Robust Website Fingerprinting Through the Cache Occupancy Channel

Anatoly Shusterman
Ben-Gurion University of the Negev
shustera@post.bgu.ac.il

Lachlan Kang
University of Adelaide
lachlan.kang@adelaide.edu.au

Yarden Haskal
Ben-Gurion Univ. of the Negev
yardenha@post.bgu.ac.il

Yosef Meltser
Ben-Gurion Univ. of the Negev
yosefmel@post.bgu.ac.il

Prateek Mittal
Princeton University
pmittal@Princeton.EDU

Yossi Oren
Ben-Gurion Univ. of the Negev
yos@bgu.ac.il

Yuval Yarom
University of Adelaide and Data61
yval@cs.adelaide.edu.au

Abstract

Website fingerprinting attacks, which use statistical analysis on network traffic to compromise user privacy, have been shown to be effective even if the traffic is sent over anonymity-preserving networks such as Tor. The classical attack model used to evaluate website fingerprinting attacks assumes an *on-path adversary*, who can observe all traffic traveling between the user's computer and the Tor network.

In this work we investigate these attacks under a different attack model, in which the adversary is capable of running a small amount of unprivileged code on the target user's computer. Under this model, the attacker can mount cache side-channel attacks, which exploit the effects of contention on the CPU's cache, to identify the website being browsed. In an important special case of this attack model, a JavaScript attack is launched when the target user visits a website controlled by the attacker. The effectiveness of this attack scenario has never been systematically analyzed, especially in the open-world model which assumes that the user is visiting a mix of both sensitive and non-sensitive sites.

In this work we show that cache website fingerprinting attacks in JavaScript are highly feasible, even when they are run from highly restrictive environments, such as the Tor Browser. Specifically, we use machine learning techniques to classify traces of cache activity. Unlike prior works, which try to identify cache conflicts, our work measures the overall occupancy of the last-level cache. We show that our approach achieves high classification accuracy in both the open-world and the closed-world models. We further show that our attack is more resistant than network-based fingerprinting to the effects of response caching, and that our techniques are resilient both to network-based defenses and to side-channel countermeasures introduced to modern browsers as a response to the Spectre attack. To protect against cache-based website fingerprinting, new defense mechanisms must be introduced to privacy-sensitive browsers and websites.

1 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 behaviour of users may reveal personal or sensitive information about the users, 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 [17, 19, 68]. Prime amongst these is the Tor network [19], 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. In addition to the network itself, the Tor Project also provides the *Tor Browser* [77], a modified version of the Mozilla Firefox web browser, that further protects users by disabling features that may allow web sites to track the users.

Past research has demonstrated that encrypting traffic is not sufficient for protecting the privacy of the users [10, 28, 34, 35, 36, 44, 57, 63, 64, 70, 83, 84]. 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 [70].

In this paper, we focus on an alternative attack model of exploiting microarchitectural side-channels, a less explored option for website fingerprinting. These attacks exploit information leaks through shared microarchitectural components such as caches [26]. The attack model assumes an adversary that can run untrusted code on the same hardware as the victim's browser, an assumption that can be justified in situations including multiuser systems, virtualized environments, and cloud-based services. These attacks observe the internal state of the target PC, rather than the network traffic. As such they offer the potential of overcoming traffic shaping, often proposed as a defense for website fingerprinting [11, 12, 15, 60, 85]. Similarly, they 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 [35].

One of the most compelling vectors for deployment of microarchitectural side-channel attacks is through JavaScript code injected into the user's web browser through a malicious advertisement or pop-up window. 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 [74]. The Der Spiegel claims that the GCHQ successfully used this tool to attack the computers of employees at the partly-government-held Belgian telecommunications company Belgacom, and that the NSA used the same technology 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 [27].

For a small number of websites, under the closed-world model, Oren et al. [61] show the possibility of fingerprinting via malicious JavaScript code. However, beyond showing the 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 [46, 54], 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 [66, 82], making it difficult to tell apart cache hits and cache misses. Traditionally, cache attacks require high-resolution timers, and while mechanisms to generate such timers in web browsers have been published [30, 47, 72], it is not clear that these can be used for website fingerprinting.

The Tor Browser poses a special challenge for cache attacks. Its timer has a resolution of 100 ms, two orders of magnitude coarser than any mainstream browser. Furthermore, it disables many features commonly supported by browsers, preventing known attack avenues. To the best of our knowledge, no microarchitectural attacks have so far been demonstrated via Javascript injection in the Tor Browser.

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

1.1 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 executed on modern mainstream browsers that include all recently introduced countermeasures for side-channel (Spectre) attacks. We further show that our attack is effective even in the highly restrictive environment of the Tor Browser, although with a drop in accuracy.

Our attack consists of collecting traces of cache *occupancy* while the browser downlods and renders web sites. Adapting the techniques of Rimmer et al. [70], 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 [55, 67, 86].

Finally, 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.

More specifically, we make the following contributions:

- We design and implement the cache occupancy side-channel attack, 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. [61] (Section 4).
- 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 5).
- 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 6).
- We evaluate both fingerprinting methods without deleting the browser response cache, and show that while the accuracy of network-based fingerprinting drops significantly, the accuracy of cache-based fingerprinting is not

affected (Section 7.3).

- 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 7.4).

2 Background

2.1 Tor

Tor, The Onion Router [19], 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 [18] which underlies the security of the HTTPS scheme [69]. However, this solution only protects the contents of the communication, leaving the identity of the communicating parties exposed to the adversary. Knowing that users merely 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` [48].

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 a few (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. [19] for further information.

2.2 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 [10, 28, 34, 35, 36, 44, 57, 63, 64, 70, 83, 84]. In response, several defense mechanisms have been proposed in the literature [11, 12, 15, 60, 85]. 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. [70] 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.

2.3 Cache Side-Channel Attacks

When programs execute on a processor, they share the use of microarchitectural components such as the cache. This sharing may result in unintended communication channels, often called *side channels*, between programs [26, 38], which may be used to leak secret information. In particular, cache-based attacks, which exploit contention on one of the processor caches, can leak secrets such as cryptographic keys [4, 25, 62, 65, 78], keystrokes [31], address layout [22, 30, 32], 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 [27, 61] employ the *Prime+Probe* technique [62, 65], 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 its 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 [62, 65] and instruction [3, 4] caches, as well as for attacks on the LLC [42, 56]. It has been shown practical in multiple settings, including in across different virtual machines in cloud environments [39] and from mobile code [27, 61].

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 publication of the first demonstration of a cache attack in JavaScript [61], some browsers started reducing the resolution of the timers they provide as a countermeasure for cache side channel attacks. This approach become wide-spread after the disclosure of the Spectre attack [46], and now all mainstream browsers incorporate this countermeasure. Furthermore, while non-traditional timers in browsers have been identified [24, 47, 72], browsers and extensions have since disabled many of the features that allow sub-microsecond resolution [58, 66, 73]. An extreme case of this behaviour can be found in the Tor Browser, which 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 [7], which rely on the absence of high-resolution timers to protect against timing attacks [80].

2.4 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 1, 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.[1] Some of these works assume that the adversary has malicious control over a hardware component or peripheral [16, 53, 88]. Others assume that the adversary can execute arbitrary native code on the target hardware [33, 43, 49, 75]. Yet others make the much more modest assumption that the adversary can induce the victim to render a webpage containing malicious JavaScript code [8, 45, 61, 81]. We investigate the last two models.

Kim et al. [45] 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 9.) 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 [61, 81] 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. [61] 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. [61] 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 [8] is able to classify a moderate amount of websites using a non-cache-based method with a millisecond clock, their attack saturates all of the victim's CPU cores with math-intensive worker threads, making it highly noticeable and easy to detect by the victim.

3 The Website Fingerprinting Attack Model

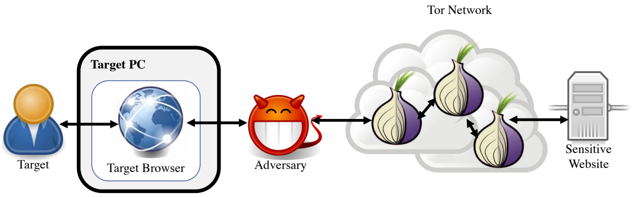

Figure 1: The classical website fingerprinting attack model. The (passive) adversary monitors the traffic between the target user and the Tor network.

The classical attack model used to evaluate website fingerprinting attacks is presented in Figure 1. In this model, a targeted user uses a web browser to display a sensitive website. To protect their privacy, the user does not connect to the website directly, but instead uses 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 as 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 secu-

[1] A different but closely related class of attacks are "history sniffing" attacks, such as [51, 87], in which the attacker wishes to learn which websites the victim has visited in the **past**.

Table 1: Related work on website fingerprinting based on local side channels.

Work	Target	Side Channel	Attack Model	Sampling rate [Hz]
Clark et al., 2013 [16]	Chrome (Mac, Win, Linux)	Power consumption	Hardware	250000
Yang et al., 2017 [88]	Multiple smartphones	Power consumption	Hardware	200000
Lifshits et al., 2018 [53]	Android Browser, Chrome Android	Power consumption	Hardware	1000
Jana and Shmatikov, 2012 [43]	Chrome Linux, Firefox Linux, Android Browser (VM)	App memory footprint	Native code	100000
Lee et al., 2014 [49]	Chromium Linux, Firefox Linux	GPU memory leaks	Native code	N/A
Spreitzer et al., 2016 [75]	Chrome Android, Android Browser, Tor Android	Data-Usage Statistics	Native code	20–50
Gülmezoglu et al., 2017 [33]	Chrome Linux (Intel and ARM), Tor Linux	Performance counters	Native code	10000
Oren et al, 2015 [61]	Safari MacOS, Tor MacOS	Last-level cache	JavaScript	10^8
Booth, 2015 [8]	Chrome (Mac, Win, Linux), Firefox Linux	CPU activity	JavaScript	1000
Kim et al., 2016 [45]	Chromium Linux, Chrome (Win, Android)	Quota Management API	JavaScript	N/A
Vila and Köpf, 2017 [81]	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**

rity 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 users' communications [10, 34, 35, 36, 44, 57, 63, 64, 70, 83, 84]. Gong et al. [28] 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 estimated traffic.

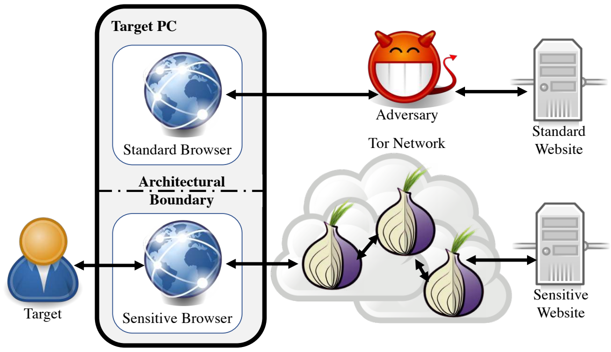

Figure 2: Remote cache-based website fingerprinting attack model. The remote attacker injects malicious JavaScript code into a browser running on the target machine.

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 web site. These two 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 [71]. 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 ultimate 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 microarchitectural state of the processor, and use this information to spy on the sensitive session.

One possible way of causing the target to browse such an adversary-controlled site is 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 web site. Alternatively, the attacker may pay an advertisement service to display a (malicious) advertisement when the user visits a third-party website [27]. Finally, when users do not encrypt some part of their traffic, an active on-path adversary is immediately capable of carrying out this kind of attack by actively injecting this malicious code into all traffic passing through it from the network to the target computer. Importantly, even a more restricted attacker can still mount this attack. For example, if the user simultaneously runs one browsing session over a VPN connection for sensitive tasks, and another browsing session over an unsecured connection for mundane tasks, a remote attack launched over the standard link can target the data exchanged over the secured link. Such an attack can be launched either by compromising the link, or by compromising any of the components of the insecure website. The main challenge of the remote attack model is the extremely restricted JavaScript runtime, which requires the attacker code to be written in a particular way, as we describe further in Section 4.

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 7.4, both of these elements contribute to the accuracy of our classifier.

4 Data Collection

4.1 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* [61]—a trace of the cache access latency measured at a constant sampling rate over a given time period. The memorygrams of Oren et al. [61] 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 [27, 61], which use the Prime+Probe side-channel attack, 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 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.

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 [41], we use the technique of Yarom and Benger [90] 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 [56, 62].

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 that 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 [27].

Website Memorygrams. We capture memorygrams when the browser navigates to websites and displays them. We use the same JavaScript-based collection method for all mainstream browsers other than the Tor Browser, where we 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.

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 timeslot. 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 7 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.

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 dis-

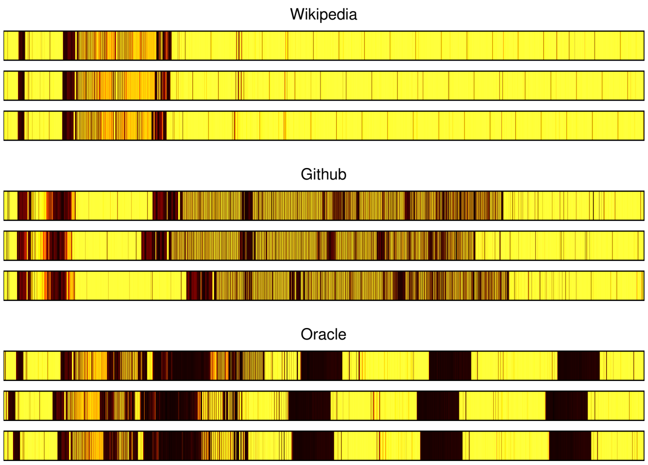

Figure 3: Examples of memorygrams. Time progresses from left to right, shade indicates the number of evictions. (Darker shades correspond to more eviction.)

played as a coloured strip, where time goes from left to right and the shade corresponds to cache activity at each time. (Lighter shades correspond to less 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.

4.2 Datasets

Closed World Datasets. We evaluate our cache-based fingerprinting on six different combinations of browsers and operating systems, summarized in Table 5. 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. [70] selected from the top Alexa sites. (See Appendix A for a complete list of websites included.) Similar to previous works, no traffic molding is applied and only one tab is opened at a time. The browser's response cache, however, is not cleared before accessing each website, an aspect of the experiment we analyze in more detail in Section 7.

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. [44]. 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. [70]. The base rate for this setting is 33.3%, since a trivial classifier can simply decide that all pages are non-sensitive.

5 Machine Learning

5.1 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 algorithm 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.

5.2 Deep Learning Models

Early works on website fingerprinting, starting from Cheng and Avnur [14], used classical machine learning methods such as Naive Bayes, Support Vector Machine (SVM) and k-Nearest Neighbors (k-NN). 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. [70] 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. [70] show that, given a large enough training set, deep-learning website-fingerprinting approaches are as effective as earlier methods which require manual feature selection. An advantage of this approach is that it allows us to compare network-based and cache-based fingerprinting based on the merit of the raw data, rather than on the 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 [29]. 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 it's weight value, and then forwarded to the next layer.

For the last layer in the DNNs we evaluate we use a softmax layer, which outputs a vector containing a-posteriori probabilities for each one of the classes.

The process of training the neural network uses backpropagation 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 backpropagation 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 [37]. 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 specialised 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 an additional 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. [70]. For cache-based fingerprinting, we manually evaluated several choices for each hyperparameter.

Table 2: Hyperparameters for the CNN classifier

Hyperparameter	**Value**	**Space**
Optimizer	Adam	Adamax, Adam, SGD, RMSprop
Learning rate	0.001	0.001–0.002
Batch size	100	40–100
Training epoch	20–30	Early stop by accuracy
Convolution layers	3	3–4
Input units (FF)	15000	15000–25000
Input units (Tor)	25000	15000–25000
CNN activation	relu	relu, tanh
Kernels	256	2–512
Kernel size	16,8,4	2–31
Pool size	4	2–8

Table 3: Hyperparameters for the LSTM classifier

Hyperparameter	**Value**	**Space**
Optimizer	Adam	Adamax, Adam, SGD, RMSprop
Learning rate	0.001	0.001–0.002
Batch size	100	40–100
Training epoch	20–30	Early stop by accuracy
Convolution layers	2	1–3
Input units (FF)	15000	15000–25000
Input units (Tor)	25000	15000–25000
CNN activation	relu	relu, tanh
LSTM activation	tanh	relu,tanh
Kernels	256	2–512
Kernel size	16,8	2–32
Pool size	4	2–8
Dropout	0.2	0.1–0.2
LSTM units	32	8,32

To prevent overfitting, we use 10-fold cross validation. We split each dataset consisting of traces into 10 folds of equal size, and select one fold, consisting of 10% of the traces, 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. The selected hyperparameters are summarized in Tables 2, 3, and 4.

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 and we we only used a pool-size of three for the max-pooling layer to limit the feature reduction before the LSTM layer. In addition, because the small amount of fea-

Table 4: Hyperparameters for the LSTM classifier for the Tor attack

Hyperparameter	Value	Space
Optimizer	Adam	Adamax, Adam, SGD, RMSprop
Learning rate	0.001	0.001–0.002
Batch size	100	40–100
Training epoch	20–30	Early stop by accuracy
Convolution layers	1	1–3
Input units	500	500
CNN activation	relu	relu, tanh
LSTM activation	tanh	relu,tanh
Kernels	256	2–512
Kernel size	32	2–32
Pool size	3	2–8
Dropout	0.4	0.1–0.4
LSTM units	128	8,32,128

tures we could increase the LSTM units to 128 for learning more complex patterns from the features.

6 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 with two Xeon E5-2660 v4 processors and 128 GB of RAM, and one with two Xeon E5-2620 v3 processors and 128 GB of RAM. Our machine learning instances took approximately 40 minutes to run in this configuration.

Table 5 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 [66, 82], the first question we need to address is whether it is even possible to implement cache-based fingerprinting attacks in such an environment.

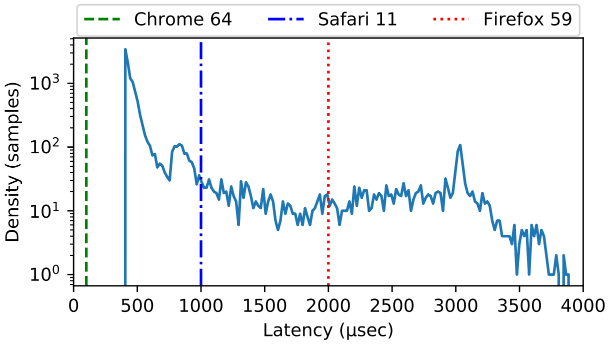

Figure 4: Cache probe latencies compared to modern browser timing resolutions.

To answer this question, we measured the latencies of the cache occupancy channel using a high-resolution timer while the browser was downloading a web page. Figure 4 shows the distribution of these latencies. The figure also uses vertical lines to indicate the timer resolutions of the various browsers. (See Table 5.) 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 4.)

The next question is whether the information we collect with this low resolution is sufficient for fingerprinting. Indeed, Table 5 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 of general-purpose browsers.

6.1 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%. If we, however, look not only at the top result output by the classifier, but also check whether the correct website is one of the top 5 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 [13, 59]. With some a-priori information an attacker can deduce which of the top 5 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. [70]. We see that while there are differences between the classification accuracy 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.

6.2 Open World Results

We next turn to the more challenging open-world scenario, in which the 100 sensitive webpages must be distinguished from an additional set of 5,000 non-sensitive pages. As seen in Table 5 the JavaScript-based website fingerprinting code performs well under this scenario as well, again achieving

Table 5: Accuracy obtained by in-browser memorygrammer— Mean (percents) and standard deviation.

Operating		LLC		Timer	Closed World		Open World	
System	CPU	Size	Browser	Resolution	CNN	LSTM	CNN	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

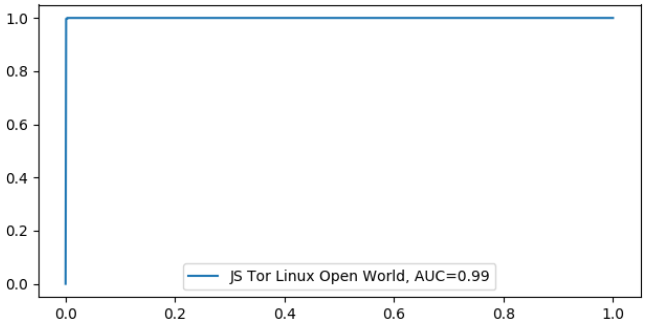

Figure 5: ROC curve of a JavaScript attack on Tor for Linux.

classification accuracy of 70–90%. We note that in most cases the results are slightly better than the closed-world results. The reason is the larger size of the "non-sensitive" class. As discussed earlier, this also significantly increases the base rate for open-world scenarios to 33.3%.

As in the case of the closed-world setting, we can evaluate the accuracy of the Tor Browser under a top-5 assumption. Under this relaxation the Tor Browser attack achieves a high accuracy rate of 83%, with a base rate of 37.3%.

The classification to sensitive vs. non-sensitive site is a binary classification problem, We can, therefore, apply standard analysis techniques to this aspect of the results. 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, as demonstrated by the ROC curve of the Tor Linux open world dataset shown in Figure 5.

7 Robustness Tests

Having demonstrated the effectiveness of our website fingerprinting technique, we now turn our attention to its robustness and test its resilience to issues known to affect network-based fingerprinting.

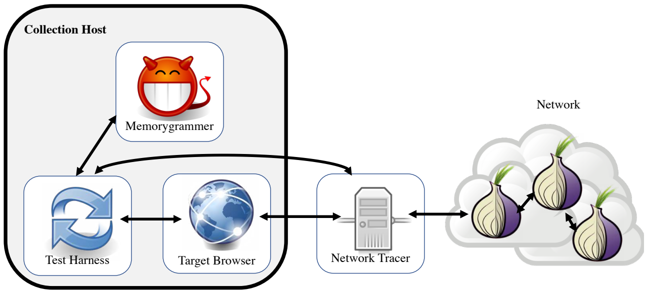

Figure 6: Data Collection Setup for the Robustness Tests.

7.1 Evaluation Setup

To compare the results of network fingerprinting with cache-based fingerprinting, we need to modify out data collection setup. The setup, illustrated in Figure 6, 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 hosts and the Internet and collect 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 [89]. 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 [40] to count the number of cache evictions rather than use the high resolution timer to identify evictions. The

Table 6: 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**
Temporal drift	–	–	–	–	64.5±2.2	81.0±0.6	**68.3±0.5**	**75.6±0.7**

use of performance counters for attack purposes has already been proposed and investigated in the past [6, 9, 50, 79].

7.2 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. [70]. As we can see in Table 6, 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, the results of the native-code memorygrammer are similar to those achievable with network-based fingerprinting.

7.3 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 [23], 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. [35] 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 [63, 83].

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 starts to emerge. As Table 6 shows, the accuracy of the standard network-based methods degrades when the response 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 essentially a special case of network-based classifiers, but are rather detecting rendering activities of the browser process.

7.4 Net-only and Render-only Results

Oren et al. [61] show that cache activity is correlated with network activity, raising the possibility that cache-based fingerprinting basically 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 [21], 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 should be countered using a different class of mitigation techniques, as we explore further in Section 9.

Network-Only Fingerprinting. In a complementing experiment, we aim to capture only the network traffic. To collect this dataset, we first capture actual 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, by replaying the data from the `pcap` file.

The results for this experiment show that the cache-based classifier is capable of classifying many pages even when no rendering activity is taking place. 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 therefore conclude that cache-based fingerprinting identifies features both in the network traffic patterns and in the actual *contents* of the displayed web pages.

7.5 Dealing with Temporal Drift

The accuracy of network-based website fingerprinting decays over time, when the contents of the website changes [70]. Many websites use content management systems (CMS), in which the page layout is based on a fixed template design, and only the resources loaded into this template vary over time. Since, as we have shown, the cache-based fingerprints capture rendering activities as well as network activities, it would seem that the rendering-related traces recorded by the cache-based method would have a longer lifetime, and be more resistant to drift, than the network-related traces captured by the traditional method.

To test this hypothesis, we repeat the data collection of the baseline experiment after a delay of 36 days (start to start). We then measure the ability of both cache-based and network-based classifiers to accurately classify the new traces, after being trained on the old traces. In this setting, we see a drop of 5–10% in the accuracy of both classifiers. We believe that further experiments are required for accurately assessing how cache-based and network-based fingerprinting handle temporal drifts.

8 Detecting Unknown Hardware Configurations

In contrast to network-based fingerprinting, which is largely target agnostic, cache-based fingerprinting needs to be tailored to the precise hardware configuration of the victim machine, specifically the set count and associativity of its last-level cache. Using a too large or a too small buffer reduces the effectiveness of the technique, and eventually the accuracy of the classifier. There are, however, not that many popular configurations. For example, four cache configurations (4096 or 8192 sets, 12 or 16 ways) cover most of the Intel Core processor models.

If the target hardware configuration is known beforehand (assuming, for example, that a particular user is singled out for attack) the attacker can customize the parameters of the JavaScript attack code to match the target PC's parameters. It would be interesting, however, to see how well an attacker can remotely determine an unknown target's cache configuration using JavaScript. To investigate this, we created a JavaScript program that allocates a 20MB 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. Thus generic attacks that adapt to the specific hardware onfiguration seem feasible.

9 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 [5] 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 this work demonstrates, timer-based defenses that reduce the timer frequency or add jitter are not effective.

Cache randomization techniques [55, 67, 86] 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 [20, 86] or via page coloring [52], 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 [76] as part of the mitigations for Spectre [46], 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 [91] limits the contention caused by each process as a protection for the Prime+Probe attack. Like cache 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.

One potential mitigation is to adapt masking techniques from the network fingerprinting and create spurious activity in the cache. 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. Our initial experiments show that this is a promising mitigation, but further research is needed to assess its effectiveness and its effect on performance and on power consumption.

10 Limitations and Future Work

While the work demonstrates the feasibility of cache-based website fingerprinting and provides an analysis of the attack, it does leave some areas for further study. Being the first analysis of its kind, the scope of the work does not match the scope of similar works on network-based website fingerprinting. In particular, our datasets are significantly smaller than those of Rimmer et al. [70], for example. Providing larger datasets would allow better analysis of the effectiveness of the technique and would be a beneficial service for the research community as a whole.

In this work we collected the memorygrams on the same hardware configuration used by the victim PC. While we show that we can adapt the data collection to the specific victim hardware (Section 8), at this stage it is not clear how much a classifier trained on data collected with one hardware configuration would be effective for classifying memorygrams collected on a different configuration.

The work further shares many of the limitations of network-based fingerprinting [44]. 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. Both Rimmer et al. [70] and our work briefly discuss temporal aspects of website fingerprinting, and we also looked a bit into the issue (Section 7.5). However, further work is required to assess the impact of this and other variables on the efficacy of cache-based fingerprinting.

11 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 result comparable with the state-of-the-art network-based fingerprinting. We further show that cache-based fingerprinting outperforms network-based fingerprinting under a common operating scenario, where the browser maintains cached 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 microarchitectural 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.

Acknowledgements

We would like to thank Vera Rimmer for her helpful comments and insights.

This research was supported by the ARC Centre of Excellence for Mathematical & Statistical Frontiers, NSF CNS-1409415, NSF CNS-1704105, and Israel Science Foundation grants 702/16 and 703/16.

References

[1] Tcpreplay. https://tcpreplay.appneta.com/.

[2] Kota Abe and Shigeki Goto. Fingerprinting attack on Tor anonymity using deep learning. In *Proceedings of the APAN – Research Workshop 2016*, 2016.

[3] Onur Acıiçmez. Yet another microarchitectural attack: : exploiting I-Cache. In *CSAW*, pages 11–18, 2007.

[4] Onur Acıiçmez, Billy Bob Brumley, and Philipp Grabher. New results on instruction cache attacks. In *CHES*, pages 110–124, 2010.

[5] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. The security impact of a new cryptographic library. In *LATINCRYPT*, pages 159–176, 2012.

[6] Sarani Bhattacharya and Debdeep Mukhopadhyay. Who watches the watchmen?: Utilizing performance monitors for compromising keys of RSA on Intel platforms. In *CHES*, pages 248–266, 2015.

[7] Zack Bloom. Cloud computing without containers. https://blog.cloudflare.com/cloud-computing-without-containers/, 2018.

[8] Jo M. Booth. Not so incognito: Exploiting resource-based side channels in JavaScript engines. Bachelor thesis, Harvard, April 2015.

[9] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure: SGX cache attacks are practical. In *WOOT*, 2017.

[10] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a distance: website fingerprinting attacks and defenses. In *ACM CCS*, pages 605–616, 2012.

[11] Xiang Cai, Rishab Nithyanand, and Rob Johnson. Csbuflo: A congestion sensitive website fingerprinting defense. In *WPES*, pages 121–130, 2014.

[12] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *ACM CCS*, pages 227–238, 2014.

[13] Aylin Caliskan-Islam, Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, Fabian Yamaguchi, and Rachel Greenstadt. De-anonymizing programmers via code stylometry. In *USENIX Sec*, pages 255–270, 2015.

[14] Heyning Cheng and Ron Avnur. Traffic analysis of SSL encrypted web browsing. Project paper, University of Berkeley, 1998.

[15] Giovanni Cherubin, Jamie Hayes, and Marc Juárez. Website fingerprinting defenses at the application layer. *PoPETs*, 2017(2):186–203, 2017.

[16] Shane S. Clark, Hossen A. Mustafa, Benjamin Ransford, Jacob Sorber, Kevin Fu, and Wenyuan Xu. Current events: Identifying webpages by tapping the electrical outlet. In *ESORICS*, pages 700–717, 2013.

[17] Wei Dai. PipeNet description. Post to the cypherpunks mailing list. Copy available at https://www.freehaven.net/anonbib/cache/pipenet10.html, 1998.

[18] T. Dierks and E. Rescola. The transport layer security (TLS) protocol version 1.2. RFC 5246, RFC Editor, 2008. https://tools.ietf.org/html/rfc5246.

[19] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX Security*, pages 303–320, 2004.

[20] Leonid Domnitser, Aamer Jaleel, Jason Loew, Nael B. Abu-Ghazaleh, and Dmitry Ponomarev. Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks. *TACO*, 8(4):35:1–35:21, 2012.

[21] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-Boo, I still see you: Why efficient traffic analysis countermeasures fail. In *IEEE SP*, pages 332–346, 2012.

[22] Dmitry Evtyushkin, Dmitry V. Ponomarev, and Nael B. Abu-Ghazaleh. Jump over ASLR: attacking branch predictors to bypass ASLR. In *MICRO*, pages 40:1–40:13, 2016.

[23] R. Fielding, M. Nottingham, and J. Reschke. Hypertext transfer protocol (HTTP/1.1): Caching. RFC 7234, RFC Editor, June 2014. http://www.rfc-editor.org/rfc/rfc7234.txt.

[24] Pietro Frigo, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Grand pwning unit: Accelerating microarchitectural attacks with the GPU. In *IEEE SP*, pages 195–210, 2018.

[25] Cesar Pereida García, Billy Bob Brumley, and Yuval Yarom. "Make sure DSA signing exponentiations really are constant-time". In *ACM CCS*, pages 1639–1650, 2016.

[26] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. *J. Cryptographic Engineering*, 8(1):1–27, 2018.

[27] Daniel Genkin, Lev Pachmanov, Eran Tromer, and Yuval Yarom. Drive-by key-extraction cache attacks from portable code. In *ACNS*, 2018.

[28] Xun Gong, Nikita Borisov, Negar Kiyavash, and Nabil Schear. Website detection using remote traffic analysis. In *Privacy Enhancing Technologies*, pages 58–78, 2012.

[29] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning (Adaptive Computation and Machine Learning series)*. The MIT Press, 2016. ISBN 0262035618.

[30] Ben Gras, Kaveh Razavi, Erik Bosman, Herbert Bos, and Cristiano Giuffrida. ASLR on the line: Practical cache attacks on the MMU. In *NDSS*, 2017.

[31] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. Cache template attacks: Automating attacks on inclusive last-level caches. In *USENIX Security*, pages 897–912, 2015.

[32] Daniel Gruss, Clémentine Maurice, Anders Fogh, Moritz Lipp, and Stefan Mangard. Prefetch side-channel attacks: Bypassing SMAP and kernel ASLR. In *ACM CCS*, pages 368–379, 2016.

[33] Berk Gülmezoglu, Andreas Zankl, Thomas Eisenbarth, and Berk Sunar. PerfWeb: How to violate web privacy with hardware performance events. In *ESORICS (2)*, pages 80–97, 2017.

[34] Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *USENIX Security*, pages 1187–1203, 2016.

[35] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *CCSW*, pages 31–42, 2009.

[36] Andrew Hintz. Fingerprinting websites using traffic analysis. In *Privacy Enhancing Technologies*, pages 171–178, 2002.

[37] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[38] Wei-Ming Hu. Lattice scheduling and covert channels. In *IEEE SP*, pages 52–61, 1992.

[39] Mehmet Sinan Inci, Berk Gülmezoglu, Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. Cache attacks enable bulk key recovery on the cloud. In *CHES*, pages 368–388, 2016.

[40] Intel Corp. Intel 64 and IA-32 architectures software developer's manual volume 3B, September 2016. URL https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf.

[41] Intel Corp. Intel 64 and IA-32 architectures optimization reference manual, June 2016. URL https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html.

[42] Gorka Irazoqui Apecechea, Thomas Eisenbarth, and Berk Sunar. S$A: A shared cache attack that works across cores and defies VM sandboxing - and its application to AES. In *IEEE SP*, pages 591–604, 2015.

[43] Suman Jana and Vitaly Shmatikov. Memento: Learning secrets from process footprints. In *IEEE SP*, pages 143–157, 2012.

[44] Marc Juárez, Sadia Afroz, Gunes Acar, Claudia Díaz, and Rachel Greenstadt. A critical evaluation of website fingerprinting attacks. In *ACM CCS*, pages 263–274, 2014.

[45] Hyungsub Kim, Sangho Lee, and Jong Kim. Inferring browser activity and status through remote monitoring of storage usage. In *ACSAC*, pages 410–421, 2016.

[46] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Haburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwartz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *IEEE SP*, pages 19–37, May 2019.

[47] David Kohlbrenner and Hovav Shacham. Trusted browsers for uncertain times. In *USENIX Sec*, pages 463–480, 2016.

[48] Nil 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, January 2018.

[49] Sangho Lee, Youngsok Kim, Jangwoo Kim, and Jong Kim. Stealing webpages rendered on your browser by exploiting GPU vulnerabilities. In *IEEE SP*, pages 19–33, 2014.

[50] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In *USENIX Security*, pages 557–574, 2017.

[51] Bin Liang, Wei You, Liangkun Liu, Wenchang Shi, and Mario Heiderich. Scriptless timing attacks on web browser privacy. In *DSN*, pages 112–123, 2014.

[52] Jochen Liedtke, Hermann Härtig, and Michael Hohmuth. OS-controlled cache predictability for real-time systems. In *IEEE RTAS*, pages 213–224, 1997.

[53] Pavel Lifshits, Roni Forte, Yedid Hoshen, Matt Halpern, Manuel Philipose, Mohit Tiwari, and Mark Silberstein. Power to peep-all: Inference attacks by malicious batteries on mobile devices. *PoPETs*, 2018 (4):1–1, 2018.

[54] Moritz Lipp, Michael Schwartz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *USENIX Security*, August 2018.

[55] Fangfei Liu and Ruby B. Lee. Random fill cache architecture. In *MICRO*, pages 203–215, 2014.

[56] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. Last-level cache side-channel attacks are practical. In *IEEE SP*, pages 605–622, 2015.

[57] Liming Lu, Ee-Chien Chang, and Mun Choon Chan. Website fingerprinting and identification using ordered feature sequences. In *ESORICS*, pages 199–214, 2010.

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

[59] Arvind Narayanan, Hristo Paskov, Neil Zhenqiang Gong, John Bethencourt, Emil Stefanov, Eui Chul Richard Shin, and Dawn Song. On the feasibility of internet-scale author identification. In *IEEE SP*, pages 300–314, 2012.

[60] Rishab Nithyanand, Xiang Cai, and Rob Johnson. Glove: A bespoke website fingerprinting defense. In *WPES*, pages 131–134, 2014.

[61] Yossef Oren, Vasileios P. Kemerlis, Simha Sethumadhavan, and Angelos D. Keromytis. The spy in the sandbox: Practical cache attacks in JavaScript and their implications. In *ACM CCS*, pages 1406–1418, 2015.

[62] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of AES. In *CT-RSA*, pages 1–20, 2006.

[63] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *WPES*, pages 103–114, 2011.

[64] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. Website fingerprinting at internet scale. In *NDSS*, 2016.

[65] Colin Percival. Cache missing for fun and profit. Presented at BSDCan. http://www.daemonology.net/hyperthreading-considered-harmful, 2005.

[66] Filip Pizlo. What Spectre and Meltdown mean for WebKit. https://webkit.org/blog/8048/what-spectre-and-meltdown-mean-for-webkit/, January 2018.

[67] Moinuddin K. Qureshi. CEASER: Mitigating conflict-based cache attacks via encrypted-address and remapping. In *MICRO*, 2018.

[68] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Trans. Inf. Syst. Secur.*, 1(1):66–92, 1998.

[69] E. Rescola. HTTP over TLS. RFC 2818, RFC Editor, 2000. https://tools.ietf.org/html/rfc2818.

[70] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. In *NDSS'18*, 2018.

[71] Joanna Rutkowska and Rafal Wojtczuk. *Qubes OS Architecture*, February 2010. URL https://www.qubes-os.org/attachment/wiki/QubesArchitecture/arch-spec-0.3.pdf.

[72] Michael Schwarz, Clémentine Maurice, Daniel Gruss, and Stefan Mangard. Fantastic timers and where to find them: High-resolution microarchitectural attacks in JavaScript. In *Financial Cryptography*, pages 247–267, 2017.

[73] Michael Schwarz, Moritz Lipp, and Daniel Gruss. JavaScript zero: Real JavaScript and zero side-channel attacks. In *NDSS*, 2018.

[74] 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`, December 2013.

[75] Raphael Spreitzer, Simone Griesmayr, Thomas Korak, and Stefan Mangard. Exploiting data-usage statistics for website fingerprinting attacks on Android. In *WISEC*, pages 49–60, 2016.

[76] The Chromium Project. Site isolation. `https://www.chromium.org/Home/chromium-security/site-isolation`.

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

[78] Yukiyasu Tsunoo, Teruo Saito, Tomoyasu Suzaki, Maki Shigeri, and Hiroshi Miyauchi. Cryptanalysis of DES implemented on computers with cache. In *CHES*, pages 62–76, 2003.

[79] Leif Uhsadel, Andy Georges, and Ingrid Verbauwhede. Exploiting hardware performance counters. In *FDTC*, pages 59–67, 2008.

[80] Kenton Varda. `https://news.ycombinator.com/item?id=18280156`, 2018.

[81] Pepe Vila and Boris Köpf. Loophole: Timing attacks on shared event loops in Chrome. In *USENIX Security*, pages 849–864, 2017.

[82] Luke Wagner. Mitigations landing for new class of timing attack. `https://blog.mozilla.org/security/2018/01/03/mitigations-landing-new-class-timing-attack/`, January 2018.

[83] Tao Wang and Ian Goldberg. Improved website fingerprinting on Tor. In *WPES*, pages 201–212, 2013.

[84] Tao Wang and Ian Goldberg. On realistically attacking Tor with website fingerprinting. *PoPETs*, 2016(4):21–36, 2016.

[85] Tao Wang and Ian Goldberg. Walkie-Talkie: An efficient defense against passive website fingerprinting attacks. In *USENIX Security*, pages 1375–1390, 2017.

[86] Zhenghong Wang and Ruby B. Lee. New cache designs for thwarting software cache-based side channel attacks. In *ISCA*, pages 494–505, 2007.

[87] Zachary Weinberg, Eric Yawei Chen, Pavithra Ramesh Jayaraman, and Collin Jackson. I still know what you visited last summer: Leaking browsing history via user interaction and side channel attacks. In *IEEE SP*, pages 147–161, 2011.

[88] Qing Yang, Paolo Gasti, Gang Zhou, Aydin Farajidavar, and Kiran S. Balagani. On inferring browsing activity on smartphones via USB power analysis side-channel. *IEEE Trans. Information Forensics and Security*, 12(5):1056–1066, 2017.

[89] Yuval Yarom. Mastik: A micro-architectural side-channel toolkit. `http://cs.adelaide.edu.au/~yval/Mastik/Mastik.pdf`, September 2016.

[90] Yuval Yarom and Naomi Benger. Recovering OpenSSL ECDSA nonces using the FLUSH+RELOAD cache side-channel attack. Cryptology ePrint Archive, Report 2014/140, 2014. URL `http://eprint.iacr.org/2014/140`.

[91] Ziqiao Zhou, Michael K. Reiter, and Yinqian Zhang. A software approach to defeating side channels in last-level caches. In *ACM CCS*, pages 871–882, 2016.

A Websites Included in Closed-World Datasets

9gag.com
abs-cbn.com
adf.ly
adobe.com
aliexpress.com
allegro.pl
amazon.com
amazonaws.com
aol.com
apple.com
archive.org
askcom.me
battle.net
blastingnews.com
booking.com
breitbart.com
bukalapak.com
businessinsider.com
conservativetribune.com
dailymail.co.uk
dailymotion.com
detik.com
deviantart.com
dictionary.com
digikala.com
doubleclick.net
doublepimp.com
ebay.com
espncricinfo.com
exoclick.com
extratorrent.cc
facebook.com
feedly.com
gamepedia.com
github.com
go.com
godaddy.com
goodreads.com
google.com
hclips.com
hola.com
hotmovs.com
imdb.com
instructure.com
intuit.com
kompas.com
leboncoin.fr
liputan6.com
livejasmin.com
livejournal.com
ltn.com.tw
microsoftonline.com
mozilla.org
msn.com
naver.com
netflix.com
nicovideo.jp
nih.gov
ntd.tv
office.com
onedio.com
openload.co
oracle.com
ouo.io
outbrain.com
pinterest.com
popads.net
quora.com
researchgate.net
roblox.com
rt.com
rutracker.org
scribd.com
skype.com
soundcloud.com
sourceforge.net
spotify.com
spotscenered.info
stackexchange.com
stackoverflow.com
steamcommunity.com
steampowered.com
t.co
theguardian.com
thesaurus.com
tistory.com
tokopedia.com
torrentz2.eu
tribunnews.com
tumblr.com
twitter.com
weather.com
wikia.com
wikipedia.org
wittyfeed.com
xhamster.com
xvideos.com
yandex.ru
yelp.com
zippyshare.com