Red Alert for Power Leakage: Exploiting Intel RAPL-Induced Side Channels

Zhenkai Zhang
Texas Tech University
Lubbock, TX, USA
zhenkai.zhang@ttu.edu

Sisheng Liang
Texas Tech University
Lubbock, TX, USA
sisheng.liang@ttu.edu

Fan Yao
University of Central Florida
Orlando, FL, USA
fan.yao@ucf.edu

Xing Gao
University of Delaware
Newark, DE, USA
xgao@udel.edu

ABSTRACT

RAPL (Running Average Power Limit) is a hardware feature introduced by Intel to facilitate power management. Even though RAPL and its supporting software interfaces can benefit power management significantly, they are unfortunately designed without taking certain security issues into careful consideration. In this paper, we demonstrate that information leaked through RAPL-induced side channels can be exploited to mount realistic attacks. Specifically, we have constructed a new RAPL-based covert channel using a single AVX instruction, which can exfiltrate data across different boundaries (e.g., those established by containers in software or even CPUs in hardware); and, we have investigated the first RAPL-based website fingerprinting technique that can identify visited webpages with a high accuracy (up to 99% in the case of the regular network using a browser like Chrome or Safari, and up to 81% in the case of the anonymity network using Tor). These two studies form a preliminary examination into RAPL-imposed security implications. In addition, we discuss some possible countermeasures.

CCS CONCEPTS

• **Security and privacy → Systems security**; **Network security**; **Side-channel analysis and countermeasures**.

KEYWORDS

RAPL, side-channel attack, covert channel, website fingerprinting

ACM Reference Format:
Zhenkai Zhang, Sisheng Liang, Fan Yao, and Xing Gao. 2021. Red Alert for Power Leakage: Exploiting Intel RAPL-Induced Side Channels. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (ASIA CCS '21), June 7–11, 2021, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3433210.3437517

1 INTRODUCTION

The power consumption of a computer system has been considered increasingly critical over the past few decades. To facilitate power management, Intel has introduced a hardware feature named Running Average Power Limit (RAPL) since its Sandy Bridge microarchitecture [30]. RAPL enables accurate energy consumption measurement and provides fine-grained power capping capability. Thus, by monitoring and reacting to the power consumption of computing, RAPL has been widely used in data centers to improve energy efficiency and enforce power budget compliance [13, 54, 73].

Several model-specific registers (MSRs) in Intel processors are dedicated to RAPL manipulation [24]. To make the use of RAPL easier, since the kernel version 3.14, Linux has added a power capping framework, which exposes the functionality of RAPL to the user space through a *sysfs* interface called *powercap* [61]. Although unprivileged users cannot change the RAPL settings through this interface, they are allowed to read all the files belonging to it, including the ones that report the energy in μJ consumed by different parts of the system. While such an interface does not exist in other operating systems, unprivileged users may still be able to read RAPL by invoking specific system calls like on Mac OS.

Even though RAPL and its supporting software abstraction can benefit power management significantly, they are designed without considering certain underlying security issues such as information leakage. Because energy is a fundamental resource needed for any computational activity, its consumption measurements inevitably carry some information about the activity. The question is whether the energy consumption information exposed by RAPL can be exploited to help or mount realistic attacks? If so, how?

Surprisingly, only little prior work has studied such questions [41, 48]. Given the fact that there are still many blanks left, this paper makes an attempt to fill in some of them by presenting two possible security implications imposed by RAPL. Specifically, the studies conducted in this paper demonstrate that we can leverage RAPL to construct covert communication channels and track website visits. These studies and their results support the already drawn conclusion that software/hardware designs, useful but oblivious to security risks like information leakage, are still common in a system. The main contributions of this paper include:

- We investigate the levels of memory power consumption under different intensive DRAM access scenarios. Based on the investigation results, we show the feasibility of exploiting RAPL to construct covert channels.
- We evaluate the proposed covert channel on several platforms to demonstrate its effectiveness on crossing isolation at different logical levels such as containers in software and CPUs in hardware. We further evaluate its robustness to continuous cache eviction noise.
- We demonstrate the possibilities of using RAPL readings to fingerprint websites. Unlike the previous work on using power traces for website fingerprinting which requires external instrumentation and/or hardware modification, the RAPL-based technique is purely in software and thus poses

ASIA CCS '21, June 7–11, 2021, Hong Kong, Hong Kong

ACM ISBN 978-1-4503-8287-8/21/06…$15.00
https://doi.org/10.1145/3433210.3437517

- more practical threats to many popular systems (e.g., Apple Macs).
- We evaluate the RAPL-based website fingerprinting technique using popular browsers like Chrome, Safari, and Tor on Linux and Mac OS. The results show that this fingerprinting method is highly effective – in terms of the regular network, up to 99% accuracy can be achieved; in terms of the anonymity network, up to 81% accuracy can be achieved.

The rest of this paper is organized as: Section 2 describes the necessary background; Section 3 presents how to exploit RAPL to construct a covert channel; Section 4 investigates if RAPL can be exploited to achieve website fingerprinting; Section 5 discusses the possible mitigation removing RAPL-induced side channels; Section 6 gives the related work; and Section 7 concludes this paper and states some future work.

2 BACKGROUND

In this section, we present necessary background information on Intel RAPL and on how to access this hardware feature. We emphasize the *powercap* interface in Linux and the diagnostic system call in Mac OS. In addition, we discuss what has been studied in prior work on its security implications.

2.1 Intel Running Average Power Limit

Since Sandy Bridge, Intel has introduced RAPL to facilitate power management [4, 53]. Because RAPL samples energy consumption at a fine granularity and at a high frequency, it enables us to monitor the consumed energy very accurately. More importantly, RAPL can adjust and optimize performance to satisfy the given power budget constraints.

RAPL is split into several power domains, and each of them has its own capabilities of measuring the consumed energy in the domain and specifying the power limit of the domain. The currently supported power domains are as follows [24, 30]:

- Package domain. This domain manages the entire CPU socket, which includes all the processor cores as well as the uncore components (e.g., the last-level cache, integrated GPU, and memory controller).
- Power plane 0 domain (PP0). This domain is in charge of all the processor cores on the socket, namely, it is a sub-domain of the package domain.
- Power plane 1 domain (PP1). This domain is in charge of the integrated GPU on the socket, namely, it is also a sub-domain of the package domain.
- DRAM domain. This domain is associated with the DRAM memory attached to the integrated memory controller. (Although it is treated as a sub-domain of the package domain in the Linux *powercap*, the energy consumption it manages however cannot be governed by the package domain.)
- Platform domain (PSys). This domain deals with the whole CPU package, memory, and other devices that are powered directly by the integrated power delivery mechanism.

Given the hierarchical structure of the power domains (e.g., PP0 is a sub-domain of the package), it is possible to manage a set of devices together using the parent power domain, and if more fine-grained control is needed, it can be applied through the sub-domains.

Notice that not every processor supports all these power domains. For instance, server processors usually do not have an integrated GPU, and thus they do not have a valid PP1 domain. Another example is that the PSys domain is not introduced until Skylake; moreover, unlike the other four power domains, to be valid, the PSys domain needs additional support from the platform hardware and BIOS, which does not exist in many systems equipped with Skylake or its successors; and hence we do not consider the PSys domain in the following of this paper. Table 1 lists several relatively new Intel microarchitectures and marks which domains exist in their mainstream consumer-grade/server-grade processors.

Table 1: RAPL power domains supported in some recent Intel microarchitectures (consumer-grade/server-grade).

Microarchitecture	Package	PP0	PP1	DRAM
Haswell	Y/Y	Y/N	Y/N	Y/Y
Broadwell	Y/Y	Y/N	Y/N	Y/Y
Skylake	Y/Y	Y/Y	Y/N	Y/Y
Kaby Lake	Y/Y	Y/Y	Y/N	Y/Y

Several MSRs are associated with each power domain. One of such MSRs is an “energy status” register which reports the total amount of energy consumed since the last time when this register is cleared. This status MSR is updated approximately every 1 ms. (For Skylake and newer processors, the PP0 energy status MSR is updated much faster, e.g., at the level of tens of microseconds.) Another important one of the associated MSRs is a “power limit” register which specifies a limit on the power consumption as an average over a time window. The units of energy, power, and time may be different in different processors. For example, the energy unit in Sandy Bridge processors is of 15.3 μJ, and that in Haswell processors is of 61 μJ. These units can be obtained from a read-only “power unit” MSR for the whole RAPL.

Aside from directly using privileged `rdmsr` and `wrmsr` instructions, there are several other ways to access RAPL under Linux. The first approach is to read/write RAPL-associated MSRs through the `/dev/cpu/<cpu#>/msr` files provided by the Linux *msr* driver, but access to these files needs root privileges. The second approach is to use the Linux *perf_event* interface to only obtain energy consumption readings from RAPL, but it also needs intervention from the root. The third approach is to leverage the Linux *powercap* interface, which allows unprivileged users to directly read RAPL.

Under Mac OS, RAPL energy status MSRs can be accessed via its diagnostic system call. The diagnostic system call has a specific mode for inquiring the power statistics of the system backed by RAPL. Thus, any unprivileged process can make this system call to read RAPL. Note that there is no available interface on Windows by default for accessing RAPL in the unprivileged manner.

2.2 Linux Power Capping Framework

Back in January 2014, the Linux kernel 3.13 release introduced a new feature called the power capping framework. This framework is designed around Intel RAPL and provides a consistent interface in the *sysfs* named *powercap*. The path to the *powercap* interface is `/sys/class/powercap/intel-rapl/`. To avoid cluttering, we use `<sys-rapl>` to denote this path in the following. The files under the *powercap* interface expose the functionality of RAPL to the user space in a uniform way.

For each valid power domain, there are several files associated with it under the *powercap* interface. Some of the files are for setting average power consumption constraints over time windows, but they can be written only by a privileged process. The other files are read-only and for giving information, e.g., the domain name in the *name* file, and reporting the current energy consumption in the *energy_uj* file. The *energy_uj* file is updated about every 1 ms. By default, all the files under the *powercap* interface are world-readable.

The files under the *powercap* interface are organized hierarchically to reflect the tree topology of the power domains. For example, the files associated with the package domain are located under `<sys-rapl>/intel-rapl:0/`, where `0` indicates the first CPU socket, while the files belonging to the PP0 domain are located under `<sys-rapl>/intel-rapl:0/intel-rapl:0:0/`, where the sub-directory `intel-rapl:0:0` signifies the first sub-domain. If there is a second CPU socket that also has a processor, the files associated with its package domain are under `<sys-rapl>/intel-rapl:1/`, and so forth.

Gao *et al.* have found that many container-based platform-as-a-service (PaaS) clouds allow the tenanted containers to pry the underlying RAPL readings through their own *powercap* interface [9]. They exploited this fact to identify co-residence, and also mentioned the possibility of utilizing it to construct covert channels. In this paper, we advance their study by implementing an RAPL-based covert channel and also present other RAPL exploitation venues. Even though, prior to our work, Paiva *et al.* have created a covert channel over RAPL [48], they simply used cache eviction and did not consider eviction noise made by other tasks. Our covert channel does not rely on evicting the cache hierarchy and performs well in the presence of eviction noise.

2.3 OS X Diagnostic System Call

There are four classes of system calls exposed by the kernel of Mac OS X, which are BSD, Mach, machine-dependent, and diagnostic [36]. The BSD and Mach system calls are interfaces to the BSD and Mach kernel parts on which the kernel of Mac OS X is built. The machine-dependent system calls are used for CPU specific features, but many of them are undefined for the x86-64 architecture.

The diagnostic class consists of only one system call named `diagCall64()` that is used exclusively for diagnostics. The mode of `diagCall64()` is determined by its first argument, and one of the modes (that is #17) is to access RAPL energy status MSRs. To invoke `diagCall64()` to read energy status MSRs, the `syscall` instruction is used[1] with the system call number 0x4000001 in the `%rax` register, the mode number 17 in the `%rdi` register, and the address of an energy statistics data structure in the `%rsi` register. After the system call, the energy statistics data structure will be filled with the readings of the RAPL MSRs. An example of using this system call to read RAPL can be found in the Firefox code base[2].

3 CONSTRUCTING COVERT CHANNELS

We have conducted two studies on RAPL-imposed security implications, and the first one is to create an RAPL-based covert channel for data exfiltration. The key observation is that RAPL has a shared nature and keeps track of the overall energy consumption in each power domain. In other words, RAPL does not have the concept of partitioning and it is influenced by any computation disregarding the upper-level security boundaries. The other important property is that RAPL is accessible to users by default in many environments. As mentioned in Section 2, unprivileged native processes in a Linux system can use the *powercap* interface to read RAPL, and so can containers hosted by many public container-based PaaS clouds [9][3]. Even in the scenarios of conventional virtual machine (VM) based infrastructure-as-a-service (IaaS) clouds, some providers do not prohibit the guest VMs from inquiring the values in the RAPL energy status MSRs. In this section, we specifically leverage the energy consumption measurements in the DRAM domain to construct an RAPL-based covert channel. Before proceeding with the details, we discuss the rationale for focusing on the DRAM power domain.

First of all, as shown in Table. 1, the DRAM domain is widely supported by mainstream Intel CPUs currently in use. Yet, more importantly, the noise generated by the background computation in the DRAM power domain can be much smaller than that in other commonly existing domains, for which there are two main reasons: (1) Compared to the package power domain, the DRAM domain is specific to only the integrated memory controller, and in contrast to the PP0 power domain, the DRAM domain is sensitive to only load/store instructions; (2) Although other computational tasks may be able to generate a large amount of memory accesses, the existence of large, multi-level caches in a processor can substantially reduce the DRAM access demands such that the signals created for covert communication will not be easily buried.

3.1 Threat Model

Like all other covert channels [6, 12, 39, 42, 44, 45, 60, 69], this RAPL-based covert channel requires a pair of colluding sender and receiver running on the same computing platform but in different security domains. However, unlike most of them, the sender and receiver do not need to share the same memory pages [12], cache sets [39, 44, 45], memory buses [64, 69], or DRAM banks [50], and they do not need to run on the same core [6, 60] either. In fact, on a platform that has multiple sockets and is managed by a single operating system, the sender and receiver can even run on different CPU packages. Therefore, this new covert channel can circumvent the normally used partitioning-based defenses against covert communications.

We assume the receiver has access to the RAPL readings in the DRAM domain, but we do not require such access for the sender. If they are two unprivileged native processes on a Linux server or in two containers with their *powercap* interface exposed by the underlying daemon [9], both of them can actually access RAPL. Yet, they may be in an environment having the *powercap* interface proactively disabled. Nevertheless, we find that the receiver may still be able to read RAPL by other means. For example, at the time

[1] The `diagCall64()` has no wrapper function in the C library, so it cannot be invoked directly. Moreover, the generic system call function `syscall()` raises SIGSYS (namely bad system call) in terms of the system call number 0x4000001. However, directly using the `syscall` instruction can successfully invoke the `diagCall64()`.

[2] https://github.com/mozilla/gecko-dev/blob/master/tools/power/rapl.cpp

[3] This way of access has already been eliminated by a few of the cloud service providers investigated in [9]. Nevertheless, it has shown that default settings can render this situation.

of this study, we discovered that an attacker-controlled receiver VM on Amazon EC2 can use the Linux *msr* driver within the VM to read RAPL energy status MSRs.

3.2 Covert Channel over RAPL

To understand how to encode information in the new covert channel, let us take a look at the effects of different memory access patterns on the RAPL readings in the DRAM domain. In this experiment, five memory access scenarios are considered: (I) We use the mov instruction to access an array of size 256 KB sequentially; (II) We use the mov instruction to access an array of size 512 MB sequentially; (III) We first use the mov instruction to write a single integer and then use the clflush instruction to remove its copy from the cache; (IV) We use the vmovntdq instruction to write an array of 16 256-bit integers sequentially; (V) We use the vmovntdq instruction to write an array of 1024 256-bit integers sequentially. In each scenario, the memory accesses are performed repeatedly to facilitate our exploration. Fig. 1 lists the code snippets of these scenarios.

```
// sizeof(long) is 8
// N is 32768 for (1)
// N is 67108864 for (2)
long a[N];
int i = 0;
while (1) {
  ++a[i];
  i = (i + 1) % N;
}
```

(I) & (II)

```
// sizeof(int) is 4
int t;
while (1) {
  asm volatile(
    "mov %%eax, (%0) \n"
    "clflush (%0) \n"
    :: "r" (&t)
  );
}
```

(III)

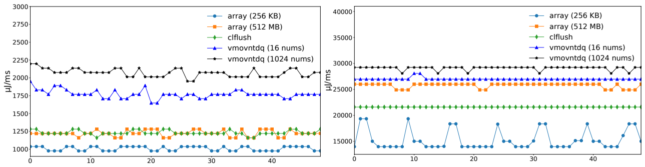
```
// sizeof(__m256i) is 32
// N is 16 for (4)
// N is 1024 for (5)
__m256i a[N], t;
int i = 0;
while (1) {
  _mm256_stream_si256(a + i, t);
  i = (i + 1) % N;
}
```

(IV) & (V)

Figure 1: Five memory access scenarios. In (I) and (II), mov is generated by the compiler to access the array. In (IV) and (V), the compiler intrinsic _mm256_stream_si256 corresponds to vmovntdq.

We use the *powercap* interface to periodically read the the *energy_uj* file in the DRAM domain to acquire a sequence of energy consumption values under each scenario. The *energy_uj* file contains only one number that gives the consumed energy and is updated nearly every 1 ms by the *powercap*. We read the value in the file every 500 μs. We take the difference between consecutive samples to obtain a sense of power consumption. Since we read the file about twice faster than it being updated, two consecutive samples may have the same value, i.e., their difference is 0, in which case we simply use the last non-zero difference as the current power consumption. The experimental results on two platforms in terms of the five memory access scenarios are shown in Fig. 2. One platform is equipped with a Core i7-7700 (Kaby Lake) processor, and the other platform is equipped with a Xeon D-1548 (Broadwell) processor.

Since the size of the array in scenario (I) is only 256 KB, far smaller than the size of the last-level cache (LLC), most of the memory accesses in that scenario cannot reach the DRAM. By contrast, the array in scenario (II) is too large for the LLC, and therefore eviction occurs constantly such that the DRAM is accessed regularly. In scenario (III), the same memory block is repeatedly written and then flushed out of the cache, so there are two DRAM accesses (i.e., one is due to write and the other one is due to write-back) in each of its iterations. In scenarios (IV) and (V), the vmovntdq instruction is used to bypass the cache to reach the DRAM. (vmovntdq is an AVX instruction, and it stores packed integers using non-temporal hint [24, 25].) In other words, the DRAM is (almost) always accessed except for being in the first scenario. Accordingly, much less power in the DRAM domain is consumed in the first scenario compared to others, as shown in Fig. 2.



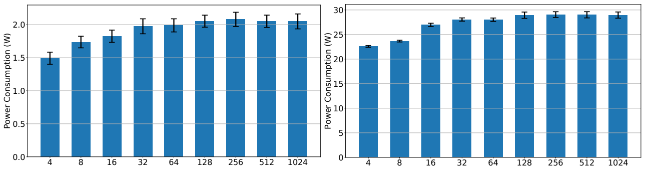
(a) Core i7-7700

(b) Xeon D-1548

Figure 2: Power consumption traces derived from RAPL readings in the DRAM domain under different scenarios.

Although the underlying DRAM is accessed constantly in each of the last four scenarios, different power consumption levels are clearly observed in Fig. 2. As illustrated in the figure, the power consumption is about 1.25 W, 1.25 W, 1.77 W, and 2.08 W respectively in scenarios (II)–(V) on Core i7-7700, and is about 25.99 W, 21.50 W, 26.97 W, and 29.23 W respectively on Xeon D-1548. (The points in Fig. 2 approximate how much energy in μJ per millisecond is used.) This shows that using the vmovntdq instruction to access memory can consume notably high power in the DRAM domain. This observation plays the fundamental role in constructing the new covert channel.

An interesting phenomenon is that the power consumption in the last two scenarios (IV) and (V) is different, although in both cases the vmovntdq instruction has been used to access the DRAM in the same pattern. The only difference is that 16 256-bit packed integers are circularly written in scenario (IV) while the number of 256-bit integers in scenario (V) is 1024. Fig. 3 further shows the variation of power consumption on both Core i7-7700 and Xeon D-1548 when varying the number of elements in the circularly written array. We can observe that in general the power consumption is higher if the array is larger but eventually plateaus.



(a) Core i7-7700

(b) Xeon D-1548

Figure 3: Power consumption (mean and standard deviation) in circularly writing an array of 2^n ($2 \leq n \leq 10$) 256-bit integers using vmovntdq.

Our conjecture is that this phenomenon is due to write combining and write collapsing performed during the corresponding memory accesses. According to [24], the non-temporal hint (as in vmovntdq) is implemented by using a write combining memory type protocol that enforces a weakly-ordered memory consistency model and has the corresponding memory accesses bypass the cache hierarchy. Although the cache hierarchy is not involved in non-temporal memory accesses, CPU uses another component named line fill buffers (LFB) to cache non-temporal stores [23]. Given these weakly-ordered stores, two performance optimization techniques are often applied to them in the LFB. One technique is called write combining which is to assemble multiple stores writing the same 64-byte aligned memory block into one to improve DRAM access efficiency (e.g., two stores writing two contiguous 256-bit integers in a 64-byte aligned block can be merged into one LFB entry). The other technique is called write collapsing that identifies stores writing the same location and may only commit the last store while ignoring the rest. As the LFB is relatively small (e.g., 10 entries on Haswell CPUs), if a large number of stores writing distinct locations are in between two store writing the same location, these two stores may not be collapsed. Therefore, when circularly writing a small array, the effect of write combining and collapsing can result in less amount of DRAM traffic than that when the array is sufficiently large. Less DRAM traffic naturally signifies less power consumption in the DRAM domain.

With respect to the effect of write combining, writing $2n$ contiguous 256-bit integers in the non-temporal manner will ideally occupy n LFB entries. According to [56], there are 10 entries in the LFB on pre-Skylake CPUs and 12 entries on post-Skylake CPUs. In the aforementioned experiment, circularly writing 1024 256-bit integers completely exhausts the LFB. We have observed that when this happens, the power consumption in the DRAM domain is also much higher than that incurred due to continuous cache eviction or flushing on any other tested platforms. Moreover, we have noticed that if circularly writing $2n$ 256-bit integers where the ratio of n to the number of LFB entries is $0.5 \sim 0.7$, a large amount of DRAM traffic is still created, and its power consumption is at least that incurred by continuous cache eviction or flushing but much less compared to that when the LFB is completely exhausted given the effect of write collapsing.

```
// $N$ is the number of bits to send
// $M$ controls how many writes are performed for sending a bit
// $T$ is a predefined delay acting as a separator
1  $a$ is an array of 1024 __m256i's;
2  $d$ is an array of $N$ bits to send;
3  for $i \leftarrow 0$ to $N-1$ do
4      if $d[i] = 0$ then
5          for $j \leftarrow 0$ to $M-1$ do
6              use vmovntdq to write $a[j \bmod 14]$;
7      else
8          for $j \leftarrow 0$ to $M-1$ do
9              use vmovntdq to write $a[j \bmod 1024]$;
10     wait for T ms;
```

Figure 4: Sending procedure.

Based on the observation and discussion stated above, we construct a covert channel that abuses RAPL readings in the DRAM domain for communication. As illustrated in Fig. 4, the sender simply leverages two characteristic power consumption levels to encode information. To transmit bit '0', the sender uses the vmovntdq instruction to circularly write 14 256-bit integers for M iterations (lines 4–6), while to transmit bit '1', the sender uses the vmovntdq instruction to circularly write 1024 256-bit integers for M iterations (lines 7–9). Notice that 14 contiguous 256-bit integers will ideally take 7 LFB entries, which equals 0.7 and 0.58 of the number of the LFB entries on pre- and post-Skylake CPUs respectively. To determine M, the sender at first measures loop time and adjusts M to make the loop run for a predefined period of time (e.g., 30 ms). In addition, to deal with the absence of synchronization, T ms delay (e.g., $T = 5$) is deliberately inserted (line 10), and this delay creates a considerable power consumption drop after a bit is sent. This drop serves as an effective bit separator on the receiver side.

```
// $S$ is the number of measured samples
// $P_0$ and $P_1$ are power consumption bounds delimiting '0' and '1'
// $L_l$ and $L_u$ are valid segment length lower and upper bounds
1  $e$ is an array of $S$ uint64_t's;
2  $p$ is an array of $S$ uint64_t's;
3  for $i \leftarrow 0$ to $S-1$ do
4      read RAPL energy status in the DRAM domain into $e[i]$;
5      wait for 500 $\mu$s;
6  for $i \leftarrow 1$ to $S-1$ do
7      $p[i] \leftarrow e[i] - e[i-1]$;
8      if $p[i] = 0$ then $p[i] \leftarrow p[i-1]$;
9  process $p$ using a lowpass filter;
10 segment $p$ into $\{p_0, p_1, \dots\}$ based on abrupt changes;
11 foreach $p_k \in \{p_0, p_1, \dots\}$ do
12     if $L_l \le |p_k| \le L_u$ then
13         if $P_0 \le \mathrm{mean}(p_k) < P_1$ then '0' is received;
14         else if $P_1 \le \mathrm{mean}(p_k)$ then '1' is received;
```

Figure 5: Receiving procedure.

The receiver recovers transmitted information following the procedure outlined in Fig. 5. It first records the RAPL energy status in the DRAM power domain every 500 μs for S times (lines 3–5). As mentioned above, this sampling can be achieved by several means, e.g., the *powercap* interface if the receiver is an unprivileged native process or the Linux *msr* driver if it is an attacker-controlled receiver VM on Amazon EC2. After collecting enough samples, the difference between consecutive samples is taken to obtain a sense of power consumption. Since the sampling rate is twice faster than the status updating rate, the difference may be 0, in which case we just use the previous difference (lines 6–8).

The receiver then processes the obtained power consumption trace using a lowpass filter (line 9). We just use a simple moving average filter. After that, the receiver divides the power consumption trace into segments based on abrupt changes (line 10). Several offline change point detection approaches may be used [63], but we simply locate the considerable drops due to the inserted delay. For a segment, the receiver checks if it has at least L_l points but at most L_u points (line 12), where L_l and L_u depend on the predetermined period of time for sending one bit. If the length is valid, the receiver tries to recognize the bit by checking the mean of the segment against two thresholds – P_0 and P_1 are the power consumption thresholds representing the cases in which 14 and 1024 256-bit integers are circularly written respectively (lines 13–14).

Note that several improvements may be made to this covert channel protocol. For example, we can leverage more advanced segmentation and recognition algorithms in the receiver. Since optimization is not the primary focus of this paper, we will not further discuss those possibilities but leave them to the future.

3.3 Evaluation

We evaluate the performance of this covert channel on three platforms that are summarized in Table 2. For the evaluations, we use Docker to establish effective isolation, i.e., the sender and the receiver run in separate Docker containers. The Docker environment has been updated to the latest (version 19.03), and all the containers are official images pulled from the Docker Hub. The container images for the sender and receiver may be different. We use this setup to simulate the PaaS cloud situation studied in [9].

Table 2: Platforms used for covert channel evaluation.

Platform	CPU (Installed #)	Memory	Host OS	Docker Images†
A	Core i7-7700 (1)	4×8 GB DDR4-2400	Ubuntu 18.04	debian/ubuntu
B	Xeon D-1548 (1)	4×16 GB DDR4-2400	Ubuntu 16.04	centos/amazonlinux
C	Xeon Gold 6130 (2)	12×16 GB DDR4-2666	CentOS 7	ubuntu/fedora

†The images are for sender/receiver respectively, and are pulled directly from the Docker Hub.

We utilize a pseudo-random number generator to create a piece of data having 1,000 bits for transmitting. The bandwidth of the covert channel is mainly determined by how long each bit-sending loop in the sender takes and how much delay needs to be inserted between the bit-sending loops. Recall that the delay is added for the synchronization purpose by generating a considerable power consumption drop after each bit is sent. We find that the delay should be more than 3 ms for this purpose, and we choose to use 5 ms as the delay in the following.

We change the bit-sending loop time and obtain the bit rate and error rate in the corresponding case. Given the bit-sending loop time X, the bit rate b is simply derived by $b = \frac{1000}{X+5}$ bps. Starting from X being 30 ms, we reduce X by 5 ms each time until X becomes 5 ms. We evaluate the error rate with respect to the Levenshtein edit distance between the sent data and the received bits. The Levenshtein edit distance is the minimum number of edits (insertions, deletions, or substitution) required to change one string into another string. The evaluation results are shown in Fig. 6.

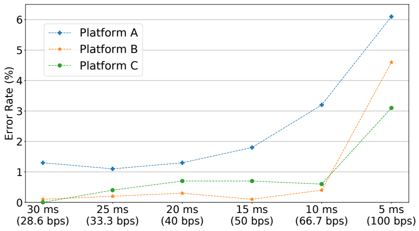


Figure 6: Error rate *w.r.t.* bit-sending loop time (bit rate).

From Fig. 6, we observe that the error rate increases as the bit rate increases. When the bit-sending loop time is not less than 10 ms, the error rate is less than 0.4% on platform B and less than 0.7% on platform C. When the loop time is 5 ms, the error rate reaches 4.6% and 3.1% on platforms B and C. On the other hand, the error rate on platform A is always higher, but still reasonable (less than 2%) if the loop time is not less than 15 ms. The error rate on platform A reaches 6.1% at the fastest 100 bps. We have manually checked the RAPL readings on platform A, and found that when bit '1' is transmitted, the power consumption occasionally climbs slowly up to the level representing '1'. When it occurs, the power consumption trace has a recognizable pattern. Thus, if we take into account this slow rise pattern, we can reduce the error rate on platform A to nearly 0 when the loop time is not less than 20 ms.

Notice that platform C is a non-uniform memory access (NUMA) system. There are two CPU sockets on this platform, on each of which there is a CPU installed, and to each CPU, there is a set of DRAM modules attached. The operating system unitedly manages these CPUs and their associated memory. The mode of this NUMA is non-interleaved, which means that the memory allocations of a process will be attempted on its local memory first. The power domains of both CPUs can be seen under the *powercap*. We run the sender on one CPU and the receiver on the other, namely, a cross-CPU covert channel is actually evaluated on platform C. In contrast to another cross-CPU covert channel DRAMA [50], our RAPL-based one has a critical advantage that is no need for having DRAM shared between the sender and receiver. In the cases of non-interleaved NUMA systems like platform C, it is in effect hard to establish DRAM sharing between the sender and receiver that run on different CPUs, as the local memory needs to be exhausted first for allocations on other NUMA nodes.

Table 3: Error rate change under disturbance.

Platform A	Platform B	Platform C
1.3% ⟶ 0.4%	0.1% ⟶ 0.7%	0.0% ⟶ 0.8%

In addition, we run another container along with the sender, in which a program continuously accesses an array of size 512 MB. This will create a large amount of DRAM traffic to disturb our covert channel. In such a situation, we evaluate the error rate of the covert channel at 28.6 bps (i.e., the bit-sending loop time is 30 ms). The results are shown in Tab. 3. Surprisingly, the error rate on platform A becomes 0.4%, and we find that the slow rise pattern when sending '1' has become rarer, which explains the improvement. The very small error rate changes on platforms B and C further show the robustness of this covert channel. Because the entire LLC will suffer constant eviction in this case, we believe the cache-based covert channels may be degraded significantly. Moreover, in this situation, the method proposed in [48] may hardly work (especially if we only have one logical processor on the sender side), since it uses eviction to increase the DRAM power consumption for transmitting '1' but eviction now can regularly occur.

4 INFERRING WEBSITE VISITS

The second study we have conducted is to leverage RAPL to infer which websites have been visited. In general, website visit history is a piece of extremely private information, and it may directly or indirectly disclose the political views, financial status, medical condition, and other sensitive secrecy of a user. Although many mechanisms/tools have been developed to protect such privacy, it has been shown that an attacker can still use various website fingerprinting techniques to track the browsed websites [2, 3, 14, 20, 27–29, 35, 37, 38, 49, 58, 65]. In this section, we demonstrate that RAPL can be exploited for effective website fingerprinting.

4.1 Threat Model

An attacker intends to track the website visits of a victim for some malicious purpose. We assume the victim will visit popular websites, and thus the attacker can profile many potential ones according to a particular list (e.g., Alexa top sites) to build a model. Note that the term "website" is abused in the literature on website fingerprinting, where, instead of the whole site, it is used to refer to specific webpages. We follow this misnomer tradition in this section, and use website and webpage interchangeably when the context is clear. We focus on the homepages of websites, which users usually visit at first.

We assume that the attacker has access to the RAPL readings, which can be achieved by a piece of malware via the methods mentioned in Section 2 (e.g., the diagnostic system call under Mac OS X). How to place the malware on the computer of the victim is out of scope, but, as presumed in the previous work, the attacker may carry it out via malicious apps [14, 27, 35] and other possible approaches include social engineering, USB interface, and physical access. Other than the availability of access to RAPL, we do not assume any other vulnerabilities.

A victim typically uses a personal computer (such as a desktop/workstation/laptop) for web browsing. The victim may use one of the most popular browsers such as Chrome or Safari to surf the Internet. If privacy is of the utmost consideration, the victim may choose to leverage the Tor browser for enhancing the protection. Because access to RAPL without privilege is assumed, the operating system in use may be either Linux or Mac OS, and the CPU should be an Intel processor that is not out-of-date (e.g., we consider pre-Sandy Bridge processors as obsolete). We do not assume any other platform configurations.

4.2 Website Fingerprinting through RAPL

When opening a website in a browser, the HTML file will be retrieved from the server, and the browser parses the HTML file for building a document object model (DOM) tree in the memory. The cascading style sheets (CSS) and JavaScript (JS) scripts will also be retrieved, and they typically augment and modify the DOM tree for aesthetic and interactive purposes. Additionally, other web elements like images will be fetched and handled (e.g., image decoding). As the contents necessary for displaying the page become available, the webpage layout will be solved and drawn on the screen. Therefore, rendering a webpage involves the use of various computational resources, e.g., CPU, GPU, and memory.

Since different webpages have different designs, rendering them can lead to different patterns on using the computational resources, which implies different power consumption patterns of the corresponding computational resources. As rendering a webpage usually takes hundreds to thousands of milliseconds, we expect that the power consumption patterns can be reflected in the RAPL readings of different power domains that are fine-grained and updated about every 1 ms (see Section 2). In other words, we anticipate to use RAPL readings for fingerprinting websites.

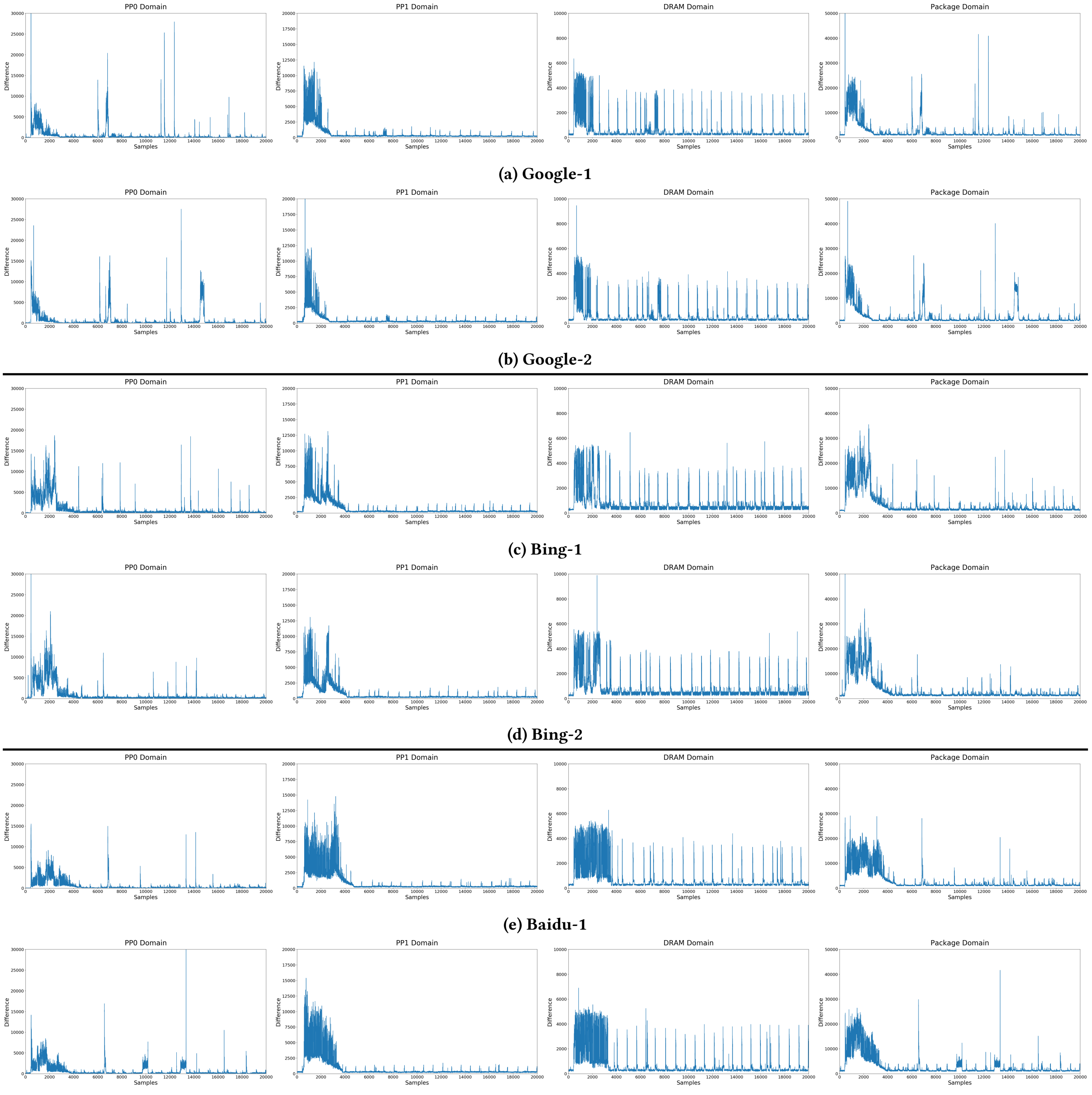
To validate our anticipation, we periodically sample each RAPL power domain when opening three different websites and derive their power traces for comparison. (Recall that the RAPL readings represent the consumed energy, so we take the difference of consecutive samples to obtain the sense of power consumption.) For a better comparison, we choose the websites to be Google, Bing, and Baidu, that are the three most popular search engines in the world according to Alexa top sites. Their webpages are among very simply designed modern ones, and thus help us test the granularity of RAPL-based fingerprinting. Moreover, their webpages have similar looks, and hence allow us to check how good the distinguishability is.

Fig. 7 illustrates the derived power traces of opening these three websites in the Chrome browser (version 83.0.4103.116) on a laptop equipped with a Core i7-8550U processor. The laptop uses the Ubuntu Linux 20.04 operating system. We read RAPL through the *powercap* interface in each power domain every 500 μs for 10 seconds when opening a website, and the traces of power consumption are derived following the lines 6–8 in Fig. 5. For each of the three websites, two measurements are taken – the first one is captured when the laptop is next to the access point and the Wi-Fi signal strength is very good, whereas the second one is measured when the laptop is placed several meters away from the access point and the Wi-Fi signal is not very strong.

Although similarly designed, the webpages of these three websites have utterly different HTML documents, CSS style sheets, JS scripts, images, and other plug-in objects. Therefore, when they are rendered, different CPU loads are generated and the power traces in the PP0 domain should be different. From Fig. 7, we can observe that the traces in the PP0 domain indeed have noticeable distinctions for different websites, although the traces when opening the same website are only similar but not identical. For instance, in terms of Google, the page rendering is apparent in the initial 3000 samples, but rendering the other two spans more than 4000 samples. Moreover, a large portion of the first 4000 samples when opening Bing has values higher than 10000, because Bing has a more complicated background to process and thus more CPU power will be consumed during rendering.

Modern popular browsers such as Chrome and Safari use GPUs not only for displaying but also for helping webpage rendering. Typically, there is an integrated GPU in desktop/mobile CPUs. If there is no discrete GPU, the integrated one will take exclusive charge of the activities using GPU. In this case, the appearance of a webpage affects the power consumption of the integrated GPU, which can be derived from the RAPL readings in the PP1 domain. The laptop we use does not have a discrete GPU, so we expect to distinguish websites by comparing the derived power traces in its PP1 domain. Fig. 7 illustrates that there exist certain patterns in the traces which can consistently differentiate these three websites. For example, similar to that in the case of PP0 power traces, it seems that Google always induces the shortest power surges among these three websites when opened. Notice that if a discrete GPU is used, the derived power trace in the PP1 domain will constantly be 0, namely it cannot be leveraged for website fingerprinting.

Rendering a modern webpage usually takes several (or even hundreds of) megabytes of memory for objects like the DOM tree, JS runtime variables, and images, on which the rendering operations



(a) Google-1

(b) Google-2

(c) Bing-1

(d) Bing-2

(e) Baidu-1

(f) Baidu-2

Figure 7: The RAPL power traces for the three most popular search engine websites.

are performed and create a large amount of memory traffic. The rendering-dependent memory traffic thus indicates the possibility of using the power traces in the DRAM domain for website fingerprinting. In terms of the three search engine websites, we can confirm that there are discernible patterns in their DRAM domain power traces shown in Fig. 7. Note that there is an interesting phenomenon that spikes emerge in the traces in all of the RAPL domains nearly every 1000 samples (i.e., 500 ms) after the initial rendering, but the spikes are the most noticeable in the DRAM domain traces. We find that this phenomenon is caused by the blinking text cursor in the input box on the search page. Each time when the cursor blinks, the displayed raw bitmap will be regenerated and redrawn on the screen. Therefore, the power consumption in each domain will rise correspondingly. This demonstrates that our RAPL-based fingerprinting has a resolution sufficiently high to distinguish common activities on a webpage.

As the PP0 and PP1 domains are both sub-domains of the package domain, the superposition of the power traces in the PP0 and PP1 domains approximates the trace in the package domain, which means that the package domain trace embraces the discernible patterns in the power consumption of CPU core and GPU when rendering a webpage[4]. (We find that the sum of the readings from PP0 and PP1 is always the dominant portion of the RAPL value in the package domain.) Hence, the power trace in the package domain can also fingerprint the websites, which can be verified via comparing the traces of these three search engine websites in Fig. 7.

Given the above discussion, we can see that the derived power trace in any RAPL domain can be used for our fingerprinting purpose. However, there are still several questions that need to be explored. Can we leverage the power traces to achieve acceptable fingerprinting performance? Which domain's traces can be better used for fingerprinting? If the technique performs decently in the regular network scenario, can we also achieve effective fingerprinting in the anonymity network scenario? In the following, we use evaluations to answer these questions.

4.3 Evaluation Setup

In the following evaluations, we use two computer systems, which are listed in Table 4. (Note that these two computer systems are not geographically close, as they are located in different cities.) On the XPS laptop, we use the *powercap* interface to access RAPL, and on the Mac mini desktop, we use the `diagCall64()` system call to access RAPL. Under the regular network scenario, we use the Chrome browser on the XPS (version 83.0.4103.116), and we use both Chrome and Safari on the Mac desktop (version 84.0.4147.89 and 13.0.2 respectively). In the anonymity network scenario, we use the latest Tor on both computer systems (version 9.5.3).

Table 4: Computer systems used for evaluations on RAPL-based website fingerprinting.

Computer	CPU	Memory	OS
Dell XPS 13	Core i7-8550U	2×8 GB LPDDR3-2133	Ubuntu 20.04
Apple Mac mini	Core i5-8500B	8 GB DDR4-2666	macOS Catalina 10.15.3

Given the Alexa top 50 websites, we select 37 of them by throwing away the websites that are adult, subsites (e.g., login pages), and country-dedicated (e.g., Google and Google UK). The websites are list in Table 10 in Appendix. We capture 100 traces in each RAPL domain for each of these 37 websites on the two computers. Each trace lasts for 15 s, and we read RAPL every 500 μs, so there are 30,000 samples in each trace.

Note that the evaluations in this section consider only the closed-world setting against the selected 37 websites. To perform website fingerprinting in open-world settings, a proper one-class classification method is needed for accurate and robust novelty detection. We leave this investigation to our future work.

4.4 Comparison between RAPL Domains

Because the RAPL readings in all of the four domains can be used for website fingerprinting (if the integrated GPU is not used, the readings in the PP1 domain cannot be used for this), we want to find out whether they have equivalent distinguishability or not. To this end, we use the k-Nearest Neighbors (kNN) classification method, which is based on the concept of feature similarity. Since we focus on checking how similar the traces in a domain are for the same website and how distant they are for different websites, we measure how accurate the kNN models can achieve only under the regular network circumstances.

We choose k to be 5 and use the most common Euclidean distance to measure the similarity. Each model is built from the *aligned* power traces corresponding to rendering the webpages of the 37 websites in each RAPL power domain. We carry out a 5-fold cross validation to estimate the accuracy of the models. The results in terms of the mean accuracy and 95% confidence interval are reported in Table 5.

Table 5: kNN classification accuracy estimates using 5-fold cross validation – mean and 95% confidence interval.

Case	Package	PP0	PP1	DRAM
XPS-Chrome	0.84 (± 0.06)	0.28 (± 0.03)	0.85 (± 0.05)	0.64 (± 0.04)
Mac-Chrome	0.73 (± 0.05)	0.69 (± 0.03)	0.11 (± 0.02)	0.20 (± 0.03)
Mac-Safari	0.91 (± 0.06)	0.90 (± 0.06)	0.18 (± 0.04)	0.56 (± 0.06)

From the results, we can observe that the websites can be well distinguished according to their power traces derived in the package domain. However, the model based on the traces in the PP0 domain on the XPS has a relatively low classification accuracy, while the model based on its PP1 domain traces can classify the websites much more accurately. Interestingly, it is exactly the opposite in the cases of the Mac as shown by the results. This phenomenon needs a further investigation, and is left for our future research. Nevertheless, as the superposition of the power traces in the PP0 and PP1 domains approximates the trace in the package domain, we may directly choose the package domain for our purpose without considering the PP0 and PP1 domains separately, given the results showing positive impacts of the combination.

The power traces in the DRAM domain seem less effective for distinguishing different websites according to their resemblance measured in distance. However, as they are independent from the traces derived in the other three domains, they can be combined with the traces in other domains to improve the performance. For example, if we add the derived power trace in the DRAM domain to the corresponding trace in the package domain, the model built from the new traces can increase the accuracy by about 2%.

Note that given any trace, if we randomly guess which of the 37 websites it corresponds to, the accuracy should be 0.027 (i.e., $1/37$). While any kNN model based on the power traces in a RAPL domain has much better accuracy than that, we can see that the model based on the traces in the package domain outperforms others considering both stability and accuracy. It indicates that the power

[4] Although the DRAM domain is also treated as a sub-domain of the package domain under the *powercap* interface, we find that the energy consumption reported in the package domain does not contain the DRAM portion. Thus, we treat the DRAM domain as an independent one of the package domain.

traces in the package domain have a better distinguishability for fingerprinting websites compared with others.

4.5 Evaluation using DNN

Although using *k*NN directly on the aligned power traces is already much better than random guessing, the accuracy should be able to be further improved if any proper feature engineering is applied. However, feature crafting usually involves heavy manual labor. To avoid such labor, we choose to leverage deep neural network (DNN) to perform automatic feature learning from the raw data [52]. Given the comparison results presented above, we decide to use only the power traces derived in the package domain as raw data in this evaluation, since they show more effectiveness than others for website fingerprinting.

In terms of the DNN model, we choose to use Convolutional Neural Network (CNN) over Recurrent Neural Network (RNN), even though our power traces belong to time series data. One of the primary reasons is that RNN is designed mainly for time series prediction, and the other reason is that RNN usually suffers from the vanishing gradient problem more severely when training on long time series [7]. Specifically, we use the residual network (ResNet) architecture that is described in [67] for our purpose.

We notice that if the time series traces are too long, the classification performance degrades significantly. Due to this reason, we remove the repeated values in each captured RAPL trace before deriving its power trace. Recall that energy status in a RAPL domain is updated roughly every 1 ms, so about half of the samples will be removed if the sampling period is 500 μs. (As energy consumption reported by RAPL increases monotonically, this removal does not loss any information.) Moreover, we find that most of the essential information for fingerprinting in a power trace is actually contained in the part corresponding to the first 5 ∼ 6 s when opening a website. Thus, we decide to use only the first 7000 samples in each power trace, which roughly corresponds to the first 7 s. (After removing the repeated values, the time between samples is about 1 ms.)

Table 6: Fingerprinting accuracy using ResNet in the regular network scenario. Testing models against traces corresponding to the same and different platforms/browsers.

		Testing Traces		
		XPS-Chrome	Mac-Chrome	Mac-Safari
Model	XPS-Chrome	**0.99 (± 0.01)**	0.04 (± 0.02)	0.05 (± 0.02)
	Mac-Chrome	0.07 (± 0.01)	**0.96 (± 0.02)**	0.05 (± 0.01)
	Mac-Safari	0.04 (± 0.02)	0.06 (± 0.02)	**0.95 (± 0.01)**

The evaluation results after replacing *k*NN with ResNet models are given by the entries on the main diagonal of Table 6. (Some confusion matrix examples are also demonstrated in Appendix.) As we can observe from the results, if the training and testing traces correspond to the same platform and browser, the ResNet model can significantly improve the classification (i.e., fingerprinting) accuracy (e.g., in the case of using Chrome on Mac, the accuracy is only 73% under *k*NN but becomes 96% under ResNet). However, the entries not on the main diagonal in Table 6 demonstrate that this RAPL-based website fingerprinting technique perform almost similar to random guessing across platforms and/or browsers, namely it is browser- and platform-sensitive.

Table 7: The average and lowest precision and recall when fingerprinting 740 traces in the regular network scenario.

		Precision		Recall	
		Mean	Min.	Mean	Min.
Case	XPS-Chrome	98.8%	90.9%	98.8%	90.0%
	Mac-Chrome	97.3%	86.9%	97.1%	75.0%
	Mac-Safari	95.9%	80.0%	95.4%	75.0%

Moreover, Table 7 shows the average and lowest precision and recall achieved in each case. With respect to a website w, we have

$$\text{precision}_w = \frac{\text{TP}_w}{\text{TP}_w + \text{FP}_w} \quad \text{and} \quad \text{recall}_w = \frac{\text{TP}_w}{\text{TP}_w + \text{FN}_w} \ ,$$

where TP_w refers to true positives for w (i.e., a trace corresponding to w is classified as “belonging to w”), and FP_w and FN_w refer to false positives (i.e., a trace not corresponding to w is classified as “belonging to w”) and false negatives (i.e., a trace corresponding to w is classified as “not belonging to w”). Precision reflects how much we can trust that the identified visits to a website are indeed made by a user, and recall indicates how much portion of the visits to a website made by a user can be correctly identified using this fingerprinting technique. From the results, we can see that both precision and recall are sufficiently high even in the worst case.

4.6 Evaluation on Tor

In addition to the evaluations of the RAPL-based website fingerprinting technique in the normal network scenario against commonly used browsers like Chrome and Safari, we further evaluate the technique under the anonymity network scenario. Specifically, we focus on Tor (The Onion Router) that encrypts network traffic and relays it between users and website servers anonymously.

In the case of Tor, using deep neural network for classification is more essential, because Tor introduces more intricacies. First, due to the unpredictable latency incurred by Tor routers and its overlay network, the power traces may be distorted in time. Second, due to the encryption and decryption operations used in each packet, the power traces are more noisy. Third, due to the changeable exit node, websites adapting to geolocations may use different languages (e.g., Google) or provide different contents (e.g., Bing), which can affect the consistency of the power traces. Neural networks, the convolutional ones in particular, should be good at inferring dependencies in time series data even in the presence of such intricacies [55].

Table 8: Fingerprinting accuracy using *k*NN & ResNet in the Tor network scenario – mean and 95% confidence interval.

Case	*k*NN	ResNet
XPS-Tor	0.41 (± 0.12)	0.81 (± 0.02)
Mac-Tor	0.34 (± 0.07)	0.78 (± 0.03)

Guided by the results of previous evaluations, we just use the package domain power traces in this evaluation as well. For model comparison, we also include the evaluation results using *k*NN. The results are shown in Table 8. From the results, we can observe that simply using *k*NN cannot achieve a decent performance (although still much better than random guessing). By contrast, using DNN can achieve an acceptable fingerprinting performance against Tor – 81% and 78% on tested XPS laptop and Mac desktop respectively.

Table 9: The average and lowest precision and recall when fingerprinting 740 traces in the Tor network scenario.

		Precision		Recall	
		Mean	Min.	Mean	Min.
Case	XPS-Tor	82.0%	47.8%	81.1%	55.0%
	Mac-Tor	79.6%	45.5%	78.4%	35.0%

We also list the average and lowest precision and recall achieved in the Tor network scenario in Table 9. We can observe that both precision and recall are not as high as the results corresponding to the regular network scenario as shown in Table 7. Nevertheless, they are still acceptable.

5 COUNTERMEASURES

To thwart the presented attacks and any other potential ones exploiting RAPL-induced side channels, the most straightforward and effective approach is to keep the RAPL energy status from being accessed by an unprivileged user.

In terms of the *powercap* interface under Linux, we expect a patch similar to that for the *pagemap*. Specifically, only users with the `CAP_SYS_ADMIN` capability can read the *energy_uj* files under the *powercap* interface correctly. If the user does not have `CAP_SYS_ADMIN`, the readings will be zeroed out.

In terms of the `diagCall64()` system call of Mac OS, we may limit the access to its power statistics mode. There is a boot argument `diag` that controls the access to some modes of this system call[5], where the power statistics mode is not one of them. (Before kernel version 2422.1.72, the `diag` argument controls the entire system call as a whole instead of separate modes.) We argue that the access to the power statistics mode should also be controlled by the `diag` argument, which is off by default.

In the cases of PaaS/IaaS clouds, many of their providers have proactively removed the exposure of the *powercap* interface to the tenants. However, we have found that an attacker-controlled VM may still be able to retrieve RAPL energy status information by accessing related MSRs on some IaaS clouds. To restrict access to RAPL MSRs, the corresponding bits in the read bitmap for low MSRs should be set and the accesses will need to be handled properly at VM exits.

To specifically mitigate the possible RAPL-based website fingerprinting attacks, we may borrow the "fuzzy time" idea, that has been used for building trusted browsers against timing attacks [33], to form a "fuzzy energy" method. A browser can randomly perform some energy-demanding computation in the background when rendering a webpage, which will introduce overwhelming energy consumption noise into the RAPL power domains. It will likely become much harder for an attacker to identify websites through the polluted RAPL fingerprints.

6 RELATED WORK

In this section, we briefly discuss some work closely related to this paper. The focuses are mainly on similar covert channel and website fingerprinting attacks.

6.1 Covert Channel Attacks

The confinement problem was formulated by Lampson in 1973 [34], which made the first mention of possible data exfiltration via covert channels. Since then, extensive research has been conducted on this topic. Roughly speaking, covert channels can be classified into logical and physical ones. Logical covert channels usually manipulate the states in some shared hardware component to encode and transfer information. While the majority of these logical covert channels are built on top of caches [12, 26, 39, 44, 45, 70], many other hardware components have also been exploited to build such channels, including execution ports [8, 66], branch predictor [6, 22], memory disambiguator [60], memory bus [69], DRAM bank row buffer [50], AES accelerator [22], and hardware random number generator [5]. Fundamentally, logical covert channels are based on resource contention, and can be mitigated using logical resource isolation [15, 66] or randomization [68].

On the other hand, physical covert channels are essentially constructed from certain physical side effects of computation. Although the presented work in this paper does not explicitly interact with the physical environment, it is the DRAM power consumption that becomes fundamentally exploited, and hence it is more likely to be categorized as a physical covert channel. Similarly, we categorize the covert channel proposed in [31] as a physical one, although it just measures the performance fluctuation due to power management. Likewise, the physical effects of voltage drop as well as thermal rise on the frequency of ring oscillators have been exploited to enable covert channel communication through shared FPGAs on clouds [11, 62], and we treat them as physical covert channels as well. Different from logical ones, physical covert channels are not based on shared resource contentions but on the changes made to the physical quantities, so the countermeasures against logical ones cannot defend against them.

Prior to our work on covert channel, it is mentioned in [9] that RAPL may be exploited to construct covert channels. Later in [48], cache eviction is leveraged to raise the DRAM power consumption for encoding information. However, the receiver may be considerably disturbed by another process accessing a large array in mistake for its collusive sender. By contrast, the covert channel built in this paper is based on our observation that non-temporal memory accesses can induce distinguishable DRAM power consumption from that caused by cache eviction or flushing. In other words, our covert channel is more robust.

There are some other physical covert channels that exfiltrate data between isolated parties on the same platform. For example, heat issued during computation has been used to form a cross-core thermal covert channel [1, 42]. Although many physical covert channels including ours target the same platform scenario, most of them are used to achieve unauthorized data transfer across air gaps, where various physical side effects are exploited, such as electromagnetic [16, 57, 72], acoustic [18], thermal [17], and optical [19].

[5] The boot argument `diag` is parsed during system initialization to set a kernel variable `dgWork.dgFlags`, whose value is checked against a constant `enaDiagSCS` (that is 0x00000008) at the beginning of the `diagCall64()` to set a flag for controlling the access to some modes. There are two ways to set Mac OS boot arguments, which are via the `nvram` command and through the boot property list file.

6.2 Website Fingerprinting Attacks

There are many website fingerprinting attacks running the gamut from methodology to technology. In general, an attacker may either try to statistically analyze the network traffic of a victim to infer the visited websites or exploit certain local side-channel information to acquire this sensitive information.

Website fingerprinting via network traffic analysis is based on the observation that some network-level characteristics such as packet timings and packet lengths may preserve and can be exploited even in the encrypted communication situations [21]. With properly designed classifiers, many proposed attacks can achieve very accurate fingerprinting [2, 20, 28, 29, 37, 40, 49].

Other than network traffic analysis, an attacker may exploit local side-channel information to track the browsed webpages. The presented technique in this paper belongs to this category. The types of the exploited side-channel information range from logic ones like memory footprints to physical ones like power consumption.

To take logical fingerprints, hardware modification is not required, and it only needs that a piece of attacker-controlled code (e.g., malware or JavaScript) runs on the target machine. Not only are logical fingerprints much easier to take than their physical counterparts, but they are also in many forms. In [27], it has been shown that the memory footprints of a browser can be used to reveal the rendered webpages. In [35] and [46], GPU memory dumps and utilization patterns are used to fingerprint websites. In [59], Android data usage statistics is leveraged to achieve website fingerprinting on mobile phones, while in [32], storage usage statistics is used to fingerprint websites browsed using the Chrome browser. In [65], it shows that the identities of visited websites can be inferred by exploiting shared event loops in Chrome. In [14], hardware performance counters are sampled through the Linux *perf* interface and their statistical information is used to fingerprint websites. (However, note that the `perf_event_paranoid` *sysctl* parameter controls the use of the *perf* interface by unprivileged users, whose default value has already disallowed unprivileged users from accessing the `perf_event_open()` system call.) In addition, in [47] and [58], website fingerprinting techniques using cache-based side-channel information have been investigated.

Compared to logical ones, physical fingerprints of websites are usually harder to take. This is traditionally more evident in terms of power-based website fingerprinting techniques, as either AC power outlet [3], USB charging station [71], or battery [38] needs to be instrumented and/or replaced. Yet, there has been some work attempting to remove the need for special hardware but estimate the power traces in software for website fingerprinting on mobile devices [51]. Similarly, our RAPL-based technique fundamentally leverages the power consumption side-channel information and is purely in software. In addition, some other physical side channels have been exploited to identify websites such as acoustic [10] and electromagnetic [43].

7 CONCLUSION

In this paper, we have conducted two studies on how to exploit RAPL-induced side channels to mount realistic attacks. In the first study, we have constructed an RAPL-based covert channel via manipulating the DRAM power consumption with a single AVX instruction, and in the second study, we have examined the potential of exploiting RAPL for website fingerprinting. These studies serve as new examples of the fact that useful system designs with security-obliviousness commonly exist. In addition, we have discussed certain possible countermeasures. In the future, we plan to investigate other possible security problems caused by RAPL. For instance, we will inspect if we can leverage RAPL to help find minimal eviction sets, as when eviction happens constantly, more power consumption should be observed in the DRAM domain than the normal situation, which can serve as an eviction indicator. Another example is to generalize the RAPL-based website fingerprinting technique to fingerprint videos and applications.

ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation (CNS-1739328, CNS-2008339, CNS-2038076, and CNS-2054657). Platforms B and C in our covert channel evaluation are from CloudLab. The authors would like to thank the anonymous reviewers and the shepherd, Daniel Gruss, for their comments and suggestions that help us improve the quality of the paper.

REFERENCES

[1] D. B. Bartolini, P. Miedl, and L. Thiele. On the capacity of thermal covert channels in multicores. In *EuroSys'16*.

[2] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *CCS'12*.

[3] S. S. Clark, H. Mustafa, B. Ransford, J. Sorber, K. Fu, and W. Xu. Current events: Identifying webpages by tapping the electrical outlet. In *ESORICS'13*.

[4] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. RAPL: Memory power estimation and capping. In *ISLPED '10*.

[5] D. Evtyushkin and D. Ponomarev. Covert channels through random number generator: Mechanisms, capacity estimation and mitigations. In *CCS'16*.

[6] D. Evtyushkin, D. Ponomarev, and N. Abu-Ghazaleh. Understanding and mitigating covert channels through branch predictors. *ACM Transactions on Architecture and Code Optimization*, 2016.

[7] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller. Deep learning for time series classification: a review. *Data Min. Knowl. Discov.*, 2019.

[8] A. Fogh. Covert Shotgun: Automatically Finding SMT Covert Channels. https://cyber.wtf/2016/09/27/covert-shotgun/.

[9] X. Gao, Z. Gu, M. Kayaalp, D. Pendarakis, and H. Wang. Containerleaks: Emerging security threats of information leakages in container clouds. In *DSN'17*.

[10] D. Genkin, M. Pattani, R. Schuster, and E. Tromer. Synesthesia: Detecting screen content via remote acoustic side channels. In *S&P'19*.

[11] I. Giechaskiel, K. Rasmussen, and J. Szefer. C^{3}APSULe: Cross-FPGA covert-channel attacks through power supply unit leakage. In *S&P'20*.

[12] D. Gruss, C. Maurice, K. Wagner, and S. Mangard. Flush+Flush: A fast and stealthy cache attack. In *DIMVA'16*.

[13] A. Guliani and M. M. Swift. Per-application power delivery. In *EuroSys'19*.

[14] B. Gulmezoglu, A. Zankl, T. Eisenbarth, and B. Sunar. PerfWeb: How to violate web privacy with hardware performance events. In *ESORICS'17*.

[15] A. Gundu, G. Sreekumar, A. Shafiee, S. Pugsley, H. Jain, R. Balasubramonian, and M. Tiwari. Memory bandwidth reservation in the cloud to avoid information leakage in the memory controller. In *HASP'14*.

[16] M. Guri, A. Kachlon, O. Hasson, G. Kedma, Y. Mirsky, and Y. Elovici. GSMem: Data exfiltration from air-gapped computers over GSM frequencies. In *USENIX Security 15*.

[17] M. Guri, M. Monitz, Y. Mirski, and Y. Elovici. Bitwhisper: Covert signaling channel between air-gapped computers using thermal manipulations. In *CSF'15*.

[18] M. Guri, Y. Solewicz, A. Daidakulov, and Y. Elovici. Acoustic data exfiltration from speakerless air-gapped computers via covert hard-drive noise ('diskfiltration'). In *ESORICS'17*.

[19] M. Guri, B. Zadov, and Y. Elovici. LED-it-GO: Leaking (a lot of) data from air-gapped computers via the (small) hard drive LED. In *DIMVA'17*.

[20] J. Hayes and G. Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *USENIX Security 16*.

[21] A. Hintz. Fingerprinting websites using traffic analysis. In *PETS'02*.

[22] C. Hunger, M. Kazdagli, A. Rawat, A. Dimakis, S. Vishwanath, and M. Tiwari. Understanding contention-based channels and using them for defense. In *HPCA'15*.

[23] Intel. Intel® 64 and IA-32 Architectures Optimization Reference Manual.

[24] Intel. Intel® 64 and IA-32 Architectures Software Developer Manuals.

[25] Intel. Intel® Architecture Instruction Set Extensions Programming Reference.

[26] G. Irazoqui, T. Eisenbarth, and B. Sunar. Cross processor cache attacks. In *AsiaCCS'16*.

[27] S. Jana and V. Shmatikov. Memento: Learning secrets from process footprints. In *S&P'12*.

[28] R. Jansen, M. Juarez, R. Galvez, T. Elahi, and C. Diaz. Inside job: Applying traffic analysis to measure Tor from within. In *NDSS'18*.

[29] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt. A critical evaluation of website fingerprinting attacks. In *CCS'14*.

[30] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou. RAPL in action: Experiences in using RAPL for power measurements. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 2018.

[31] S. K. Khatamifard, L. Wang, A. Das, S. Kose, and U. R. Karpuzcu. POWERT channels: A novel class of covert communication exploiting power management vulnerabilities. In *HPCA'19*.

[32] H. Kim, S. Lee, and J. Kim. Inferring browser activity and status through remote monitoring of storage usage. In *ACSAC'16*.

[33] D. Kohlbrenner and H. Shacham. Trusted browsers for uncertain times. In *USENIX Security 16*.

[34] B. W. Lampson. A note on the confinement problem. *Commun. ACM*, 1973.

[35] S. Lee, Y. Kim, J. Kim, and J. Kim. Stealing webpages rendered on your browser by exploiting GPU vulnerabilities. In *S&P'14*.

[36] J. Levin. *Mac OS X and iOS Internals: To the Apple's Core.* Wrox Press Ltd., 2012.

[37] S. Li, H. Guo, and N. Hopper. Measuring information leakage in website fingerprinting attacks and defenses. In *CCS'18*.

[38] P. Lifshits, R. Forte, Y. Hoshen, M. Halpern, M. Philipose, M. Tiwari, and M. Silberstein. Power to peep-all: Inference attacks by malicious batteries on mobile devices. In *PETS'18*.

[39] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. Last-level cache side-channel attacks are practical. In *S&P'15*.

[40] L. Lu, E.-C. Chang, and M. C. Chan. Website fingerprinting and identification using ordered feature sequences. In *ESORICS'10*.

[41] H. Mantel, J. Schickel, A. Weber, and F. Weber. How secure is green IT? the case of software-based energy side channels. In *ESORICS'18*.

[42] R. J. Masti, D. Rai, A. Ranganathan, C. Müller, L. Thiele, and S. Capkun. Thermal covert channels on multi-core platforms. In *USENIX Security 15*.

[43] N. Matyunin, Y. Wang, T. Arul, K. Kullmann, J. Szefer, and S. Katzenbeisser. MagneticSpy: Exploiting magnetometer in mobile devices for website and application fingerprinting. In *WPES'19*.

[44] C. Maurice, C. Neumann, O. Heen, and A. Francillon. C5: Cross-cores cache covert channel. In *DIMVA'15*.

[45] C. Maurice, M. Weber, M. Schwarz, L. Giner, D. Gruss, C. A. Boano, K. Römer, and S. Mangard. Hello from the other side: SSH over robust cache covert channels in the cloud. In *NDSS'17*.

[46] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh. Rendered insecure: GPU side channel attacks are practical. In *CCS'18*.

[47] Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis. The spy in the sandbox: Practical cache attacks in JavaScript and their implications. In *CCS'15*.

[48] T. B. Paiva, J. Navaridas, and R. Terada. Robust covert channels based on DRAM power consumption. In *ISC'19*.

[49] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle. Website fingerprinting at internet scale. In *NDSS'16*.

[50] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard. DRAMA: Exploiting DRAM addressing for cross-cpu attacks. In *USENIX Security 16*.

[51] Y. Qin and C. Yue. Website Fingerprinting by Power Estimation Based Side-Channel Attacks on Android 7. In *TrustCom'18*.

[52] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen. Automated website fingerprinting through deep learning. In *NDSS'18*.

[53] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan. Power-management architecture of the Intel microarchitecture code-named sandy bridge. *IEEE Micro*, 2012.

[54] O. Sarood, A. Langer, L. Kalé, B. Rountree, and B. De Supinski. Optimizing power allocation to cpu and memory subsystems in overprovisioned HPC systems. In *CLUSTER'13*.

[55] R. Schuster, V. Shmatikov, and E. Tromer. Beauty and the burst: Remote identification of encrypted video streams. In *USENIX Security 17*.

[56] M. Schwarz, M. Lipp, D. Moghimi, J. Van Bulck, J. Stecklina, T. Prescher, and D. Gruss. Zombieload: Cross-privilege-boundary data sampling. In *CCS'19*.

[57] N. Sehatbakhsh, B. B. Yilmaz, A. Zajic, and M. Prvulovic. A new side-channel vulnerability on modern computers by exploiting electromagnetic emanations from the power management unit. In *HPCA'20*.

[58] A. Shusterman, L. Kang, Y. Haskal, Y. Meltser, P. Mittal, Y. Oren, and Y. Yarom. Robust website fingerprinting through the cache occupancy channel. In *USENIX Security 19*.

[59] R. Spreitzer, S. Griesmayr, T. Korak, and S. Mangard. Exploiting data-usage statistics for website fingerprinting attacks on android. In *WiSec'16*.

[60] D. Sullivan, O. Arias, T. Meade, and Y. Jin. Microarchitectural minefields: 4k-aliasing covert channel and multi-tenant detection in IaaS clouds. In *NDSS'18*.

[61] The Linux Kernel Documentation. Power Capping Framework. https://www.kernel.org/doc/html/latest/power/powercap/powercap.html.

[62] S. Tian and J. Szefer. Temporal thermal covert channels in cloud FPGAs. In *FPGA'19*.

[63] C. Truong, L. Oudre, and N. Vayatis. Selective review of offline change point detection methods. *Signal Processing*, 2020.

[64] V. Varadarajan, Y. Zhang, T. Ristenpart, and M. Swift. A placement vulnerability study in multi-tenant public clouds. In *USENIX Security 15*.

[65] P. Vila and B. Kopf. Loophole: Timing attacks on shared event loops in Chrome. In *USENIX Security 17*.

[66] Z. Wang and R. B. Lee. Covert and side channels due to processor architecture. In *ACSAC'06*.

[67] Z. Wang, W. Yan, and T. Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *IJCNN'17*.

[68] M. Werner, T. Unterluggauer, L. Giner, M. Schwarz, D. Gruss, and S. Mangard. ScatterCache: Thwarting cache attacks via cache set randomization. In *USENIX Security 19*.

[69] Z. Wu, Z. Xu, and H. Wang. Whispers in the hyper-space: High-speed covert channel attacks in the cloud. In *USENIX Security 12*.

[70] Y. Xu, M. Bailey, F. Jahanian, K. Joshi, M. Hiltunen, and R. Schlichting. An exploration of L2 cache covert channels in virtualized environments. In *CCSW'11*.

[71] Q. Yang, P. Gasti, G. Zhou, A. Farajidavar, and K. S. Balagani. On inferring browsing activity on smartphones via USB power analysis side-channel. *IEEE Transactions on Information Forensics and Security*, 2016.

[72] Z. Zhan, Z. Zhang, and X. Koutsoukos. BitJabber: The world's fastest electromagnetic covert channel. In *HOST'20*.

[73] H. Zhang and H. Hoffmann. Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. In *ASPLOS'16*.

8 APPENDIX

The list of the 37 websites used in this paper is given in Table 10. The number in the parentheses after each website is used to identify this website in the two confusion matrix examples given in Table 11 and Table 12.

Table 10: The 37 websites used in this paper.

360.cn (10)	Aliexpress.com (8)	Alipay.com (31)	Amazon.com (21)
Baidu.com (14)	Bing.com (9)	Blogger.com (24)	China.com.cn (25)
Csdn.net (20)	Ebay.com (23)	Facebook.com (3)	Google.com (37)
Instagram.com (4)	Jd.com (27)	Live.com (34)	Microsoft.com (26)
Myshopify.com (6)	Naver.com (7)	Netflix.com (19)	Office.com (28)
Okezone.com (22)	Qq.com (5)	Reddit.com (18)	Sina.com.cn (33)
Sohu.com (17)	Taobao.com (11)	Tianya.cn (12)	Tmall.com (32)
Tribunnews.com (29)	Twitch.tv (35)	Vk.com (30)	Weibo.com (15)
Wikipedia.org (2)	Xinhuanet.com (1)	Yahoo.com (16)	Youtube.com (36)
Zoom.us (13)			

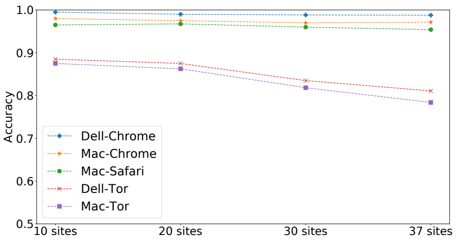


Figure 8: Fingerprinting accuracy in different closed-world scenarios where the number of monitored websites varies.

In addition, Fig. 8 shows the fingerprinting accuracy in different closed-world scenarios, where we order the websites in Table 10 alphabetically and select the first 10, 20, 30, and all of them to construct the corresponding groups. For each closed-world scenario and the combination of computer and browser, we train a model and perform fingerprinting on a set of test traces. From Fig. 8, we can see that the accuracy does not change much when the size of the closed-world group changes.

Table 11: Confusion matrix in terms of classifying 740 RAPL power traces (each of the 37 websites contributes 20 traces) in the Mac-Safari case.

		Predicted																																				
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
True	1	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	2	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	3	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	4	0	0	0	18	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
	5	0	0	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
	6	0	0	0	0	0	19	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	7	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	8	0	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	9	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	10	0	0	0	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
	11	0	0	0	0	0	0	0	0	0	0	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0
	12	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
	13	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	14	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	20	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	22	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	1	0	1	0	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	1	0	0	0	0	0	0	0	0	0	0
	25	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0
	26	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0	0	0
	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0
	28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0
	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0
	30	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0	0	0	0
	31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	19	0	0	0	0	0	0
	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0
	33	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	16	0	0	0	0
	34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0
	35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0
	36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0
	37	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19

Table 12: Confusion matrix in terms of classifying 740 RAPL power traces (each of the 37 websites contributes 20 traces) in the Dell-Tor case.

		Predicted																																				
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
True	1	17	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	2	0	17	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	3	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	4	0	0	0	17	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
	5	0	0	0	0	17	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	6	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	7	0	0	0	0	0	0	16	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	8	0	0	0	0	0	1	0	13	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	2	1	0	0	0	0	0	0
	9	0	0	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	10	0	0	0	0	2	0	0	0	0	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	11	0	0	0	0	1	0	0	0	0	1	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
	12	0	0	0	0	0	0	0	0	0	0	0	18	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	13	0	0	0	1	0	2	0	0	0	0	0	2	11	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
	14	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0
	16	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	17	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	1	0
	18	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	13	0	1	0	1	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0
	19	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	3	12	0	0	0	0	2	0	0	0	1	0	1	0	0	0	0	0	0	0
	20	1	0	0	0	0	0	0	0	0	0	0	0	3	0	1	0	0	1	0	13	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	21	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	22	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	15	1	0	0	0	0	0	0	0	2	0	0	0	1	0	0
	23	0	0	0	1	0	0	0	1	1	0	0	2	0	0	0	0	0	1	0	1	0	0	12	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	24	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	16	0	0	0	1	0	0	0	0	0	0	0	0	0
	25	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	16	0	0	1	0	0	0	0	0	0	0	0	0
	26	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	13	0	0	0	1	2	0	0	0	0	0	0
	27	0	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0
	28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0
	29	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	14	0	1	0	0	0	0	0	0
	30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	17	0	0	0	0	0	0	0
	31	1	0	0	1	0	0	1	0	0	0	0	0	2	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	2	11	0	0	0	0	0	0
	32	0	0	0	0	0	0	0	0	0	1	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	0	0	0	0	0
	33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	17	0	0	0	0
	34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	18	0	0	0
	35	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	17	0	0
	36	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	0
	37	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19