS*, 2018.

[91] P. Sirinam, N. Mathews, M. S. Rahman, and M. Wright, "Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning," in *CCS*, 2019.

[92] Spiegel Online, "Documents reveal top NSA hacking unit," http://www.spiegel.de/international/world/the-nsa-uses-powerful-toolbox-in-effort-to-spy-on-global-networks-a-940969-2.html, Dec. 2013.

[93] C. D. Spradling, "SPEC CPU2006 benchmark tools," *SIGARCH Computer Architecture News*, vol. 35, no. 1, pp. 130–134, 2007.

[94] R. Spreitzer, S. Griesmayr, T. Korak, and S. Mangard, "Exploiting data-usage statistics for website fingerprinting attacks on Android," in *WISEC*, 2016.

[95] The Chromium Project, "Site isolation," https://www.chromium.org/Home/chromium-security/site-isolation.

[96] The Squid Software Foundation, "The Squid Proxy," http://www.squid-cache.org.

[97] The Tor Project, Inc., "The Tor Browser," https://www.torproject.org/projects/torbrowser.html.en.

[98] Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeri, and H. Miyauchi, "Cryptanalysis of DES implemented on computers with cache," in *CHES*, 2003.

[99] L. Uhsadel, A. Georges, and I. Verbauwhede, "Exploiting hardware performance counters," in *FDTC*, 2008.

[100] K. Varda, https://news.ycombinator.com/item?id=18280156, 2018.

[101] P. Vila and B. Köpf, "Loophole: Timing attacks on shared event loops in Chrome," in *USENIX Sec*, 2017.

[102] L. Wagner, "Mitigations landing for new class of timing attack," https://blog.mozilla.org/security/2018/01/03/mitigations-landing-new-class-timing-attack/, Jan. 2018.

[103] T. Wang and I. Goldberg, "Improved website fingerprinting on Tor," in *WPES*, 2013.

[104] ——, "On realistically attacking Tor with website fingerprinting," *PoPETs*, vol. 2016, no. 4, pp. 21–36, 2016.

[105] ——, "Walkie-Talkie: An efficient defense against passive website fingerprinting attacks," in *USENIX Sec*, 2017.

[106] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in *ISCA*, 2007.

[107] Z. Weinberg, E. Y. Chen, P. R. Jayaraman, and C. Jackson, "I still know what you visited last summer: Leaking browsing history via user interaction and side channel attacks," in *IEEE SP*, 2011.

[108] Y. Xu, T. Wang, Q. Li, Q. Gong, Y. Chen, and Y. Jiang, "A multi-tab website fingerprinting attack," in *ACSAC*, 2018.

[109] J. Yan and J. Kaur, "Feature selection for website fingerprinting," *PoPETs*, vol. 2018, no. 4, pp. 200–219, 2018.

[110] Q. Yang, P. Gasti, G. Zhou, A. Farajidavar, and K. S. Balagani, "On inferring browsing activity on smartphones via USB power analysis side-channel," *IEEE Trans. Information Forensics and Security*, vol. 12, no. 5, pp. 1056–1066, 2017.

[111] Y. Yarom, "Mastik: A micro-architectural side-channel toolkit," http://cs.adelaide.edu.au/~yval/Mastik/Mastik.pdf, Sep. 2016.

[112] Y. Yarom and N. Benger, "Recovering OpenSSL ECDSA nonces using the FLUSH+RELOAD cache side-channel attack," Cryptology ePrint Archive, Report 2014/140, 2014. [Online]. Available: http://eprint.iacr.org/2014/140
[113] Z. Zhou, M. K. Reiter, and Y. Zhang, "A software approach to defeating side channels in last-level caches," in *CCS*, 2016.
[114] Z. Zhuo, Y. Zhang, Z. Zhang, X. Zhang, and J. Zhang, "Website fingerprinting attack on anonymity networks based on profile hidden markov model," *IEEE Trans. Information Forensics and Security*, vol. 13, no. 5, pp. 1081–1095, 2018.

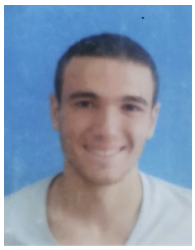
Anatoly Shusterman is a Ph.D student in the Department of Software and Information Systems Engineering in Ben-Gurion University of the Negev, Israel.

Zohar Avraham is an undergraduate student in the Department of Software and Information Systems Engineering in Ben-Gurion University of the Negev, Israel.

Eliezer Croitoru is a Linux System Engineer at Internet Rimon and a contributor to the Squid-Cache and ICAP open-source projects.

Yarden Haskal is an undergraduate student in the Department of Software and Information Systems Engineering in Ben-Gurion University of the Negev, Israel.

Lachlan Kang Lachlan Kang is a network security and internet privacy expert who has spent his research career trying to improve online anonymity by finding flaws in existing systems and patching them. His current research interests include offensive network and internet security.



Dvir Levi is an undergraduate student in the Department of Software and Information Systems Engineering in Ben-Gurion University of the Negev, Israel.

Yosef Meltser is an undergraduate student in the Department of Software and Information Systems Engineering in Ben-Gurion University of the Negev, Israel.

Prateek Mittal (SM' 17) is an Associate Professor in the Department of Electrical Engineering at Princeton University. He obtained his Ph.D. from the University of Illinois at Urbana-Champaign in 2012. He is the recipient of the NSF CAREER award (2016), ONR YIP award (2018), M.E. Van Valkenburg award, Google Faculty Research Award (2016, 2017), Cisco Faculty research award (2016), Intel Faculty research award (2016, 2017), and IBM Faculty award (2017). He was awarded Princeton University's E. Lawrence Keyes Award for outstanding research and teaching, and is the recipient of multiple outstanding paper awards including ACM CCS and ACM ASIACCS.

Yossi Oren (SM' 17) received his M.Sc. degree in Computer Science from the Weizmann Institute of Science, Israel, and his Ph.D. degree in Electrical Engineering from Tel Aviv University, Israel, in 2008 and 2013 respectively. He is a Senior Lecturer (Assistant Professor) with the Department of Software and Information Systems Engineering in Ben-Gurion University, Israel. His research interests include implementation security (power analysis and other hardware attacks and countermeasures; low-resource cryptographic constructions for lightweight computers) and cryptography in the real world (consumer and voter privacy in the digital era; web application security).

Yuval Yarom (M'16) is a senior lecturer in computer science at the University of Adelaide, where he heads the security domain in the Centre for Distributed and Intelligent Technologies. His research focuses on the security implications of the discrepancy between the nominal and the true behaviour of processors, with a focus on side channel and speculative execution attacks. He is the recipient of the 2020 Chris Wallace Award for Outstanding Research and is a DECRA Fellow.