The Security Risks of Power Measurements in Multicores

Philipp Miedl
Computer Engineering and Networks Laboratory, ETH Zurich
Zürich, Switzerland
miedlp@ethz.ch

Lothar Thiele
Computer Engineering and Networks Laboratory, ETH Zurich
Zürich, Switzerland
thiele@ethz.ch

ABSTRACT

Two of the main goals of power management in modern multicore processors are reducing the average power dissipation and delivering the maximum performance up to the physical limits of the system, when demanded. To achieve these goals, hardware manufacturers and operating system providers include sophisticated power and performance management systems, which require detailed information about the current processor state. For example, Intel processors offer the possibility to measure the power dissipation of the processor. In this work, we are evaluating whether such power measurements can be used to establish a covert channel between two isolated applications on the same system; the power covert channel.

We present a detailed theoretical and experimental evaluation of the power covert channel on two platforms based on Intel processors. Our theoretical analysis is based on detailed modelling and allows us to derive a channel capacity bound for each platform. Moreover, we conduct an extensive experimental study under controlled, yet realistic, conditions. Our study shows, that the platform dependent channel capacities are in the order of 2000 bps and that it is possible to achieve throughputs of up to 1000 bps with a bit error probability of less than 15%, using a simple implementation. This illustrates the potential of leaking sensitive information and breaking a systems security framework using a covert channel based on power measurements.

CCS CONCEPTS

• **Security and privacy** → **Security in hardware**; **Operating systems security**;

KEYWORDS

covert channel; power; capacity bound; empirical study

ACM Reference Format:
Philipp Miedl and Lothar Thiele. 2018. The Security Risks of Power Measurements in Multicores. In *SAC 2018: SAC 2018: Symposium on Applied Computing , April 9–13, 2018, Pau, France.* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.1145/3167132.3167301

SAC 2018, April 9–13, 2018, Pau, France

ACM ISBN 978-1-4503-5191-1/18/04…$15.00
https://doi.org/10.1145/3167132.3167301

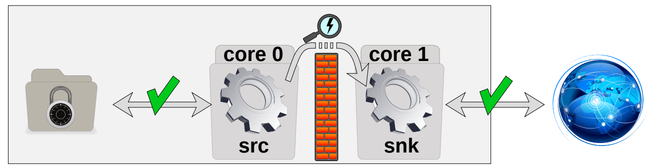

Figure 1: The source application (src) has access to restricted data, while the sink application (snk) has access to the internet. Although source and sink are isolated from each other, they manage to establish a covert channel by observing the processor power dissipation, compromising the security paradigm of permission separation and application isolation.

1 INTRODUCTION

Multicore processors are increasingly used in general purpose as well as embedded applications, despite two major problems associated with current architectures: (i) the high maximal power density may lead to overheating, and (ii) a low average power dissipation is required, while delivering high quality of service despite a large variability of the processing load. To deal with these problems, different power management schemes have been developed and implemented in computing systems. Implementations range from simple hardware based Dynamic Thermal Management (DTM), which impose a high performance penalty on the system, to more sophisticated methods like sleep states and Dynamic Voltage and Frequency Scaling (DVFS). Sleep states are specific power saving modes that drastically reduce the power dissipation using, for example, clock gating or flushing and turning off the cache. DVFS allows the Operating System (OS) to change the operating frequency and operating voltage of the processor cores according to the system utilization and can therefore reduce the power dissipation without a noticeable performance impact. On some architectures, DVFS also allows overclocking of the processor cores to deliver high performance for a limited time. To allow such a control of the clock speeds of the cores, without exceeding their physical limitations and causing overheating, the power management needs more detailed information than the common performance counters. Such detailed information can be for example temperature or power measurements, which are offered by many processor architectures. In a system where many different applications with different security clearances share the set of cores, these measurements can be used to compromise the systems security framework.

In this work, we investigate the potential security threat that can arise from accessible processor power information, by using power measurements on Intel cores. In particular, we are considering a scenario similar to the one presented in Figure 1. Here, a simple dual core system is only running the basic OS services besides our two attack applications, the *source* application `src` and the *sink* application `snk`. The source application has access to highly sensitive information, i. e. cryptographic keys, but it cannot use any communication interfaces. Contrary, the sink application has no access to highly sensitive information but it has access to the communication interfaces and can send data to third party servers. The two applications are isolated from each other following the security paradigm of privilege separation and application isolation. In case of low system load, the source application can take advantage of the power management and sleep states to modulate the power dissipation. If the sink application is capable of logging the power dissipation and forward the data to a third party server for analysis, the source and sink application can establish the so called *power covert channel*. By establishing the power covert channel, the source and sink application break the security paradigm of privilege separation and application isolation.

Contribution. In this work, we present a detailed study on a covert channel based on processor power information: the power covert channel. We present the following contributions:

(1) A generally applicable method to derive a tight channel capacity bound for covert channels with similar characteristics as the power covert channel. The results help to estimate the security threat caused by the power covert channel.
(2) To the best of our knowledge, we are the first ones to show an implementation of a communication scheme that proves the functionality of the power covert channel on Intel-based platforms.

Section 2 presents related work and Section 3 the basis of the power management in Linux. The sections 4 and 5 outline the channel and the threat model, which are the theoretical basis of our channel analysis. The implementation of the power covert channel is outlined in Section 6 and the derivation of the channel capacity bound is presented in Section 7. In Section 8 we present the experimental analysis for two different Intel-based platforms and we give concluding remarks in Section 9.

2 RELATED WORK

Lampson [8] first discussed privilege separation and the security issues connected with it, defining the *confinement problem* and identifying *side* and *covert channels* as main issues. While side channels allow an attacker to infer sensitive information by observing the system, covert channels are used for direct communication of two entities without the knowledge of a controlling entity, i. e. the OS. In this work, we study a covert channel as the source and the sink application directly communicate with each other by modulating the power dissipation of the core.

Architectural Side and Covert Channel. Modern multicore systems with their complex architecture have proven to be especially prone to side channel and covert channel attacks [4, 22]. Previous work showed that by exploiting shared caches it is possible to disclose the existence of other virtual environments via a side channel attack [20], or that it is possible to establish covert channels between isolated applications [23, 24]. In more recent work, Rong et al. [18] presented an improved methodology to establish covert channels in systems with shared cache, called Cloud Covert Channel based on Memory Deduplication (CCCMD). These cache side and covert channels have been refined [11, 16] or targeted to compromise special hardware like the Intel SGX [5]. Other security issues due to architectural characteristics have been shown by Evtyushkin et al. [3]. The authors present how an application can manipulate the shared branch predictor table such that it is possible to establish a robust, noise-free, high-capacity covert channel. Hunger et al. [7] introduced a mathematical abstraction called the *bucket model*, which is capable of capturing the common characteristics of different micro-architectural side and covert channels and derive their capacities. In a similar manner, we will also take advantage of an architectural feature of our platforms and use a theoretical model to analyse the power covert channel in detail.

Device Power Related Side and Covert Channels. Leaking information by influencing the timing of a device by heating it up (high power dissipation) or letting it cool (low power dissipation) has been well studied. Murdoch [13] showed that it is possible to use temperature induced changes in the clock skew of the timestamps, in response packets of a server, to identify servers within the Tor network. This technique was later improved by Zander and Murdoch [26] by minimizing the jitter in the clock skew using a synchronization mechanism, and the capacity of this channel was quantified as 20.5 bits per hour [25]. Similarly, Ristenpart et al. [17] used the changing clock skew of a device due to temperature changes to identify the physical infrastructure a Virtual Machine (VM) resides on. Moreover, the authors could also use this information to determine which VMs share this infrastructure. Power related covert channels have been shown by Guri et al. [6]. The authors established a covert channel between two air-gapped systems by only using the temperature sensors on the two systems. In a similar work, Masti et al. [10] showed the possibility of such a temperature based covert channel within a processor, to establish a data transmission between different application which are run sequentially on the same core or between two different cores in a multicore system. Bartolini et al. [2] later analysed these covert channels between multiple cores in a processor in detail. The authors derived capacity bounds for the different inter-core channels in the order of 300 bits per second (bps) and presented an improved implementation of the covert channel that achieved a throughput of up to 50 bps with less than 1% error probability. Similar to Bartolini et al. [2], we will also use a processor parameter to establish a covert channel that allows applications on the same device to communicate and therefore violate the security paradigms enforced by the OS.

Side and Covert Channel based on Device Power. In more closely related work, Michalevsky et al. [12] presented a methodology that allows location tracking of a mobile device based on the power dissipation. The authors generated a power-map of an area using power fingerprints for every location. Using this map, the authors were able to reconstruct the movement trajectory of a mobile device by analysing the power trace. Spolaor et al. [19] showed that it is possible to extract data from a charging phone by modulating the amount of power which is taken in via the charger, by changing the utilization of the mobile phones processor. The authors manipulated the charger to measure the current input to the

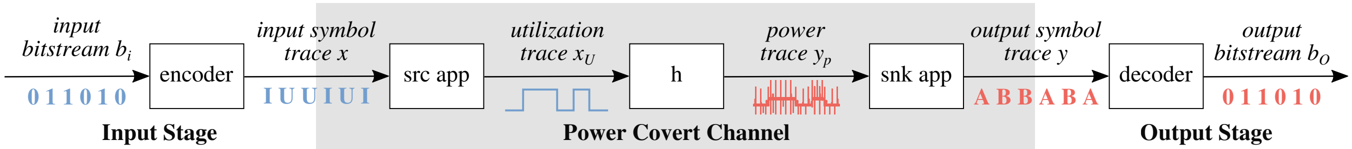

Figure 2: The proposed system abstraction model for a power covert channel.

phone and could reconstruct the leaked data stream by analysing the recorded power trace. In contrast to Spolaor et al. [19], in this work we will not use any external devices to establish the covert channel but only rely on internal device measurements. Further, we will directly transfer data from one application to another within the device, which can leak the data later via conventional communication interfaces.

3 POWER MANAGEMENT IN LINUX

The power covert channel depends on variations of the power dissipation of the device cores. As we base our experimental setup on Linux systems, we will give a brief insight into the two main parts of the Linux power management important for this work: the `cpufreq` and the `cpuidle` subsystem. We have to note, that although we execute all our experiments on Linux, Intel also provides power measurements on other OS's like Windows or Mac OS, which can be tested using the Intel Power Gadget API[1].

`cpufreq`. The `cpufreq` subsystem is responsible for the power management during the active time. Its main purpose is to scale the operating frequency and voltage according to the system utilization, based on a so called governor policy [15]. In our experimental evaluation we will use the *acpi-cpufreq* driver, which offers various governor policies. Among the most notable governor policies are the *conservative* and the *userspace* governor. While the conservative governor scales the operating frequency of the cores stepwise up or down depending on whether the utilization is above or below certain thresholds, the userspace governor allows direct control of the operating frequency from the userspace via *sysfs* nodes.

`cpuidle`. The `cpuidle` subsystem controls the sleep mechanisms of the device also based on governor policies [14]. Whenever a core is not utilized, `cpuidle` decides to which sleep state, also called *C-state*, that core is sent. Deeper C-states have a higher power saving effect but also have a higher exit latency. The standard implementation of `cpuidle` offers two governor policies, the *ladder* and the *menu* governor. While the ladder governor selects the C-state with a step-wise approach, moving down from the shallowest to the deepest C-state, the menu governor is more sophisticated. The menu governor selects the deepest possible C-state by evaluating various parameters like the expected core sleep time, latency requirements or the last C-state used by the core.

Power Measurements. In order to optimize the power dissipation and maximize the performance of the processor up to its physical limits, apart from system utilization, power measurements are also needed. As an example, the *Intel Turbo Boost* allows processor overclocking in case of high performance needs. The system can overclock the processor for a certain amount of time without overheating if the processor has sufficient overclocking budget. To determine the remaining overclocking budget, the system uses power and thermal measurements. The granularity of the power measurements depends on the platform. For example the platforms used in this work provide only one power measurement for the entire multicore processor.

[1]https://software.intel.com/en-us/articles/intel-power-gadget-20

4 CHANNEL MODEL

The proposed system abstraction model in Figure 2 is composed of three parts: the Input Stage, the power covert channel and the Output Stage.

Input Stage. The encoder performs channel coding to convert the input bitstream b_i with bits $b_i[k]$ into the input symbol trace x containing symbols $x[k]$. One symbol can, for example, represent a specific power dissipation of the processor. The set of feasible symbols depends on granularity of the power measurements, i. e. the number of observable levels. The input symbol trace x is transferred to the power covert channel.

Power Covert Channel. In the proposed system model, the source and the sink applications, `src` and `snk`, are part of the power covert channel. Input to `src` is the input symbol trace x generated by the encoder component. The source application `src` converts the input symbol stream x to an utilization trace x_U with core utilizations $x_U[k]$ and it applies these utilizations at run-time. The power trace y_p with power values $y_p[k]$ is obtained as a result of the transformation h. In our model, h depends on the platform configuration (i. e. the power management) and power characteristics (hardware specific parameters) and transforms the current system utilization to the corresponding power value. The transformation h cannot be determined easily, but in Section 7 we show that h can be time-invariant or time-variant, depending on the platform configuration. The sink application `snk` observes the power changes and generates the corresponding output symbol trace y with symbols $y[k]$. We can state that the power covert channel has the following three characteristics: (i) It is time-discrete; we can represent every channel usage as a single sample k as the Model Specific Registers (MSRs) used for the power measurements are just updated with a period of T_{msr}. (ii) It is value-discrete, as there is only a limited set of power values. As we will show later in Section 7, the power trace represents a sequence of discrete system states and can therefore be considered to contain discrete values. A system state is defined by the number of active cores at a certain operating frequency. (iii) It is noise-free. The power values are taken from noise-free system power counters and any measurements artifacts in the power trace are not visible in the output symbol trace y, due to the conversion from power values to symbols by the sink application. Due to those characteristics, one symbol $y[k]$ can, for example, directly represent a system state which corresponds to the power dissipation of the cores.

Output Stage. A decoder converts the output symbol trace y to a bitstream b_o. In an error-free information transfer, the output bitstream b_o equals the input bitstream b_i.

Our model enables us to better understand the power covert channel and to determine a capacity bound (Section 7). We also refer to this model when we design our test environment and the experiments to evaluate the channel (Section 8).

5 THREAT MODEL AND TARGET SETUP

Our threat model is based on the scenario outlined in Figure 1. Furthermore, we assume that the sink application can record a power trace, which it can forward via the Ethernet interface to a third party server. As presented in Section 3, the power management is based on the current system utilization. We assume that the device is idle during the time of the attack, i. e. a Laptop in an office during a weekend or during the night. Therefore, the utilization of the device is mainly controlled by the source application. Besides the system utilization, the source application cannot control any of the systems parameters, i. e. the `cpufreq` or `cpuidle` governors, as this would require special permission. Compensating any influences of these system parameters would require specific knowledge about the attacked platform before the attack and/or calibration of the source and sink applications. We give a basic assessment of some external influences on the power covert channel and possible countermeasures in Section 8. However, due to space constraints we cannot address all of these issues in detail.

We demonstrate the power covert channel on the example of Intel-based platforms. There are two reasons for our choice: (i) they allow power measurements through energy estimations provided by the system via MSRs, and (ii) Intel-based processors are the most significant platform type in the server, desktop and laptop market, with more than 80% market share in each sector in 2014 reported by Forbes[2]. In this work, we consider the following platforms:

(1) A Lenovo ThinkPad T440p laptop based on a 4th generation Intel Core i7-4710MQ quad-core processor. The CPU supports two hyper-threads per core and allows operating frequencies between 800 MHz and 2.4 GHz as well as turbo boost of up to 3.5 GHz.
(2) A server rack based on a 3rd generation Intel Xeon E5-2690 octa-core processor. This processor also features two hyper-threads per core and allows operating frequencies from 1.2 GHz up to 2.9 GHz and a turbo boost of up to 3.8 GHz.

For the rest of this paper, we will refer to platform 1 as *Laptop*, and platform 2 as *Server*. In favour of repeatability of our experiments, we define a controlled scenario which is used throughout all experiments, unless stated differently. Both platforms are situated in a server room with an average ambient temperature of $\approx$ 23 C° and run Ubuntu, version 16.04.02 on *Laptop* and version 14.04.2 on *Server*. Also, both platforms use the `cpuidle` menu governor and the `cpufreq` userspace governor, such that the operating frequency is locked to the maximum. By locking the operating frequency to the maximum we can perform an initial analysis of the power covert channel shown in Section 7 and 8. Moreover, the source and the sink application are pinned to specific cores during the experiments using `pthread_setaffinity_np()` and are run with the highest priority with `SCHED_FIFO` scheduling class to minimize the scheduling artifacts using `pthread_setschedparam()`. Our initial experiment showed that any further differentiation between C-states that are deeper than `C1E` is not possible; therefore, we set the maximum wakeup latency to 10 μs to limit the maximum C-state to `C1E`[3].

6 CHANNEL IMPLEMENTATION

The core of the power covert channel implementation consists of two applications, source and sink.

Source Application. The source application needs a core list and an execution trace as input. It creates as many threads as cores specified in the input list and pins them to the defined cores. Then it replays the execution trace by activating as many threads as specified and sending the rest of the threads to idle-state using `usleep()`. Active threads will execute a tight loop similar to the cpuburn benchmark[4] to ensure that the cores are not sent to a C-state by the `cpuidle` subsystem, while cores with idle threads will be sent to a C-state. All timing checks are done using `gettimeofday()`, which proves sufficiently lightweight and accurate for our task. In addition, the application checks the overall timing and adapts the execution trace to avoid timing drifts due to jitter in the execution of the tight loop and `usleep()`.

Sink Application. The sink application samples the MSR for the `PP0` power plane (`MSR_PP0_ENERGY_STATUS`) with a sampling rate T and immediately converts the samples to power values. These power values are then kept in an in-memory log which is dumped to a file as soon as the execution is stopped. Similar to the source application, the sink application uses `gettimeofday()` to monitor the overall timing to adjusts T to avoid a long term timing skew.

7 CHANNEL CAPACITY BOUND

Equation (1) shows how to calculate an upper bound C for the capacity per channel use of a noise-free channel, see for example MacKay [9, Chapter 17]. Here, N denotes the number of uses of

$$C = \lim_{N\to\infty} \frac{1}{N} \log M_N [bit] \quad (1)$$

the channel, and M_N denotes the number of distinct and feasible symbol series that could be sent by using the channel N times.

Following the derivations, the constraints on feasible symbol sequences can be used to determine M_N. To this end, we construct a state diagram, where the states S represent the states of the channel. Every valid path in the state diagram corresponds to a sequence of transitions. Consequently, every state transition in the diagram can represent a symbol that is forwarded to the channel. Starting from the initial state of the channel, M_N is equal to the number of distinct paths of length N in the state diagram. The state diagram

[2] https://www.forbes.com/sites/rogerkay/2014/11/25/intel-and-amd-the-juggernaut-vs-the-squid/#327951fe2981

[3] The wakeup latencies can be read via the sysfs interface at `/sys/devices/system/cpu/cpui/cpuidle/state/n/latency`.

[4] https://patrickmn.com/projects/cpuburn/

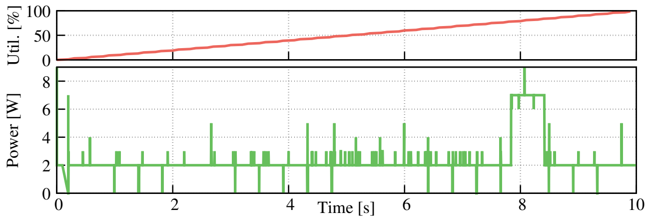

Figure 3: The power dissipation does not correlate to increasing utilization from 0% to 100% on a single core.

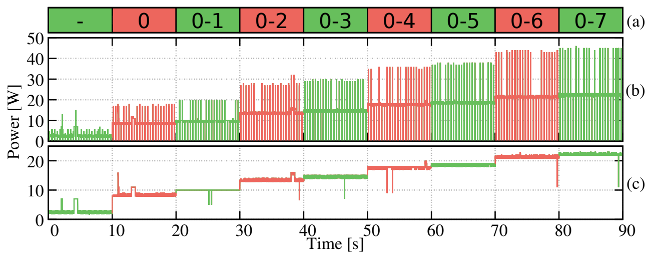

Figure 4: When setting a fixed operating frequency we are able to identify 5 system states depending on the number of fully utilized cores (a), if we apply median filtering to the raw power trace (b) to obtain the power trace (c).

of the channel can be derived from the results of a set of simple experiments, as we illustrate here for *Laptop*.

First, we need an initial experiment to determine if the utilization of a single active core influences the power measurements. Therefore, we ramp up the utilization of one logical core from 0% to 100% to check whether the resolution of the power trace is fine enough to detect different utilization levels. The results for *Laptop* are depicted in Figure 3 and show some fluctuations for different utilizations. However, it is not possible to draw a direct connection from the utilization trace to the power trace, as long as the core is not in any C-state. We can state that utilization changes may cause measurement artifacts in the power trace, but do not have an influence on the measured average power.

Next, we conduct an experiment to determine the number of system states for a *fixed frequency* case, by using the highest operating frequency. This allows us to determine the channel performance in a static scenario, a time-invariant transformation from utilization to power h (see Section 4). In Figure 4, the top plot (a) shows the set of fully utilized logical cores (utilization is 100%) in a 10 s time interval, (b) the power trace with a sampling rate of 1 ms, and (c) the power trace median filtered with a window size of 8 samples. The experiment reveals that there is a high amount of measurement artifacts which may cause errors at higher symbol rates and can be explained by following MSR characteristics: (i) reading is destructive, this means that the value is set to 0 after reading, and (ii) according to Intel® 64 and IA-32 Architectures Software Developer's Manual [1] the MSR is updated "*approximately*" every 1 ms. Due to these two characteristics, it can happen that the sink application reads the MSR twice between two updates, causing a 0 in the power trace. Moreover, two logical cores are mapped to

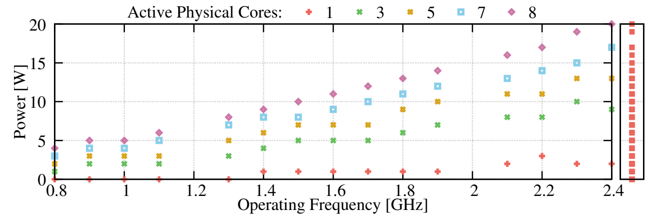

Figure 5: By varying the operating frequency more system states can be exposed. Due to the lack of knowledge on the operating frequency, the sink application can only distinguish 20 system states, as illustrated in the power plane to the right.

one physical core, for example logical cores 0 and 1 are mapped to physical core 0. Therefore, we can identify 5 system states in our power trace for the fixed frequency case: (i) from time 0 to 10 s when no physical core is utilized, (ii) from 10 to 30 s when one physical cores is utilized, (iii) from 30 to 50 s with two utilized physical cores, (iv) from 50 to 70 s with three physical cores utilized, and (v) from 70 to 90 s where all four physical cores are utilized. This equals $N_C + 1 = 5$ states for *Laptop*, where N_C is the number of physical cores.

The usage of a different `cpufreq` governor makes the channel more complex as the power levels, respectively the system state, also depend on the used processor frequency. Considering our channel model from Section 4, this means that the transformation from utilization to power h is time-variant. Therefore, we repeated the experiment illustrated in Figure 4 for every operating frequency of *Laptop* to exploit all possible transformations h and the respective utilization to power transformation. Figure 5 illustrates the identified system states for different number of active physical cores and different operating frequencies, whereas one point equals the integer-rounded mean of the power trace values within one system state, i. e. the integer-rounded power mean of interval 0 to 10 s from Figure 4 is represented as point for 0 active physical cores at frequency 2.4 GHz in Figure 5. The right scale presents the projection of all system states onto the power plane, as the sink application cannot determine the operating frequency of the cores. The sink application can only determine the power trace. The power plane shows that there are 20 distinguishable system states in an *variable frequency* case where the operating frequency changes can be fully controlled. We do not include the operating frequencies reachable through the Intel turbo boost into our analysis because these frequencies can only be reached under certain conditions and cannot be forced through the userspace governor.

From the results of the fixed frequency and variable frequency case we can derive the state diagram and further the $S{\times}S$ connection matrix $\mathbf{A}$. An element $A_{s,s'}$ is 1 if there is a transition from state s to s' and 0 otherwise. For our two cases fixed and variable frequency all elements of $\mathbf{A}$ are one. In other cases where a real governor does not allow all kinds of frequency changes, the size of $\mathbf{A}$ is still the same for the variable frequency case. The difference between the variable frequency case and a real governor case is that not all frequency transitions are possible and therefore not all elements of $\mathbf{A}$ would be 1, leading to a lower channel capacity bound. As long as we know all possible state transitions, we can determine M_N using $\mathbf{A}$ independent of the setup. To this end, we count the transitions to a state by means of Equation (2) and Equation (3). $\mathbf{c}^{(0)}$ is the initial state vector consisting of one 1, representing the initial state, and zero otherwise. $\mathbf{c}^{(n)}$ holds the number of paths

$$\mathbf{c}^{(n+1)} = \mathbf{A}\mathbf{c}^{(n)} \tag{2}$$

$$\mathbf{c}^{(N)} = \mathbf{A}^N\mathbf{c}^{(0)} \tag{3}$$

$$\lim_{N\to\infty} \mathbf{c}^{(N)} = \text{constant} \cdot \lambda_1^N \cdot \mathbf{e}_1 \tag{4}$$

$$M_N = \sum_s c_s^{(N)} \tag{5}$$

that lead to a certain state after n uses of the channel. In the limit, the principal eigenvalue of $\mathbf{A}$, i.e. the eigenvalue with the largest absolute value, starts dominating the iteration in Equation (3). As a result, we obtain Equation (4), which shows that the dominating term of $\mathbf{c}^{(N)}$ is λ_1^N. Here, λ_1 is the principal eigenvalue of $\mathbf{A}$ and $\mathbf{e}_1$ is the corresponding eigenvector. The number of possible paths is calculated as shown in Equation (5), where $c_s^{(N)}$ is element s of the vector $\mathbf{c}^{(N)}$. We can now use Equation (4) and Equation (5) in Equation (1) to obtain Equation (6). Inserting the parameters for

$$C = \log_2 \lambda_1 \tag{6}$$

Laptop, we can derive the upper channel capacity bound for the variable frequency case of 4.32 bits per channel use and a channel capacity bound of 2.32 bits per channel use for the fixed operating frequency case.

We also performed the same experiments and evaluation on our second platform *Server*. This evaluation showed that on *Server* we can identify $N_C + 1 = 9$ states for the fixed frequency case and 13 states for the variable frequency case. We can derive the upper channel capacity bound of 3.17 bits per channel use for the fixed frequency case and a channel capacity bound of 3.70 bits per channel use for the variable frequency case on *Server*.

Knowing the capacity bound and the update period of the MSR $T_{\text{msr}} = 1$ ms, we can calculate the theoretical bandwidth of the channel as shown in Equation (7). This yields a maximum bandwidth

$$B = \frac{C}{T_{\text{msr}}} \tag{7}$$

of $B_{\text{max}} = 2322$ bps for *Laptop* and 3170 bps for *Server* considering a fixed operating frequency (time-invariant utilization to power transformation h). According to the US department of defence 1985 *Orange Book* [21], "a covert channel bandwidth that exceeds a rate of one hundred (100) bits per second is considered high". Based on this definition, we can state that the power covert channel is a high risk channel in terms of capacity.

8 EXPERIMENTAL ANALYSIS

To evaluate how close the throughput of a simple implementation of the power covert channel gets to the channel capacity bounds, we deployed our channel setup (see Section 6) on the two platforms *Laptop* and *Server* (see Section 5). The sink application used a sampling rate T of 1 ms, as oversampling of the MSR did not improve signal quality without implementing a sophisticated zero replacement and filtering scheme. All data transmissions are initiated with a pulse with the maximum power difference, to synchronize the source and the sink application. Furthermore, we used different source codes for transmission, where each symbol is equal to a system state: (i) the Binary code defines a zero as no active core and a one as 4 cores active, (ii) a Huffman code with 5 states called Huffman 5, and (iii) a Huffman code with 9 states called Huffman 9 for *Server* only. The Huffman codes exploit all of the available states for the two platforms *Laptop* and *Server*, respectively. The decoding is done according to the block diagram in Figure 6. We first filtered the signal with a moving median filter to remove outliers. Equation (8) ensures that the window length of the median filter

$$\omega_i = \min\left\{\max\left\{2 \cdot round\left(\frac{N_i}{3}\right) + 1, 1\right\}, 9\right\} \tag{8}$$

decreases proportional to the samples per symbol. The window length of the median filter ω_i at bitrate i was calculated according to Equation (8), where N_i is the number of samples per symbol. The minimum and maximum filter length 1 and 9 as well as the proportionality factor of 3 where evaluated experimentally towards a minimal bit error probability. After filtering, the signal is oversampled such that there is an odd number, or at least 11 samples per symbol. The signal was interpolated using nearest-neighbour interpolation. As last steps, the signal is quantized according to the known platform specific power levels using a majority voter and then decoded.

8.1 Controlled Environment

Figure 7 shows the average error probability for eight runs for both platforms *Laptop* and *Server*. Each run consists of a 5000 bit random message transmitted at a specified bitrate, for bitrates from 1 to 2200 bps with a spacing of 10 bps. Due to the platform limitations, the Binary code only allows a maximum bitrate of 1000 bps (see Section 7).

The analysis of the results presented in Figure 7 shows that in general *Laptop* performs better than *Server*. For *Laptop* transmissions with an error probability of less than 15% are possible with bitrates of almost 500 bps using the Binary code and bitrates of almost 1000 bps using the Huffman 5 code. In contrast, on *Server* the error probability already exceeds 15% at bitrates of around 200 bps for Binary and 100 bps Huffman 5 coding. The experiments also show that the use of all 9 system states on *Server* is not possible as the transmissions with the Huffman 9 code always causes an error probability of around 40%. A detailed analysis of the experimental data of *Server* shows that the majority of the errors on *Server* can be traced back to architectural properties. Despite the fact that we limit the wakeup latency for cores to return from C-states, the power trace does not follow our input utilization trace fast enough. This can be caused by higher latencies when waking up from, and sending cores to C-states. Moreover, the power traces collected from *Server* show that there is slow long term rise in the power trace. This can be caused by the rising temperature of the core, which also causes higher power dissipation. While the latencies cannot be compensated in the symbol decoding, de-trending methods can be used to nullify the long term rise of the power traces.

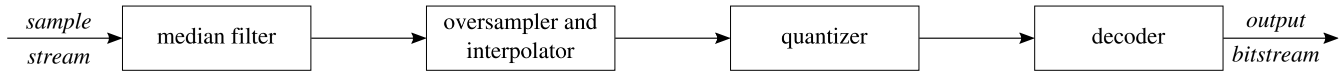

Figure 6: The signal decoding is performed in multiple stages.

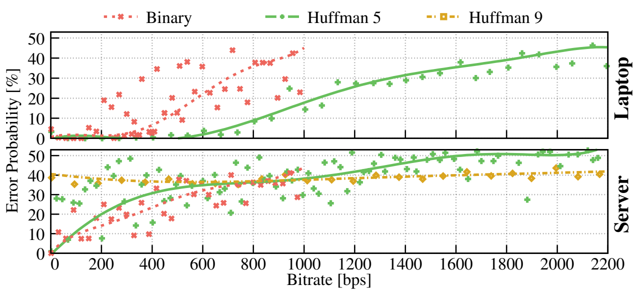

Figure 7: *Laptop* (upper plot) shows better performance in terms of error probability for rising bitrates than *Server* (lower plot) for different source coding schemes.

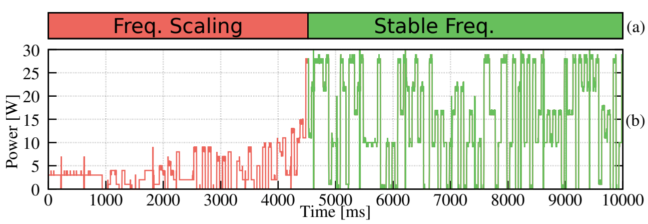

Figure 8: If the operating frequency is not stable, the data transmission is disturbed as the frequency scaling leads to a time-variant utilization to power transformation h.

Another problem of the power covert channel, which applies to both platforms *Laptop* and *Server*, is the synchronization between source and sink application. Inaccurate synchronization leads to higher error probabilities at higher bitrates as there are less samples per symbol, but this could be handled with a more sophisticated signal processing. Finally, experiments have shown that the power measurements representing the system states can vary slightly for repeated executions of the same utilization trace. To compensate varying power measurements, an improved implementation of the power covert channel needs to be capable of adapting the power threshold for system state detection according to the current trace. Adaptive system state detection can be implemented using a calibration header in the data transmission and threshold detection during decoding.

8.2 Robustness

To briefly evaluate the robustness of the power covert channel, we analyse two scenarios on *Laptop*: (i) the *Variable Frequency* case where the conservative governor is used to set the operating frequency of the cores, and (ii) the *Application Interference* case where a second source application is used to permanently keep all but one physical core active.

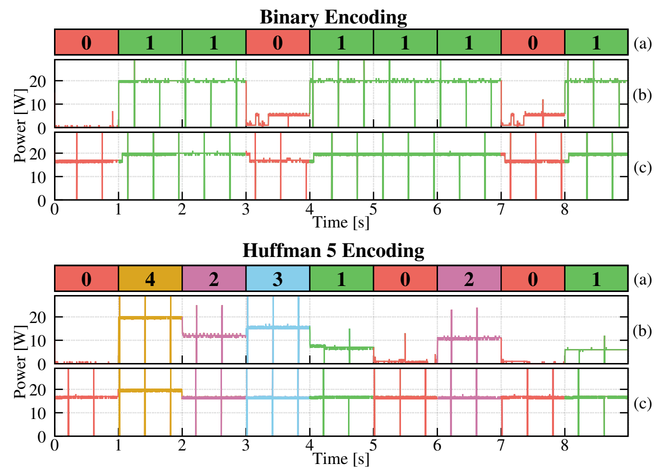

Figure 9: Binary encoding is more robust against interference as the symbols (a) can be distinguished without (b) and with interference (c), while a distinction of all symbols is not possible anymore for Huffman 5 encoding.

Variable Frequency. The usage of frequency governors adds more disturbance to the channel as the transformation from utilization to power h (see Section 4) is time-variant. Figure 8 illustrates the start of a data transmission in a system using the conservative governor and Huffman 5 encoding at 100 symbols per second. During the start of the transmission power measurements increase as the system is scaling up the frequency due to the high utilization generated by the source application. At approximately 4500 ms the system reaches the maximum operating frequency and stays in that state. Therefore, the system is now in a stable state during which h can be considered time-invariant and the power measurements can be properly mapped to the system states and decoded. We can state that the effects of a governor on the power covert channel can be compensated with a more sophisticated coding scheme, i. e. for the widely used *conservative* governor presented in this example, only a simple preamble and special message coding is needed to ensure the system is in a stable state during the transmission. A preamble that fully utilizes the cores would force the system to scale to the highest frequency and a message encoding that ensures that at least one core has a utilization higher than the lower threshold of the conservative governor will establish the same frequency configuration as in the controlled environment scenario.

Application Interference. In this scenario, a second source application is occupying all but one physical core. This prohibits the occupied cores from entering C-states for power saving and therefore some system states cannot be reached. Figure 9 illustrates power traces for the controlled environment and the inference scenario for both encoding schemes Binary and Huffman 5. The figures show that only a differentiation between two states is possible, therefore the transmission with Binary encoding is still possible while the transmission with Huffman 5 is disturbed. These experiments show that a transmission is viable whenever the number of states needed by to transmit $N_T \leq (N_C - N_I + 1)$, where N_C is the number of physical cores available on the system and N_I is the number of physical cores occupied by interfering applications. Furthermore, an attacker can always use the power measurements to detect when the platform utilization is sufficiently low to start the transmission (see Section 5).

Our experiments show that compared similar cover channels, like the thermal covert channel [2], the power covert channel channel allows a higher throughput but is less robust to disturbances by other applications on the platform.

9 CONCLUDING REMARKS

In this work we present the power covert channel, a covert channel based on processor power measurements. For our evaluation we chose broadly used Intel-based platforms, which offer power measurements via a MSR, and therefore allow power measurements. We present a detailed theoretical analysis and derived a capacity bound for the power covert channel. We considered a fixed operation frequency of the platform during the transmission, which resulted in a capacity bound of 2322 bps for a platform with 4 physical cores and 3170 bps with 8 physical cores. Our methodology to derive the channel capacity bound is generally applicable to other covert channels with similar characteristics as the power covert channel. Moreover, we conducted a thorough experimental analysis to exploit achievable throughputs under controlled conditions and evaluated the robustness of the power covert channel against external influences. Our experiments showed that under controlled conditions we can achieve throughputs of up to 1000 bps with an error probability of less than 15% using a very simple channel implementation.

For the power covert channel we present a high channel capacity bound and high throughput for a simple implementation, which shows the channel's potential to leak information. Furthermore we found that the power covert channel is more prone to disturbance than comparable covert channels and a successful attack requires detailed knowledge about the attacked platform utilization pattern, architecture and power management.

ACKNOWLEDGEMENTS

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644080.

This work was supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 15.0025. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the Swiss Government.

The authors would like to thank Rehan Ahmed, Stefan Draskovic, Roman Trüb and Davide Basilio Bartolini for their valuable feedback to get this paper camera ready.

REFERENCES

[1] 2016. Intel® 64 and IA-32 Architectures Software Developer's Manual. *Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B and 3D* (2016).

[2] Davide B. Bartolini, Philipp Miedl, and Lothar Thiele. 2016. On the Capacity of Thermal Covert Channels in Multicores. In *Proceedings of the Eleventh European Conference on Computer Systems (EuroSys '16)*. ACM, New York, NY, USA, Article 24, 16 pages. https://doi.org/10.1145/2901318.2901322

[3] Dmitry Evtyushkin, Dmitry Ponomarev, and Nael Abu-Ghazaleh. 2015. Covert Channels Through Branch Predictors: A Feasibility Study. In *Proceedings of the Fourth Workshop on Hardware and Architectural Support for Security and Privacy*. 5:1–5:8.

[4] Apostolos P Fournaris, Lidia Pocero Fraile, and Odysseas Koufopavlou. 2017. Exploiting Hardware Vulnerabilities to Attack Embedded System Devices: a Survey of Potent Microarchitectural Attacks. *Electronics* 6, 3 (2017), 52.

[5] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. 2017. Cache Attacks on Intel SGX. In *EUROSEC*.

[6] Mordechai Guri, Matan Monitz, Yisroel Mirski, and Yuval Elovici. [n. d.]. BitWhisper: Covert Signaling Channel between Air-Gapped Computers using Thermal Manipulations. ([n. d.]). http://arxiv.org/abs/1503.07919

[7] C. Hunger, M. Kazdagli, A. Rawat, A. Dimakis, S. Vishwanath, and M. Tiwari. 2015. Understanding contention-based channels and using them for defense. In *Proceedings of the 21st IEEE International Symposium on High Performance Computer Architecture*. 639–650.

[8] Butler W. Lampson. 1973. A Note on the Confinement Problem. *Commun. ACM* 16 (Oct. 1973), 613–615.

[9] David JC MacKay. 2003. *Information theory, inference and learning algorithms*. Cambridge university press.

[10] Ramya Jayaram Masti, Devendra Rai, Aanjhan Ranganathan, Christian Müller, Lothar Thiele, and Srdjan Capkun. 2015. Thermal Covert Channels on Multi-core Platforms. In *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, Washington, D.C., 865–880. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/masti

[11] Clémentine Maurice, Manuel Weber, Michael Schwarz, Lukas Giner, Daniel Gruss, Carlo Alberto Boano, Stefan Mangard, and Kay Römer. 2017. Hello from the other side: SSH over robust cache covert channels in the cloud. *NDSS, San Diego, CA, US* (2017).

[12] Yan Michalevsky, Aaron Schulman, Gunaa Arumugam Veerapandian, Dan Boneh, and Gabi Nakibly. 2015. PowerSpy: Location Tracking Using Mobile Device Power Analysis.. In *USENIX Security Symposium*. 785–800.

[13] Steven J. Murdoch. 2006. Hot or Not: Revealing Hidden Services by Their Clock Skew. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS '06)*. 27–36.

[14] Venkatesh Pallipadi, Shaohua Li, and Adam Belay. 2007. cpuidle: Do nothing, efficiently. In *Proceedings of the Linux Symposium*, Vol. 2. Citeseer, 119–125.

[15] Venkatesh Pallipadi and Alexey Starikovskiy. 2006. The ondemand governor. In *Proceedings of the Linux Symposium*, Vol. 2. sn, 215–230.

[16] Danny Philippe-Jankovic and Tanveer A Zia. 2017. Breaking VM Isolation-An In-Depth Look into the Cross VM Flush Reload Cache Timing Attack. *International Journal of Computer Science and Network Security (IJCSNS)* 17, 2 (2017), 181.

[17] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. 2009. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 199–212.

[18] Hong Rong, Huimei Wang, Jian Liu, Xiaochen Zhang, and Ming Xian. 2015. WindTalker: An Efficient and Robust Protocol of Cloud Covert Channel based on Memory Deduplication. In *Big Data and Cloud Computing (BDCloud), 2015 IEEE Fifth International Conference on*. 68–75.

[19] Riccardo Spolaor, Laila Abudahi, Veelasha Moonsamy, Mauro Conti, and Radha Poovendran. 2017. No Free Charge Theorem: a Covert Channel via USB Charging Cable on Mobile Devices. In *International Conference on Applied Cryptography and Network Security*. Springer, 83–102.

[20] Kuniyasu Suzaki, Kengo Iijima, Toshiki Yagi, and Cyrille Artho. 2011. Memory Deduplication As a Threat to the Guest OS. In *Proceedings of the Fourth European Workshop on System Security (EUROSEC '11)*. 1:1–1:6.

[21] U.S. Department of Defense. 1985. *DOD Trusted Computer System Evaluation Criteria "The Orange Book" [DOD 5200.28]*.

[22] Zhenghong Wang and Ruby B Lee. 2006. Covert and side channels due to processor architecture. In *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*. IEEE, 473–482.

[23] Zhenyu Wu, Zhang Xu, and Haining Wang. 2012. Whispers in the Hyper-space: High-speed Covert Channel Attacks in the Cloud. In *Proceedings of the 21st USENIX Conference on Security Symposium (Security'12)*. 9–9.

[24] Xu, Yunjing and Bailey, Michael and Jahanian, Farnam and Joshi, Kaustubh and Hiltunen, Matti and Schlichting, Richard. 2011. An Exploration of L2 Cache Covert Channels in Virtualized Environments. In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop (CCSW '11)*. 29–40.

[25] S. Zander, P. Branch, and G. Armitage. 2011. Capacity of Temperature-Based Covert Channels. *Communications Letters, IEEE* 15, 1 (2011), 82–84.

[26] Sebastian Zander and Steven J Murdoch. 2008. An Improved Clock-skew Measurement Technique for Revealing Hidden Services.. In *USENIX Security Symposium*. 211–226.