PerfWeb: How to Violate Web Privacy with Hardware Performance Events

Berk Gulmezoglu[1](✉), Andreas Zankl[2], Thomas Eisenbarth[1], and Berk Sunar[1]

[1] Worcester Polytechnic Institute, Worcester, MA, USA
{bgulmezoglu,teisenbarth,sunar}@wpi.edu
[2] Fraunhofer Research Institution AISEC, Munich, Germany
andreas.zankl@aisec.fraunhofer.de

Abstract. The browser history reveals highly sensitive information about users, such as financial status, health conditions, or political views. Private browsing modes and anonymity networks are consequently important tools to preserve the privacy not only of regular users but in particular of whistleblowers and dissidents. Yet, in this work we show how a malicious application can infer opened websites from Google Chrome in Incognito mode and from Tor Browser by exploiting hardware performance events (HPEs). In particular, we analyze the browsers' microarchitectural footprint with the help of advanced Machine Learning techniques: k-th Nearest Neighbors, Decision Trees, Support Vector Machines, and in contrast to previous literature also Convolutional Neural Networks. We profile 40 different websites, 30 of the top Alexa sites and 10 whistleblowing portals, on two machines featuring an Intel and an ARM processor. By monitoring retired instructions, cache accesses, and bus cycles for at most 5 seconds we manage to classify the selected websites with a success rate of up to 86.3%. The results show that hardware performance events can clearly undermine the privacy of web users. We therefore propose mitigation strategies that impede our attacks and still allow legitimate use of HPEs.

Keywords: Website Fingerprinting, Hardware Performance Events, Machine Learning, Incognito Mode, Chrome, Tor, Onion Routing, Privacy

1 Introduction

Web browsers are indispensable components in our lives. They provide access to news and entertainment, and more importantly serve as a platform through which we perform privacy and security sensitive interactions such as online banking, web enabled healthcare, and social networking. Knowing the websites a user is visiting therefore reveals personal and highly sensitive information. To preserve the privacy of users, browsers consequently implement *private browsing* or *incognito* modes, which leave no traces of visited websites. More comprehensive protection is achieved by *Onion routing*, e.g. *Tor*, which protects users against Internet surveillance by obscuring packet routing information. By using a Tor enabled browser users may hide the websites they visit from adversaries monitoring

their network communication. This has become indispensable for whistleblowers and dissidents who try to protect their identity against powerful corporations and repressive governments. Besides web browsers, other tools have emerged to mask the identity of the user, e.g. Signal/Redphone, Silent Phone, and Telegram. However, even the installation of such tools can be viewed as subversive action by a repressive regime. In contrast, privacy preserving browsers come pre-installed on many platforms.

While browsers have significantly matured in providing privacy assurances, they are still far from perfect. In 2012, Jana and Shmatikov [15] found that memory footprints of browser processes are unique enough while browsing to detect opened websites. We expand on this observation and show that browsers have a somewhat similar footprint at the hardware level, more specifically in the microarchitecture of processors, and that this footprint can be used to infer web browsing activity with high success. The key to our inference attack is that most applications exhibit different execution behavior depending on the input they are processing. They consequently stress the processor hardware in different ways. Whichever application is able to observe these load patterns can learn a great deal of what is being processed in other programs. Modern processors and operating systems provide such interfaces to their microarchitectural state through hardware performance events (HPEs). While HPEs are legitimately used for debugging purposes and performance analysis, e.g. to keep track of cache misses or branch mispredictions, they can also be leveraged by adversaries. We show that it is feasible for a malicious application to monitor HPEs from user space and to infer opened websites even when users browse in Incognito mode or with the Tor Browser. For the experiments in this work, we use the `perf` subsystem of the Linux kernel, a wide-spread user space interface to hardware performance events. Since HPE based information is incidental and often noisy, advanced methods for data analysis are needed. The recent advances in Machine Learning (ML) provide us with a powerful tool to classify the complex noisy data in an effective manner. We show that while k-th Nearest Neighbors, Support Vector Machines, and Decision Trees are not sufficient to classify the complex and noisy observed data into a high number of different classes, Convolutional Neural Networks, a Deep Learning technique, can efficiently extract meaningful data even in the presence of severe noise. As a result, we demonstrate that it is possible to infer the web activity from HPEs with very high success rates and in a highly automated fashion. This can be a considerable threat in practice, as many of the applications we use every day stem from third-parties and run in the background. We trust these applications, even though we have little control over what is executed on our devices.

Our Contribution. In summary, we use advanced Machine Learning techniques, including Convolutional Neural Networks, to process different types of HPEs that have been acquired through `perf` and combined to get a better classification rate. We cover 40 different websites, including 30 of the top Alexa sites and 10 whistleblowing portals, and detect different web pages of a domain to show that fine-grained browser profiling is possible. We demonstrate that an at-

tacker does not need to precisely synchronize with the browser, as misalignment is compensated by the ML techniques. We also show that it suffices to monitor Google Chrome and Tor Browser for at most 5 seconds to classify websites with high accuracy. We finally outline possible mitigation strategies that impede website inference while still allowing access to performance profiling.

2 Background and Related Work

This following paragraphs provide background information and related work regarding HPEs and website fingerprinting. Subsequently, we briefly compare our work to previous ones.

Hardware Performance Events. The microarchitectures of modern processors implement a large spectrum of performance enhancing features that speed up memory accesses and code execution. As a compromise, performance enhancements introduce input dependent runtimes and weak separation between executing applications. As a direct consequence, the microarchitectural state of a processor contains crucial information about the processes that are executed on it. Hardware performance events are an interface to this state that is implemented on most modern processors. A dedicated piece of hardware, the performance monitoring unit (PMU), is responsible to keep track of microarchitectural events that occur while executing code on the processor. These events include, e.g., instruction retirements, branch mispredictions, and cache references. They provide a comprehensive picture of a processor's runtime behavior and are therefore interesting for adversaries and developers alike. In literature, clock cycle events have been recognized as a vital timing source for a large class of cache-based attacks [22,35]. Cache miss [30] and branch misprediction events [5] have been exploited in targeted attacks against implementations of AES and RSA. In contrast, HPEs have improved our understanding of attacks [3] and modern processor implementations [23], facilitated the evaluation of software components [34], and helped to analyze malware samples [32]. A large class of previous work is dedicated to the real time detection of attacks and malware infections, a selection of which relies on Machine Learning and related techniques. In particular, naive Bayes [26], probabilistic Markov models [16], k-Nearest Neighbors [10], Decision Trees [10, 26], Random Forests [10], Support Vector Machines [4,16,26,28], and (Artificial) Neural Networks [8,10] are studied.

Website Fingerprinting. The protection of the browser history is important to ensure the privacy of web users. Yet, literature offers a large spectrum of history stealing attacks that allow to recover entries of previously visited websites. Most of them are launched by malicious web servers [11, 19], but attacks have also been demonstrated on the client side in the form of malicious browser extensions [29]. If no browsing history is stored, e.g. in private browsing modes, it is still possible to detect websites a user is actively visiting. This is investigated in the field of website fingerprinting, to which we contribute with this work. A significant fraction of website fingerprinting literature is dedicated to

network traffic analysis [13, 27]. Attacks typically require an adversary to sniff network communication between the web server and the client machine. Most of the previous works tolerate encrypted traffic, e.g., generated by SSL/TLS or SSH connections, and some even work with anonymized traffic, e.g., routed over the Tor network. Other website fingerprinting approaches target the browser or the underlying operating system and typically require to execute malicious code, e.g., JavaScript, on the client machine [12, 17, 31]. Through this attack vector, Jana and Shmatikov [15] measure the memory footprint of browsers that is available through the `procfs` filesystem in Linux. The authors show that different websites exhibit different footprints and that it is possible to recover opened websites by comparing their footprints to previously recorded ones. Yet other approaches leverage properties of the hardware that runs the web browser. Attacks are typically mounted by malicious code within the browser, by other processes on the same system, or by an external adversary with physical access to the device. Oren et al. [25] demonstrate that websites exhibit different profiles in the processor cache that can be observed from JavaScript. Hornby [14] also fingerprints websites via the cache, but from another process that is running on the same processor as the web browser. Lee et al. [18] demonstrate that websites can be inferred from rendering traces that are retained in GPUs. Booth [6] demonstrates that website fingerprints can also be constructed from the CPU load. Clark et al. [9] measure the power consumption of laptop and desktop systems and attribute different power profiles to different websites. Yang et al. [33] extend this idea to mobile devices that are charged via USB.

Our Work. Similar to Hornby [14] and Lee et al. [18], we assume that a malicious application is running on the same processor as the web browser. In contrast to previous hardware based website fingerprinting, we leverage more than just the processor cache [25] or the processor load [6]. To the best of our knowledge, this work is the first that investigates hardware performance events in the context of website fingerprinting. In compliance with the state of the art in this field, we employ supervised Machine Learning techniques in the form of k-Nearest Neighbors, Decision Trees, and Support Vector Machines. While these are recognized instruments for network based fingerprinting, their application to hardware based website inference attacks is still fragmented [6, 9, 33]. In this work, we directly compare their effectiveness in multiple practical scenarios. In addition, we demonstrate that Deep Learning (in the form of Convolutional Neural Networks) outperforms traditional Machine Learning techniques that are established in both hardware performance event and website fingerprinting literature. To the best of our knowledge, CNNs have not been investigated in neither of these fields before.

3 Monitoring Hardware Performance Events

The performance monitoring unit (PMU), which is responsible for counting hardware performance events, implements a set of counters that can each be configured to count events of a certain type. The number of available events is often

considerably larger than the number of available counters. Consequently, only a limited number of events can be counted in parallel. In order to measure more events, software layers that use the PMU typically implement time multiplexing. All experiments in this work succeed by measuring only as many events as hardware counters are available, i.e., time multiplexing is not needed. Access to PMUs is typically restricted to privileged, i.e., kernel or system level code, but interfaces exist through which user space applications can gather event counts. On Unix and Linux based operating systems, `PAPI` [24] or `perf` [20] interfaces are commonly implemented. In this work, we focus on the `perf` interface that is mainly found on Linux systems. Note that this work demonstrates the general feasibility of website fingerprinting with HPEs. Therefore, similar results are also expected on systems with other HPE interfaces.

Profiling with Perf. The `perf` event monitoring subsystem was added to the Linux kernel in version 2.6.31 and subsequently made available to the user space via the `perf_event_open` system call. Listing 1.1 shows the system call signature.

```
int perf_event_open(struct perf_event_attr *attr,
                    pid_t pid, int cpu,
                    int group_fd,
                    unsigned long flags);
```

Listing 1.1: `perf_event_open` system call signature [21].

The `perf_event_attr` is the main configuration object. It determines the type of event that will be counted and defines a wide range of acquisition properties, which are documented in the Linux man-pages [21]. We focus only on a very limited number of settings and use zero values for all others. This renders our measurements to be reproducible on a larger number of systems. The `type` field in `perf_event_attr` specifies the generic event type. As we focus on hardware based events, we only use `PERF_TYPE_HARDWARE` or `PERF_TYPE_HW_CACHE`. The `config` field determines the specific event type. The event selection used in this work is given in Section 4. In addition, we set the `exclude_kernel` option, which avoids counting kernel activity. This improves the applicability of our measurement code, because kernel profiling is prohibited on some systems. Finally, the `size` field is set to the size of the event attribute struct. The `pid` and `cpu` parameters are used to set the scope of the event profiling. In this work, we focus on two profiling scenarios: *process-specific* and *core-wide*. To limit event counting to a single process, `pid` is set to the process identifier and `cpu` is set to `-1`. Subsequently, events are counted only for the given process, but on any processor core. To enable core-wide counting, `cpu` is set to the core number that should be observed and `pid` is set to `-1`. Events are then counted only on one processor core, but for all processes running on it. The `group_fd` parameter is used to signal that a selection of events belongs to a group. The `perf` system then counts all members of a group as a unit. Since this is not a strict requirement for our approach, we omit `group_fd` and set it to `-1`. The `flags` parameter is used

to configure advanced settings including the behavior when spawning new processes and monitoring Linux control groups (cgroups). As none of these settings are relevant to our measurement scenarios, we set `flags` to zero.

Once `perf_event_open` succeeds, the returned file descriptor can be used to read and reset event counts, and to enable and disable counting. In our measurements, we read event counts using the standard `read` system call. This yields a sufficiently high sampling frequency and subsequently high success rates during website fingerprinting. On our test systems, the duration of the `read` system call ranges between 1.5 μs and 3.0 μs when reading one counter value.

Access Control. On Linux, access to `perf` can be configured for user space applications. The access level is specified as an integer value that is stored in `/proc/sys/kernel/perf_event_paranoid` in the `procfs` filesystem. A negative value grants user space applications full access to performance profiling. If the `paranoid` level is set to `0`, comprehensive profiling of the kernel activity is prohibited. A value of `1` prevents user space applications from core-wide event counting (`pid = -1`, `cpu` $\geq$ `0`). A `paranoid` level of `2` prohibits process-specific event counts while the application gives control to kernel space, e.g., during a system call. Values above `2` disable event counting even in user space and effectively deactivate `perf` for user space applications. Note that the `paranoid` setting is typically overridden by applications started with the `CAP_SYS_ADMIN` capability, e.g., programs started by the root user.

4 Browser Profiling Scenarios

We investigate the inference of opened websites via HPEs in three distinct scenarios hosted on two Linux test systems. The first system features an ARM Cortex-A53 processor with six programmable hardware counters, the second one comprises an Intel i5-2430M processor with three programmable counters. Given the limited number of counters on both systems, only a selection of HPEs is measured in each experiment. A complete list of events supported by `perf` can be obtained from the Linux man-pages [21]. As we are relying on the standardized `perf_event_open` system call of the Linux kernel, there is no need to change the measurement code when switching between systems. Further details about the profiling scenarios are given in the following paragraphs.

Google Chrome on ARM. In this scenario, we profile the Google Chrome browser (v55.0.2883) with default options on the ARM system. While the browser loads websites, a malicious user space application is measuring six hardware performance events: `HW_INSTRUCTIONS`, `HW_BRANCH_INSTRUCTIONS`, `HW_CACHE_REFERENCES`, `L1_DCACHE_LOADS`, `L1_ICACHE_LOADS`, and `HW_BUS_CYCLES`. This selection of events covers instruction retirements, cache accesses, and external memory interfaces. It gives a comprehensive view of the microarchitectural load the browser is putting on the processor. The selected events are measured core-wide, hence including noise from other processes and background activity of the operating system. Since we want to assess the feasibility of core-wide profiling,

the browser process is bound to the measured processor core. The events are then measured for five seconds.

Google Chrome (Incognito) on Intel. In this scenario, we profile Google Chrome in Incognito mode with default options on the Intel system. Since the number of counters is limited to three on this processor, the malicious user space application is measuring only three events: `HW_BRANCH_INSTRUCTIONS`, `HW_CACHE_REFERENCES`, and `LLC_LOADS`. Moreover, the events are acquired in a process-specific fashion, hence the browser processes float on all processor cores. The events are then measured for one second specifically for the rendering process of the opened website. Compared to the ARM scenario, the reduced event selection still provides a meaningful view of the microarchitectural load. As the browser processes are not bound to one core anymore, we substitute events related to the L1 cache with last-level cache loads. In addition, the bus cycle event is omitted, because it is noisier on the Intel platform. Also, overall retired instructions are omitted, because we found the retired branch instructions to yield more usable information.

Tor Browser on Intel. In this scenario, we profile the Tor Browser (v6.5.1, based on Firefox v45.8.0) on the same Intel platform as before. In contrast to Chrome, the Tor Browser renders all tabs in one process, which is subsequently profiled by the malicious application. While the same three performance events are observed, the measurement duration is prolonged to 5 seconds. This is because the Tor network introduces significant delays while opening websites.

Synchronization. None of the scenarios require strict synchronization between the browser and the malicious application. Small misalignment is simply passed on to the Machine Learning step. Therefore, we only investigate simple synchronization techniques that can be achieved in practice. For Google Chrome on Intel, the adversary scans the running processes twice per second and checks whether a new rendering process has been spawned. Once a new process is detected, the adversary starts to measure the corresponding process-specific events. The Tor Browser, in contrast, is started freshly for every opened website. Again, the adversary checks all running processes twice per second and once the Tor Browser is detected, the process-specific profiling is started. This includes additional noise as the browser startup phase is also captured. In the ARM scenario, the measurements are precisely aligned with the start of loading a website. This is used to investigate whether more precise alignment yields better results. Such a trigger signal could be derived from a sudden change or characteristic pattern in the event counts, as the load of the system changes when a website is opened.

5 Machine Learning Techniques

Machine Learning provides powerful tools to automate the process of understanding and extracting relevant information from noisy observations. All of the techniques we use in this work are *supervised*, meaning that known samples

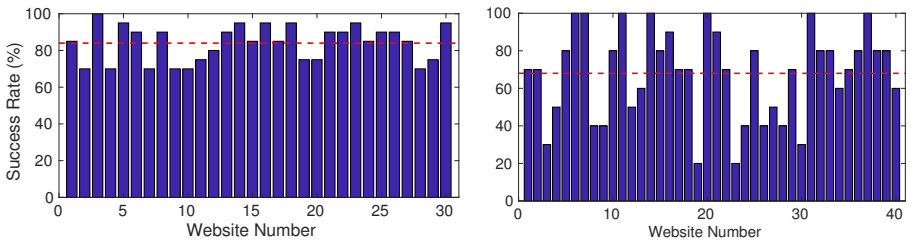
(training set) are used to derive a model that is subsequently employed to classify unknown samples (test set). The success rate of an ML technique denotes the percentage of unknown samples that are classified correctly. To reliably determine the success rate, classification is performed multiple times with different training and test sets that are derived through statistical sampling. This so-called cross-validation is performed, if the number of overall samples is low. In all our experiments, the acquired hardware performance events are concatenated to create the input data for the Machine Learning techniques. Both training and test sets are normalized to reduce the computation time. All algorithms are implemented in Matlab 2017a and run on a standard dual-core Intel processor. The training phase of the Convolutional Neural Network is reduced with the help of an NVIDIA Tesla K20 GPU accelerator. Further details of the Machine Learning techniques used in our experiments are given in the following paragraphs.

k-th Nearest Neighbor (kNN). The main goal of kNN is to find a training sample that is closest to a test sample according to the Euclidean distance. The smallest distance is taken as the first nearest neighbor and the test sample is marked with the corresponding label. In our experiments, we use the *fitcknn* command to implement kNN and to train our models. By default, the prior probabilities are the respective relative frequencies of the classes in the data, which are initially set to be equal to each other.

Decision Tree (DT). Decision Trees are used to classify samples by creating branches for given data features that yield the best split among all classes. The values for each branch are chosen such that they minimize the entropy. In our experiments, we use the *fitctree* command to train the model. The default value for maximum split is N-1, where N denotes the number of classes. For the training phase, the minimum leaf size is 1 and the minimum parent size is 10.

Support Vector Machine (SVM). In SVM based learning, input data is converted to a multi-dimensional representation by using mapping functions. Hyperplanes are then created to classify the data. The general strategy is to find the optimal decision boundaries between classes by increasing the distance between them. In our experiments, we use `libsvm` [7] to implement multi-class Support Vector Machines. The model is created and trained based on a linear SVM. We set the type of the SVM to *C-SVC*, where the parameter C is used to regularize the mapping function.

Convolutional Neural Network (CNN). In contrast to the other ML techniques, Convolutional Neural Networks automatically determine important features of the input data. This is achieved by creating nodes between higher and lower dimensional input data mappings. The meaningful features are then extracted by finding the optimal functions for each node. In our experiments, we choose two autoencoders to classify our measurements into N classes. In each autoencoder, different levels of abstraction are learned from the feature vectors and mapped to a lower dimensional space. While the number of layers in the first autoencoder is $100 \cdot N$, the second autoencoder has $10 \cdot N$ layers. The maximum number of iterations is set to 400 and L2 weight regularization is set to



(a) SVM Success Rates (b) CNN Success Rates

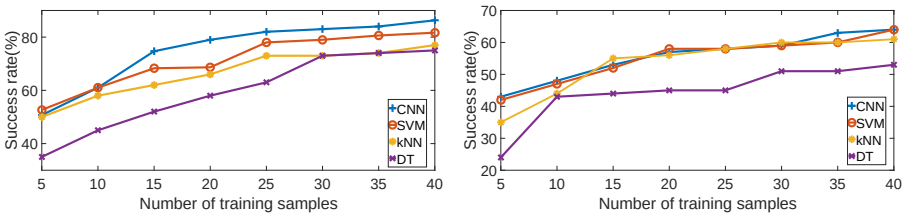
Fig. 1: Success rates per website for (a) Google Chrome on ARM with the dashed line showing an average classification rate of 84%, and (b) Tor Browser on Intel with the dashed line showing an average classification rate of 68%.

0.001 for both autoencoders. The last layer is the softmax layer. The training data is trained in a supervised fashion using labels. After the neural network is established and first classification results are obtained, the accuracy of the multilayer network model is improved using backpropagation and repeated training using labeled data. While CNNs have many advantages, the most important disadvantage is their memory demands. When we run out of GPU memory, we downsample the input data to reduce the length of the feature vectors.

6 Website Profiling Results

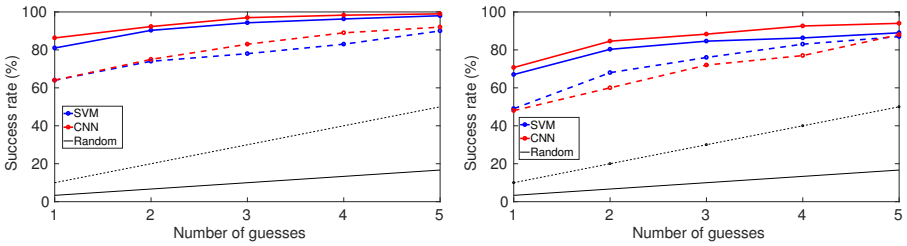
In each of the profiling scenarios described in Section 4, we monitor events when loading the homepages of the most visited websites according to Alexa [1]. The tested websites, excluding adult ones, is listed in Appendix A (1-30). This illustrates the general effectiveness of the Machine Learning techniques to classify websites based on HPEs. To demonstrate that also fine-grained profiling is feasible, 10 different sub-pages of the `Amazon.com` domain are monitored in Google Chrome on Intel. Finally, a selection of whistleblowing websites is measured when visited with the Tor browser. They are also given in Appendix A (31-40).

Google Chrome on ARM. For the experiments on ARM, each website is monitored 20 times to train the models. For each of these visits, 25,000 samples are acquired per hardware performance event. The samples of all events are then concatenated to yield a final measurement size of 150,000 samples. For 30 websites, the total training data size is therefore $90 \cdot 10^6$ samples. Based on this training set, the success rates after cross-validation are 84% for linear SVM, 80% for kNN, and less than 50% for DT and CNN. The low success rates of DT and CNN indicate that not enough samples have been acquired. Figure 1(a) illustrates the success rates for each of the visited websites when classified with SVM. Since the number of samples collected in this scenario is small, 10-fold cross-validation is used. The lowest detection rate is 70%, which shows that core-wide profiling is feasible even in the presence of background noise. The average classification rate of 84% is shown as a dashed line in the figure.



(a) Alexa Top 30 (b) Same Domain Pages

Fig. 2: Success rate vs. number of training measurements for Google Chrome (Incognito), and (a) 30 different websites (b) 10 same domain web pages.

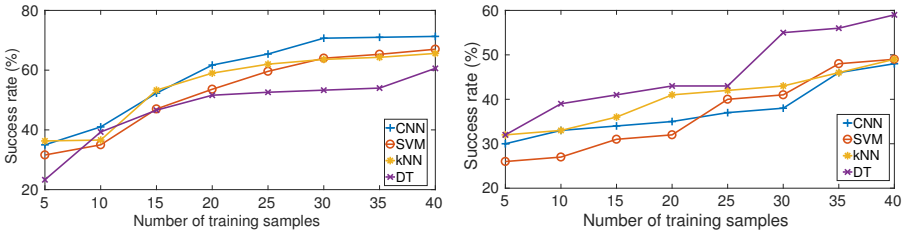


(a) Google Chrome (Incognito) (b) Tor Browser

Fig. 3: Number of guesses vs. classification rate for (a) Google Chrome (Incognito), and (b) Tor Browser. Solid lines represent results for Alexa Top 30, while the dashed lines illustrate the same domain results.

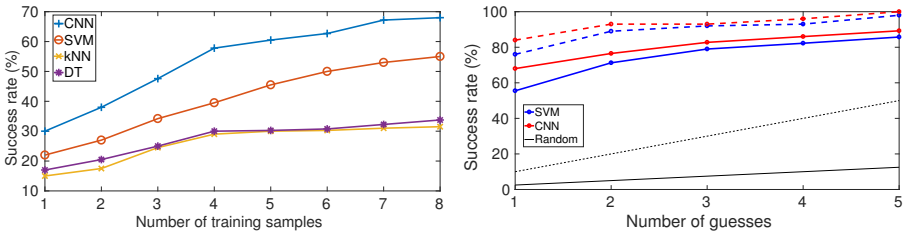
Google Chrome (Incognito) on Intel. For the Google Chrome experiments on Intel, the number of measurements per website is increased to 50. As more samples are acquired, fixed training and test sets are derived instead of using cross-validation. Out of the 50 observations, 40 are used for the training phase whereas 10 are collected to test the derived models. Since each website is monitored for only one second, every measurement now consists of 10,000 samples per event. With three observed events, this yields a total training set size of $36 \cdot 10^6$ and a test set size of $9 \cdot 10^6$ samples.

Figure 2(a) shows the success rates over an increasing number of training measurements for all Machine Learning techniques. Clearly, CNN achieves the highest classification rate, if enough training samples are available. In particular, the success rate for 40 training observations per website is 86.3%. If the training data size is small, SVM and kNN achieve similar success rates as CNN. Due to the large size of feature vectors in the training and test data, DT gives lower success rates than other ML techniques. Regarding the computational effort, the training phase of CNN takes 2 hours on a GPU and is consequently the longest among the Machine Learning techniques. In contrast, the test phase takes approximately 1 minute for every ML technique.



(a) Alexa Top 30 (b) Same Domain Pages

Fig. 4: Success rate vs. number of training measurements for the Tor Browser and (a) 30 different websites, or (b) 10 same domain web pages.



(a) All Websites (b) Whistleblowing vs. Others

Fig. 5: (a) Success rate vs. number of training measurements for Tor Browser and all websites. (b) Number of guesses vs. classification rate for whistleblowing (dashed) and all websites (solid).

The second experiment for Google Chrome in Incognito mode on Intel assumes that an adversary has detected a website that the user has visited. Consequently, the attacker tries to infer which page of the website the user is interested in. To illustrate the feasibility of this attack, we selected 10 pages of the `Amazon.com` domain that display different sections of the online store (kitchen, bedroom, etc.). Naturally, this scenario is more challenging, as the difference between web pages of the same domain is smaller than for entirely different websites. Nevertheless, it is still possible to correctly classify the visited web pages with moderate success. This is illustrated in Figure 2(b). When using CNN and SVM, the success rate is 64%. kNN yields 60% success rate, while DT drops to 52%. For CNN and SVM, we also investigate the success rates when the number of guesses is increased in Figure 3(a). If the first 5 result classes are considered, websites can be detected with 99% accuracy for SVM and CNN. Similar results are obtained for the same domain experiments, where both CNN and SVM yield 92% accuracy.

Tor Browser on Intel. For the Tor Browser experiments, the same events as before are observed as well as the same number of measurements are taken for each website. Again, 40 of those measurements are used to construct the training set, while 10 measurements form the test set. As the Tor Browser is monitored

for 5 seconds, 50,000 samples are acquired for each event and website. This yields 150,000 samples for one measurement, $180 \cdot 10^6$ samples for the entire training set, and $45 \cdot 10^6$ samples for the test set.

Figure 4(a) shows the success rates over an increasing number of training measurements for all Machine Learning techniques. CNN yields the highest success rate of 71%. While SVM and kNN have similar success rates around 66%, Decision Tree yields a lower accuracy of 60%. The results show that CNN can handle noisy data and misalignment problems better than other methods, since CNN learns the relations between traces. The experiment results for the 10 web pages on `Amazon.com` are illustrated in Figure 4(b). In contrast to the Google Chrome results, Decision Tree yields the highest success rate of 59%. We believe the reason is the small number of classes that increases the efficiency of DT. The remaining algorithms classify the same domain web pages with a similar success rate of approximately 49%. In addition, Figure 3(b) shows the success rates for CNN and SVM over an increasing number of guesses. While the random selection success rate is around 16% for 5 guesses, CNN achieves a success rate of 94%. For the same domain web pages, the success rate of CNN is 88% for 5 guesses. SVM achieves slightly worse results.

Finally, we investigate whistleblowing websites, as visiting them anonymously is an important reason to use the Tor Browser. For this experiment, we select 10 whistleblowing portals from [2] (also given in Appendix A). In the first step, these websites are classified using all ML techniques. While CNN yields the best classification rate of 84%, SVM exhibits a success rate of 78%. In contrast, DT and kNN have lower success rates around 60%. In the second step, the classification is repeated for all websites considered so far (whistleblowing and Alexa Top 30). Figure 5(a) illustrates the success rates for all ML techniques. When classifying 40 websites, CNN yields a success rate of 68%, while SVM achieves 55%. In contrast, kNN and DT algorithms cannot classify the websites effectively. When the number of guesses is increased, the success rate improves again. Figure 5(b) shows the classification rates over an increasing number of guesses. If only whistleblowing websites and 5 guesses are considered, CNN yields a success close to 100%. When all websites are considered, the success rate of CNN is 89.25%. SVM achieves slightly worse results. Individual success rates for CNN are shown in Figure 1(b). The lowest success rate is around 20% for two websites and seven websites are classified correctly with 100% accuracy. An interesting observation is that among the 40 websites, the whistleblowing portals are still classified with good success rates. With an average success rate of 68%, CNN is more capable than other ML techniques to correctly classify websites opened in Tor browser.

7 Discussion

The experiments on ARM were conducted with core-wide measurements, whereas HPEs were acquired in a process-specific fashion on Intel. In general, core-wide acquisition is expected to introduce more noise in the measurements, e.g., from system activity in the background. For process-specific acquisition the activity of

the rest of the system does not impair the measurements, as the `perf` subsystem accumulates event counts only when the specified process is running. According to the results presented in the previous section, however, both scenarios allow to classify websites with success rates of over 80% for SVM. This is because the test systems were mostly idling during the experiments. If this is not the case, the effects of increased system load can be countered by increasing the profiling time in order to obtain more information from the renderer process. Furthermore, an adversary can train multiple models for multiple levels of system load to account for an unknown load of the target system. Also, a slight increase of the number of guesses yields a significant increase in classification success. These measures also help against other impairments of the measurement quality, e.g., if multiple websites are opened in parallel in the Tor browser, if direct URLs are used instead of visiting the homepage of a website, if the number of profiled websites grows, or if websites are visited that have not been profiled and cannot be classified as a consequence.

Compared to Google Chrome in Incognito mode, the results of the Tor Browser are worse on average. This can be explained with the browser start-up phase, which is always captured for Tor. Also, random network delays introduce jitter in the observations of the website loading. Large delays require to prolong the event acquisition phase, otherwise the success rates are expected to drop. Another adverse effect is the changing geo-location of the Tor exit nodes. Many websites, particularly news sites like New York Times and Yahoo, customize their appearance based on the location of their visitors and therefore introduce additional noise in the measurements. Similar effects can occur if websites contain personalized advertisements and other frequently changing content. While this potentially decreases the success rates, we believe that an important part of the profiling relies on the website templates.

Among the Machine Learning techniques, Convolutional Neural Networks have proven to be the most capable for classifying websites, if enough samples are available. This is the reason why CNNs performed well in Google Chrome and Tor Browser experiments, but not in ARM experiments. CNNs are built for multi-classification of complex structures by extracting meaningful features. On the contrary, SVM and kNN are designed to create hyperplanes to separate space into classes. Since the number of dimensions is high in the experiments, it is difficult to find the best hyperplane for each dimension. Nevertheless, there is still a need for further studies on CNN, since the results could be improved by modifying the parameters, number of layers and neurons.

In general, the feasibility of website fingerprinting via hardware performance events is not limited to the specific profiling scenarios and test platforms used in our experiments. This is because of the fundamental phenomenon that loading different websites creates different microarchitectural footprints. This a logical consequence of optimized software that is designed to provide best user experience. Therefore, similar results are expected also for other x86 and ARM processors, as well as for other HPE interfaces and web browsers, unless mitigation strategies are implemented.

8 Countermeasures

The website inference technique presented in this work requires that i) websites loaded by a browser exhibit a unique footprint in the microarchitectural state of a processor, and that ii) this state can be observed via hardware performance events with sufficient precision. Any efforts impacting these two requirements directly affect the reliability, success, or practicality of our approach. The following two paragraphs discuss such efforts and formulate possible countermeasures.

Displaying Websites. The first requirement of our classification technique implies that the executed operations during downloading and rendering of website elements are closely related to the type and amount of content displayed on a website. From a more abstract perspective this means that the execution flow and memory accesses of the browser vary for different websites. A thorough approach for solving this issue is writing code such that instruction sequences and operand addresses are independent of the input that is processed. While this is reasonable to aspire for security software, it has considerable practical drawbacks in the context of web browsers. First, removing input dependencies almost always impairs performance, because runtime optimizations typically rely on skipping operations and handling special cases differently. As a result, websites take longer to display, which is not in favor of user experience. Second, the larger the code, the more complex it gets to remove input dependencies. For web browsers, at least the code related to networking, storing, and rendering of elements must be changed. Given that security critical software has much smaller code bases and still struggles to remove input dependencies in practice, it is questionable that browser software will successfully implement this in the foreseeable future. If input dependencies cannot be entirely removed, artificial noise can be added to the website loading process. This is, for instance, achieved by introducing random delays between operations, adding functions that process dummy data instead of real inputs, or randomly loading unrelated website elements in the background. While this does not solve the underlying problem, it distorts the microarchitectural footprint each website exhibits while being displayed.

Observing Events. The second requirement is the ability to observe the state of the processor microachitecture with high precision. Since performance monitoring units are dedicated parts of the processor, they cannot simply be removed or permanently deactivated. However, operating systems can block access to them from the software side. On Linux, the kernel can be compiled without the `perf` subsystem, e.g., by disabling the `CONFIG_PERF_EVENTS` configuration option. Also, the `perf_event_paranoid` file can be set to `3` or above to disable event counter access from user space. However, blocking or deactivating `perf` impairs applications that use performance events for legitimate profiling or debugging purposes. If event counting is generally needed, a possible compromise could be more fine-grained profiling restrictions, such that processes can only count events caused by themselves. Profiling any other process is prohibited, even if it belongs to the same user. While this requires changes to the `perf` interface, it provides legitimate applications access to profiling and at the same

time impairs the fingerprinting technique presented in this work. This profiling restriction could be added as a dedicated setting in the `perf_event_paranoid` configuration file. An alternative solution is to lower the measurement precision of hardware performance events. This can, for instance, be achieved by artificially adding a certain level of noise to the event counts while retaining a sufficiently high signal-to-noise ratio, or by reducing sampling frequencies with which applications can acquire event counts. Yet again, this would also affect benign applications. A possible solution is to detect malicious programs and then only degrade their observations. However, the presented measurement approach behaves identically to legitimate applications and does not rely on exotic operations or measurement settings.

9 Conclusion

When websites are loaded in the browser, they stress the underlying hardware in a distinct pattern that is closely related to the contents of the website. This pattern is reflected in the microarchitectural state of the processor that executes the browser, which can be observed with high precision by counting hardware performance events. Since these events can be legitimately measured by user space applications, it is feasible to infer opened websites via performance event measurements. We demonstrate this by utilizing Machine Learning techniques and achieve high recognition rates even in the presence of background noise, trace misalignment, and varying network delays. The results show that CNN is able to obtain better classification rates from high number of classes in the presence of noise. By applying CNN, the whistleblowing websites are classified with 79% accuracy among 40 websites while the overall classification rate increases up to 89.25% with 5 guesses in Tor browser. Yet, further work is necessary to investigate practical aspects that go beyond the proof of concept presented in this work. Since many websites change between visits and users, e.g., due to daily news or personalized ads, it is necessary to systematically analyze how geolocation, personalization, and aging affect the classification accuracy. Another direction is investigating the contribution of each website element to the microarchitectural fingerprint and subsequently to the classification success rates. This is especially useful for countermeasures, which could then handle critical elements more carefully.

Acknowledgments. We would like to thank the anonymous reviewers for their valuable comments and suggestions. This work has been supported by the National Science Foundation, under grants CNS-1618837 and CNS-1314770.

References

1. Alexa Internet Inc.: The top 500 sites on the web (2017), `http://www.alexa.com/topsites`. Last accessed: 2017-05-10.
2. Anonymous Contributors: Leak site directory, `http://www.leakdirectory.org/index.php/Leak_Site_Directory`

3. Atici, A., Yilmaz, C., Savas, E.: An approach for isolating the sources of information leakage exploited in cache-based side-channel attacks. In: Software Security and Reliability-Companion (SERE-C), 2013 IEEE 7th International Conference on. pp. 74–83 (June 2013)
4. Bahador, M.B., Abadi, M., Tajoddin, A.: Hpcmalhunter: Behavioral malware detection using hardware performance counters and singular value decomposition. In: 2014 4th International Conference on Computer and Knowledge Engineering (ICCKE). pp. 703–708 (Oct 2014)
5. Bhattacharya, S., Mukhopadhyay, D.: Who watches the watchmen?: Utilizing performance monitors for compromising keys of rsa on intel platforms. In: Güneysu, T., Handschuh, H. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2015: 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings. pp. 248–266. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
6. Booth, J.: Not So Incognito: Exploiting Resource-Based Side Channels in JavaScript Engines. Master's thesis, School of Engineering and Applied Sciences, Harvard University (2015), http://nrs.harvard.edu/urn-3:HUL.InstRepos:17417578
7. Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology 2, 27:1–27:27 (2011), software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm
8. Chiappetta, M., Savas, E., Yilmaz, C.: Real time detection of cache-based side-channel attacks using hardware performance counters. Applied Soft Computing 49, 1162 – 1174 (2016)
9. Clark, S.S., Mustafa, H., Ransford, B., Sorber, J., Fu, K., Xu, W.: Current events: Identifying webpages by tapping the electrical outlet. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) Computer Security – ESORICS 2013: 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings. pp. 700–717. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
10. Demme, J., Maycock, M., Schmitz, J., Tang, A., Waksman, A., Sethumadhavan, S., Stolfo, S.: On the feasibility of online malware detection with performance counters. In: Proceedings of the 40th Annual International Symposium on Computer Architecture. pp. 559–570. ISCA '13, ACM, New York, NY, USA (2013)
11. Felten, E.W., Schneider, M.A.: Timing attacks on web privacy. In: Proceedings of the 7th ACM Conference on Computer and Communications Security. pp. 25–32. CCS '00, ACM, New York, NY, USA (2000)
12. Gruss, D., Bidner, D., Mangard, S.: Practical memory deduplication attacks in sandboxed javascript. In: Pernul, G., Y A Ryan, P., Weippl, E. (eds.) Computer Security – ESORICS 2015: 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part I. pp. 108–122. Springer International Publishing, Cham (2015)
13. Hayes, J., Danezis, G.: k-fingerprinting: A robust scalable website fingerprinting technique. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 1187–1203. USENIX Association, Austin, TX (2016), https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/hayes
14. Hornby, T.: Side-channel attacks on everyday applications: Distinguishing inputs with flush+reload. Black Hat USA (2016), https://www.blackhat.com/docs/us-16/materials/us-16-Hornby-Side-Channel-Attacks-On-Everyday-Applications-wp.pdf
15. Jana, S., Shmatikov, V.: Memento: Learning secrets from process footprints. In: 2012 IEEE Symposium on Security and Privacy. pp. 143–157 (May 2012)

16. Kazdagli, M., Reddi, V.J., Tiwari, M.: Quantifying and improving the efficiency of hardware-based mobile malware detectors. In: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). pp. 1–13 (Oct 2016)
17. Kim, H., Lee, S., Kim, J.: Inferring browser activity and status through remote monitoring of storage usage. In: Proceedings of the 32Nd Annual Conference on Computer Security Applications. pp. 410–421. ACSAC '16, ACM, New York, NY, USA (2016)
18. Lee, S., Kim, Y., Kim, J., Kim, J.: Stealing webpages rendered on your browser by exploiting gpu vulnerabilities. In: 2014 IEEE Symposium on Security and Privacy. pp. 19–33 (May 2014)
19. Liang, B., You, W., Liu, L., Shi, W., Heiderich, M.: Scriptless timing attacks on web browser privacy. In: 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. pp. 112–123 (June 2014)
20. Linux Kernel Developers: perf: Linux profiling with performance counters (2015), `https://perf.wiki.kernel.org/index.php/Main_Page`
21. Linux Programmer's Manual: perf_event_open - set up performance monitoring (2016), `http://man7.org/linux/man-pages/man2/perf_event_open.2.html`
22. Lipp, M., Gruss, D., Spreitzer, R., Maurice, C., Mangard, S.: Armageddon: Cache attacks on mobile devices. In: 25th USENIX Security Symposium (USENIX Security 16). USENIX Association, Austin, TX (Aug 2016)
23. Maurice, C., Le Scouarnec, N., Neumann, C., Heen, O., Francillon, A.: Reverse engineering intel last-level cache complex addressing using performance counters. In: Research in Attacks, Intrusions, and Defenses: 18th International Symposium, RAID 2015, Kyoto, Japan,November 2-4, 2015. Proceedings. pp. 48–65. Springer International Publishing, Cham (2015)
24. Mucci, P.J., Browne, S., Deane, C., Ho, G.: Papi: A portable interface to hardware performance counters. In: In Proceedings of the Department of Defense HPCMP Users Group Conference. pp. 7–10 (1999)
25. Oren, Y., Kemerlis, V.P., Sethumadhavan, S., Keromytis, A.D.: The spy in the sandbox: Practical cache attacks in javascript and their implications. In: Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security. pp. 1406–1418. CCS '15, ACM, New York, NY, USA (2015)
26. Singh, B., Evtyushkin, D., Elwell, J., Riley, R., Cervesato, I.: On the detection of kernel-level rootkits using hardware performance counters. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. pp. 483–493. ACM (2017)
27. Sun, Q., Simon, D.R., Wang, Y.M., Russell, W., Padmanabhan, V.N., Qiu, L.: Statistical identification of encrypted web browsing traffic. In: Proceedings 2002 IEEE Symposium on Security and Privacy. pp. 19–30 (2002)
28. Tang, A., Sethumadhavan, S., Stolfo, S.: Unsupervised anomaly-based malware detection using hardware features. In: Stavrou, A., Bos, H., Portokalidis, G. (eds.) Research in Attacks, Intrusions and Defenses, Lecture Notes in Computer Science, vol. 8688, pp. 109–129. Springer International Publishing (2014)
29. Ter Louw, M., Lim, J.S., Venkatakrishnan, V.N.: Enhancing web browser security against malware extensions. Journal in Computer Virology 4(3), 179–195 (2008)
30. Uhsadel, L., Georges, A., Verbauwhede, I.: Exploiting hardware performance counters. In: Fault Diagnosis and Tolerance in Cryptography, 2008. FDTC '08. 5th Workshop on. pp. 59–67 (Aug 2008)
31. Vila, P., Köpf, B.: Loophole: Timing attacks on shared event loops in chrome. In: 26th USENIX Security Symposium (USENIX Security 17). USENIX

Association, Vancouver, BC (2017), https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/vila
32. Willems, C., Hund, R., Fobian, A., Felsch, D., Holz, T., Vasudevan, A.: Down to the bare metal: Using processor features for binary analysis. In: Proceedings of the 28th Annual Computer Security Applications Conference. pp. 189–198. ACSAC '12, ACM, New York, NY, USA (2012)
33. Yang, Q., Gasti, P., Zhou, G., Farajidavar, A., Balagani, K.S.: On inferring browsing activity on smartphones via usb power analysis side-channel. IEEE Transactions on Information Forensics and Security 12(5), 1056–1066 (May 2017)
34. Zankl, A., Miller, K., Heyszl, J., Sigl, G.: Towards efficient evaluation of a time-driven cache attack on modern processors. In: Askoxylakis, I., Ioannidis, S., Katsikas, S., Meadows, C. (eds.) Computer Security – ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part II. pp. 3–19. Springer International Publishing, Cham (2016)
35. Zhang, N., Sun, K., Shands, D., Lou, W., Hou, Y.T.: Truspy: Cache side-channel information leakage from the secure world on arm devices. Cryptology ePrint Archive, Report 2016/980 (2016), http://eprint.iacr.org/2016/980

A List of Profiled Websites

Table 1: Profiled websites. Entries 1-30 are taken from the top websites listed by Alexa [1], while URLs 31-40 are a selection of whistleblowing portals from [2].

Website Numbers and URLs		
1) Netflix.com	15) Bing.com	29) Espn.com
2) Amazon.com	16) Imgur.com	30) Wikia.com
3) Facebook.com	17) Ntd.tv	31) Wikileaks.org
4) Google.com	18) Cnn.com	32) Aljazeera.com/ investigations
5) Yahoo.com	19) Pinterest.com	
6) Youtube.com	20) Tumblr.com	33) Balkanleaks.eu
7) Wikipedia.org	21) Office.com	34) Unileaks.org
8) Reddit.com	22) Microsoftonline.com	35) Globaleaks.com
9) Twitter.com	23) Chase.com	36) Liveleak.com
10) Ebay.com	24) Nytimes.com	37) Globalwitness.org
11) Linkedin.com	25) Blogspot.com	38) Wikispooks.com
12) Diply.com	26) Paypal.com	39) Officeleaks.com
13) Instagram.com	27) Imdb.com	40) Publeaks.nl
14) Live.com	28) Wordpress.com	