UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

Dataset for Hardware Trojan Detection

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
(ETSETB), Universitat Politècnica de Catalunya
by

Sergi Mus

In partial fulfillment
of the requirements for the Master in
CYBERSECURITY

Advisors:
Dr. Beatriz Otero, UPC, Spain
Dr. Ramon Canal, UPC, Spain
Barcelona, July 2022

Abstract

Nowadays, cloud services rely extensively on the use of virtual machines to enforce security by isolation. However, hardware trojan attacks break this assumption. Within these attacks, cache side-channel attacks such as Spectre and Meltdown are the focus of this work. In this project, we develop (1) a set of tools to generate a dataset; and (2) a dataset that will allow the use of Machine Learning techniques to detect Spectre and Meltdown attacks (i.e. using a cache side-channel). When released, this dataset will enable researchers to compare their ML-based detection proposals based on the same dataset (which is not currently the case). Also, it eliminates the need of an infected computer to generate the attacks and the corresponding dataset for subsequent research studies.

Contents

Chapter 1

Introduction

This project has the goal of providing a better solution to the current problem of modern processor security, specifically of processor vulnerability exploitation.

One of the most dangerous kinds of security attacks in modern CPUs are side-channel attacks such as Spectre and Meltdown[1]. These attacks take advantage of modern CPU speculative execution plus cache-timing side-channel. A side channel attack is a way to extract sensitive information from a system by some means other than the intended input and output channels. On a computing system, there are well-known side-channels that may be exploited: electrical magnetic waves, power consumption, timing clues (i.e. memory access timing).

In order to understand the problem, we must first understand how modern processor architectures execute instructions. CPUs make use of what is known as a pipeline, a series of sequential blocks that take care of each of the sub-steps for instruction execution (see Figure 1.1). The main point of interest in this figure is that while the execution block is executing the current instruction, the dispatch block is decoding the next instruction simultaneously, and the fetch block is fetching two instructions after the current one. The problem arises when the instruction executed is a branch instruction (it can have two possible next instructions, like for instance a conditional jump instruction). In this case, the fetch and dispatch blocks have a Branch Prediction Unit that takes an informed guess as which instruction will be next in line. Moreover, even if we are not going to go into much detail, the processor can execute instructions out of order to maximize throughput.

Another relevant element of modern CPU architectures is the usage of cache memories. Modern CPUs can process beyond a terabyte of data per second, however modern volatile memories (i.e. DRAM) operate up to the gigabit scale. For this reason, most CPUs would be bottlenecked by memory. Cache is a volatile memory with a higher throughput (placed in

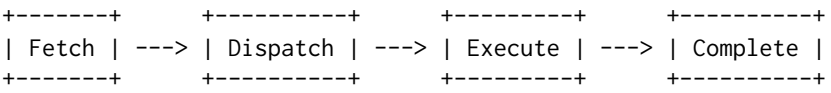


Figure 1.1: Modern processor simplified pipeline

the CPU itself), it prevents the CPU from accessing the main memory. There usually are several layers of cache as seen in Figure 1.2. Each of these layers stores a copy of a portion of the contents of main memory. With each increasing layer of cache, the speed gets slower but the size increases.

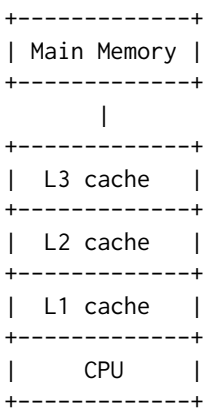


Figure 1.2: Simplified cache architecture

The instructions executed out of order can have side effects without the operating system checks restricting them. These side-effects are not committed to main memory, they take place in the cache. This allows an attacker to take advantage of these side effects of speculative execution to ex-filtrate data from one process to another by using the cache as a side-channel.

As it is a hardware level exploit, it could be dismissed as minor, however it breaks one of the most fundamental assumptions in cybersecurity. This exploit breaks userspace and kernel space memory isolation and virtual machine isolation. It allows to read the memory content from one guest virtual machine to the host or another virtual machine (see Figure 1.3).

This is specially relevant in today's world where most companies infrastructure is running on virtual machines in a cloud provider and not in exclusive owned servers. This means that essentially a corporate infrastructure is running in a virtual machine in a data center. If an attacker is able to buy another instance that runs in the same physical machine, it is able to ex-filtrate data from the legitimate company's instance. The attacker could even ex-filtrate cryptographic material

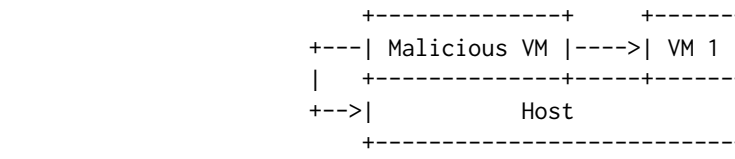


Figure 1.3: Virtual machine isolation breaking

from a running program as it can access any arbitrary memory location.

Patches for this vulnerabilities exist in numerous forms, the most straight forward is to upgrade the hardware which is often discarded due to its high cost. The other option available to users is to patch the software to disallow the conditions that can cause such vulnerability. For example, in the case of Linux, through a kernel patch. The main drawback of this approach is the hit to performance that it supposes. For some AMD processors the impact of this patch is only of 3% while on Intel processors the impact can reach values as high as 24.8% for the 8700k series processors [2].

This is why a new approach is arising in popularity. This approach consists in a more passive approach where we monitor for footprints of the attack and take appropriated measures to kill the attacking process. It is really similar to a host monitoring software (e.g. Sysmon and Prometheus) but instead of monitoring system events like calls to the cryptographic library, they monitor hardware counters. This approach is more effective than current ones because the performance hit is minimal, as the counting is performed by dedicated hardware that CPUs have already integrated and the overhead of analysing these counters is minimal.

In order to perform such detection, many papers are making use of Machine Learning. However, there is a lack of publicly available datasets to train the models. All the literature develops its own dataset to be trained with their own model. This makes the task to compare several methods impossible as they are each working under a different set of rules.

1.1 Related work

As previously mentioned, there are multiple articles attempting to perform similar approaches to ours. In this section, we are going to attempt to provide a view into their methodology, that will aid in guiding the project.

The first of such article is Mushtaq et al.[3] In this article the authors are interested in providing an overview of different Machine Leaning approaches and their performance. The algorithms tested are shown in Table 1.1. As shown in the table, the proposal tests both

Machine Learning Model	Category
Linear Regression (LR)	Linear
Linear Discriminant Analysis (LDA)	Linear
Support Vector Machine (SVM)	Linear
Quadratic Discriminant Analysis (QDA)	Non-linear
Random Forest (RF)	Non-linear
K-Nearest Neighbors (KNN)	Non-linear
Nearest Centroid	Linear
Naive Bayes	Linear
Perceptron	Linear
Decision Tree	Non-linear
Dummy	Non-linear
Neural Networks	Non-linear

Table 1.1: Mustaq et al. tested algorithms

linear and non-linear algorithms.

However, the main item of interest from the article is how they generate the dataset. In their case, they use an AES and RSA encryption routine as the victim process. During their attack they record the events shown in Table 1.2, as we can see the events recorded mainly focus on cache metrics. An interesting distinguishing factor about this article is the recording of the attack under several load conditions of the machine (Zero Load, Medium Load and Heavy Load). To set the load, some memory intensive benchmarks are used such as: gobmk, mcf, omnetpp, and xalancbmk.

The main issue with this paper is that for creating the load it uses none standard ways. Meaning that, while using memory heavy programs, the workload can vary a lot depending on the machine and how the program is used. This is one of the ways our project will attempt to improve. Furthermore, only memory load is used. Even if it is reasonable to consider that memory load will have a bigger impact on the program, it is important to consider other types of workloads and how they affect performance of the attack and recording.

The second issue with this article is the wide range of metrics used. Many vulnerable processors do not have access to as many counters. This means that, even if the accuracy obtain is very high, it is not widely applicable.

The second representative article is Depoix et al.[4] where the authors concentrate in Deep Learning. They solve the issues previously presented by only recording 3 events: total instructions, total cache misses for L3 and total cache accesses for L3. They use a proper profiling tool to generate load. This tool allows them not only to

Scope of Event	Hardware event
L1 Caches	Data cache misses
	Instruction Cache Misses
	Total Cache Misses
L2 Caches	Instruction Cache Accesses
	Instruction Cache Misses
	Total Cache Accesses
	Total Cache Misses
L3 Caches	Instruction Cache Accesses
	Total Cache Accesses
	Total Cache Misses
System wide	Total CPU Cycles
	Branch Miss-Predictions

Table 1.2: Mustaq et al. recorded events

generate memory load but also CPU load and other kinds of loads like a process spinning on sync().

1.2 Goals

The main goal of this project is to generate a dataset that serves as a common testing ground for future work in detecting cache side-channel attacks. For this reason, we want to focus beyond the generated dataset, we want to create an extensible tooling that allows for reproducible steps in dataset generation. Therefore, we are more interested into how to properly generate the dataset, than the generated dataset characteristics.

This approach is a direct consequence of the strict hardware limitations. As we only have one working computer available for performing the experiments, we will not be able to recreate the variety of contexts and hardware the attack will be performed.

The result of the project will be the dataset plus the tooling needed to perform the recording of hardware events as a trace. This means that the generated dataset will be a set of timestamps with the number of each event occurrences since the last timestamp and with a label indicating weather or not an event had place during this window.

1.3 Planning

The project can be separated into three main components, the first one is developing the recording software, the second is finding one or multiple attacks to be recorded and the final is to create the tools to run the experiments and analyse the results.

The first stage of the project will be concentrated into accessing the hardware counters. This counters can be configured to count several of the events taking place in the CPU. They will increment every time a particular event takes place in the CPU. An event can be any of the actions performed by the CPU from a clock cycle to an instruction successfully executing. Accessing this counters is very platform dependant as it is closely related to the hardware used.

The second stage consists in finding the processes that will run while the previous stage is recording. We could implement our own attack, but we deemed it unnecessary. Many publicly available implementations exist, using a well tested implementation removes the added complexity of working with our own attack. As a side effect, it allows the project to be as independent as possible of the attack so when a new variant is found it can be added to the dataset.

The third stage is creating and environment for running the previously generated software. This script will be in charge of ensuring that the recording process is running during the attack execution and it will be in charge of labeling the dataset generated by the recording process. Additionally, a tool to visualize the results should be available for the user in order to ensure through visual inspection that the results are coherent.

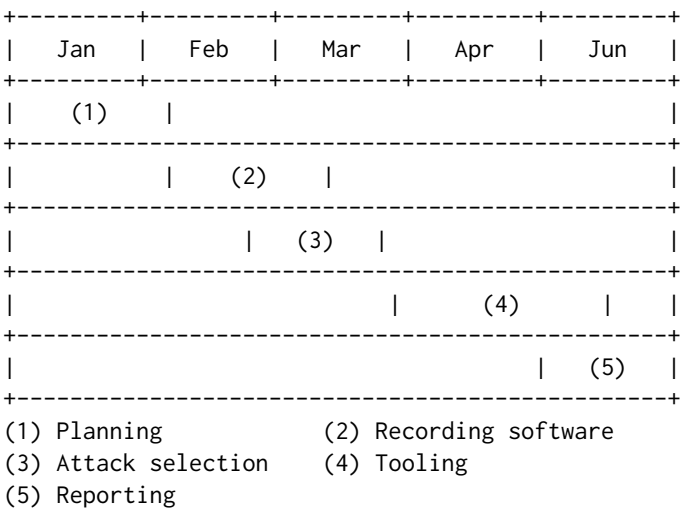


Figure 1.4: Gantt diagram

Chapter 2

Project development

2.1 Architecture

Figure 2.1 shows the monitoring stack in a system. This can seem not intuitive as the monitoring does not interact with the hardware. This relies in the kernel providing the access to the hardware counters. For performance monitoring it is often enough with this approach as it provides fullfils all the needs for this problem.

The main issue with this approach is that in our scenario, the attack is performed from an external process reading the contents of the critical process. Therefore the anomalies with the hardware counters will probably be located within the malicious process that is unknown to the victim. For this reason our monitoring system has to provide a system wide monitoring. Therefore it is a process that runs alongside all other processes in the machine. The process architecture ends up looking like Figure 2.2. We can clearly see that the monitoring process is just another process alongside the others that periodically polls the hardware counters.

The system library PAPI provides two APIs: a high level API and

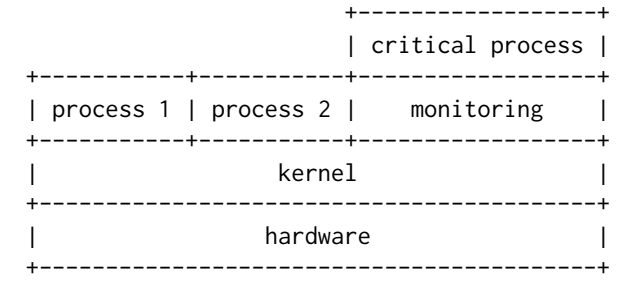


Figure 2.1: Classical hardware monitoring stack

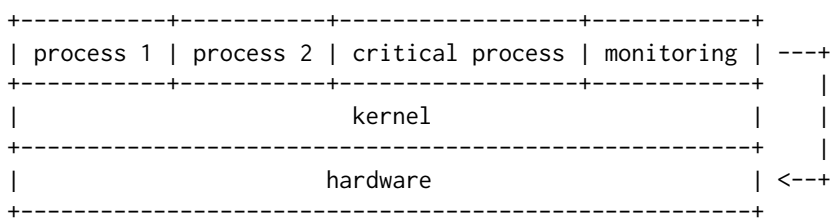


Figure 2.2: System wide hardware monitoring

a low level API. The high level API is designed for recording events inside instrumented regions. It is designed for simplicity not for flexibility. We can not use this API for our goals as it does not provide system wide counters. For this reason, we are forced to use the other API, the low level API. This API requires a lot more work for recording events, for example it requires manual initialization of the library and manual error checking. However, it allows the user a fine-grained control over the events recorded and the counters.

The library by default is not configured for monitoring events in the system but in a particular thread. In order to monitor system metrics we have to change the granularity and domain. The granularity can be set to thread, process, CPU or system. The domain allows to monitor only events in user-mode or to also include events in kernel-mode. An example of doing so can be found in the PAPI repository in the file:

src/components/perf_event/tests/perf_event_system_wide.c

In order to use PAPI, the kernel needs to be patched. The Linux kernel 2.6.32 and newer already include the patch so no kernel patching is required. If using the latest releases for most Linux distributions the patch will be included.

2.2 Monitoring facilities

As we want to fabricate a dataset, we want to define what will be the features that will later be used by Machine Learning models. The attack we wish to detect is a cache side-channel attack. As this kind of attacks use the cache to ex-filtrate data, if we wish to detect them, we must monitor the cache to detect them. Therefore for these reasons the metrics provided as feature in our dataset will be related to cache metrics. In order to access this metrics we must use some facilities provided by the processor. This facilities are hardware counters in the machine. This hardware counters count certain hardware events such as cache misses or number of instructions executed.

2.2.1 Performance API

As creating facilities to access the counter is beyond the scope of the project as it would require to patch the kernel. For this reason we use the very mature library called Performance Application Programming Interface (PAPI) [5]. PAPI provides an interface for low hardware level counters. It is so widely use with this goal that its kernel patch has been integrated into the Linux kernel.

One of the main problems that will be encountered during the development of the project is that PAPI was designed as an application that monitors performance. For this reason, most its facilities are built around accessing the hardware counters to monitor a particular section of code and not monitoring system wide.

Installation

The installation of the library is very modular as it provides facilities for not only recording events in the CPU but also in GPU through CUDA, and a variety of specialized processors. As our application is oriented to regular CPUs we can stick to the default configuration.

For the default configuration, many distributions already have prepared packages. In the project case, it was compiled from source for one main reason, when installing with the package manager we where unable to get CMAKE to locate the package and link it with the project executable.

Granularity

The events monitored are restricted to events that run in the thread that initialized the library and create the Event Set. In order to change this behaviour from the default, we must use what the library defines as **Granularity**.

Granularity has 4 stages, the first one and default is **thread**. The Event Set only records the events that happen when the kernel switches to the specified thread, when the execution changes thread the counters do not increment. An important note is that an Event Set can be attached to an external thread other than the creator in order to do remote monitoring of another thread. The second granularity is **process**, that behaves really similarly to the thread granularity but at a process level. Similarly it can also be attached to another process. The third granularity is **CPU**, as its name implies the counters count the events happening in a single CPU. If the library can attach to another CPU is depending on the chip used. The last granularity is **system**, where the counters aggregate all the different CPU's in the system.

Constant	Granularity
PAPI_GRN_THR	PAPI counters for each individual thread
PAPI_GRN_PROC	PAPI counters for each individual process
PAPI_GRN_PROCG	PAPI counters for each individual process group
PAPI_GRN_SYS	PAPI counters for the current CPU, are you bound?
PAPI_GRN_SYS_CPU	PAPI counters for all CPUs individually

Table 2.1: Granularity constants extracted from the PAPI documentation

Choosing the proper granularity is therefore important for the relevance of the extracted features. Intuition would have us directly discard the first two granularity's as they are bound to a process and the cache side-channel attacks are usually performed in a cross process context. However, the attacking process must performed some cache artifacts in order to induce the victim process to leak its data. Therefore, by monitoring only suspicious processes we could detect the attacks.

This approach however has one main flaw. As stated previously one of the places where these attacks have special impact is in cloud computing. A monitoring of this kind would imply that a user not making a security mistake by executing the attackers software without our monitoring could expose the data of another user. For this reason, the recorded features will be at a CPU level and manually aggregated or directly at a system level.

An additional benefit of this approach is that the monitoring facility can be maintained and run by the cloud provider without the users worrying. This is especially important, as some permissions are required to access the hardware counters.

Domain

Constant	Domain
PAPI_DOM_USER	User context counted
PAPI_DOM_KERNEL	Kernel/OS context counted
PAPI_DOM_OTHER	Exception/transient mode (like user TLB misses)
PAPI_DOM_SUPERVISOR	Supervisor/hypervisor context counted

Table 2.2: Domain constants extracted from the PAPI documentation

Events

Another important decision to make is which events to record. The PAPI library provides a facility to list the events in any particular processor. The events can be listed in several ways native (events defined by the processor) and preset (events available for most processor

that can be derived defined by the library). In our case we will stay to preset events as they are available in most of the processors. The result of listing the events can be seen in Appendix **??**.

As we want the dataset to be as complete as possible the better approach would be to record all events a let up to the user of the dataset to select which fields are important for performance of their particular application. However it is not possible to record all events. As for recording the events we rely in hardware counters, those are not numerous. The amount of counters varies depending on the particular processor but the amount of counters usually is between 4 and 8. More events can however be recorded through multiplexing [5, 6], by worsening the quality of the measurements.

2.3 Attacking

Once the monitoring facilities are in place, we need some attacks to record. One of the issues is that very few actual viruses have been detected. Therefore executing malware in order to record samples is not possible. Furthermore, by using malware we could bias the dataset. The detectors trained with our virus could learn to detect the malicious actions of the virus (the encryption routine for ransomware, backdoor, ...) instead of the vulnerability exploitation we want this detectors to work with.

By looking at the literature we can see that most of it works on the problem use their own implementation of the attack. This implementation is then used to get hardware counter measures. This measure are then used as the dataset to train ML models. We can proceed in a similar way they did, and use an open implementation.

In our case we will use publicly available implementations. This has two main advantages, it first of all allows the final dataset to have a wider variety of implementations. This will make the tools built with this dataset more generic and applicable. The second advantage, is that it allows the project to focus on other areas by saving implementation time.

We could attempt to attack different cryptographic libraries and various programs using cryptographic material. Due to the nature of the attack, we can see that the data inside the stolen memory makes no difference. For this reason, we steal a random string that can be verified of the attacker's side.

Where the attacks can find some variation is in where the stolen information is. For example, to read memory from the same process is very different to reading memory from another VM because for the second attack we would need to brake Kernel Address Space Layout Randomization.

2.3.1 IAIK implementation

The selected implementation for the attack was the one publicly available on Github by the Institute of Applied Information Processing and Communications. In it they provide the implementation of several attack with different memory scopes. From a simple attack that reads memory inside a same process to a kernel address space randomization breaking program.

2.4 CPU affinity

One of the problems with the attack in particular is that it requires both the target and the attacked program to be on the same CPU. All the processors used have more than one CPU therefore we need a way to bind the processes to a particular CPU.

The solution to this problem is to use the taskset program included with GNU/Linux this program allow the user to give to the linux scheduler a particular CPU affinity por a program and the scheduler will have to honor the given affinity.

It is not unreasonable that an attacker would perform such actions as in order to perform the attack it is required that the CPU executing both tasks is the same.

2.5 Implementation

The code implementation of the project starts from the earliest stages as some of the information necessary for design decisions must be obtained from the device. The first module implemented therefore allows the user to list all available preset events in the machine. This action can be performed in user space without any elevated privileges.

2.5.1 Event listing

To perform the event listing we use the following function of the PAPI library. This function will provide both the ability to get the first valid code for an event and get the following elements. To obtain the first element the first argument must be the table we want to look at, it can be the preset table or the native table. In our case we are interested in the preset table therefore we pass the value PAPI_PRESET_MASK. To obtain the first element we just pass PAPI_ENUM_FIRST to the second parameter.

To iterate over the elements we pass the current event as the first argument and pass PAPI_ENUM_EVENTS to the second argument. This will return the next entry in the table. As the first element obtained was

from the preset table all subsequent elements will pertain to this table.

```
int PAPI_enum_event(int *EventCode,int modifer)
```

This would simply yield a list of integers that gives us no information about what the event represents. To access the details of each event the library provides the following function.

```
int PAPI_get_event_info(int EventCode, PAPI_event_info_t *info)
```

If we take a look at the header providing the event information structure, we can see that it contains the following information. From this fields, we have interest in the symbol. This is a short string representing an abbreviation of the event, most of the literature uses this strings to refer to particular events. For simplicity reason the program user will use this instead of the codes to refer to the events. The next fields of interest are the descriptions that can be used to better understand what the event records.

```
typedef struct event_info {
    unsigned int event_code;
    char symbol[PAPI_HUGE_STR_LEN];
    char short_descr[PAPI_MIN_STR_LEN];
    char long_descr[PAPI_HUGE_STR_LEN];
    int component_index;
    char units[PAPI_MIN_STR_LEN];
    int location;
    int data_type;
    int value_type;
    int timescope;
    int update_type;
    int update_freq;
    /* PRESET SPECIFIC FIELDS FOLLOW */



    unsigned int count;
    unsigned int event_type;
    char derived[PAPI_MIN_STR_LEN];
    char postfix[PAPI_2MAX_STR_LEN];
    unsigned int code[PAPI_MAX_INFO_TERMS];
    char name[PAPI_MAX_INFO_TERMS]
             [PAPI_2MAX_STR_LEN];
    char note[PAPI_HUGE_STR_LEN];
} PAPI_event_info_t;
```

2.5.2 Event recording

The next component that needs to be implemented is the module allowing for recording the events. For this have to make use of another component of the PAPI library. This time we will need to create an event set. This abstractraction groups events in order to later be recorded. An event set can be created with the following function. The events can also be added to the set.

```
int PAPI_create_eventset (int *EventSet)
int PAPI_add_event(int EventSet, int EventCode)
int PAPI_add_events(int EventSet, int *EventCodes, int number)
```

As its typing implies an event set is an opaque data structure not meant for direct manipulation by the user. The structure provides some interfaces that allow us to access the counters. We have a function for starting the counter, a function for reading its state and a function for stoping the counters.

```
int PAPI_start(int EventSet)
int PAPI_read(int EventSet, long_long *values)
int PAPI_accum(int EventSet, long_long *values)
int PAPI_stop(int EventSet, long_long *values)
```

2.5.3 Output

After recording the events the program generates an output in the form of a CSV file. This file can then be parsed and plotted with many standard data processing libraries. An example is presented in Figure 2.3 where the data is read in Python using Pandas and plotted using Matplotlib.

As seen in Figure 2.3 the x axis does not have any meaningful unit, right now an index is used. However, this axis will be required in order to label the dataset. As it represents time, we will use a Unix timestamp in milliseconds to store its value. This way each sample can be identified in time.

2.6 Results

Now that all the necessary tools are developed we only need to execute the created software. A script is created with this purpose. Before running the script the programs need to be compiled. The attacks are added to the project by using git submodules in the project repository. So if we wish to run it we will first need to pull the repository. Then after pulling the repository we enter the project directory and pull the submodules.

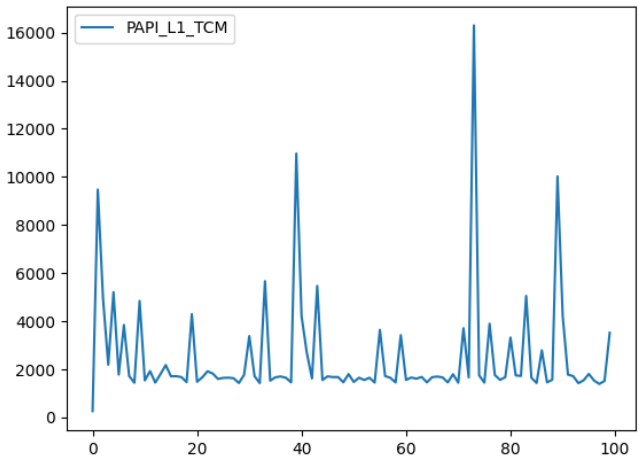


Figure 2.3: Example recording of the PAPI metric PAPI_L1_TCM

```
git pull https://github.com/musergi/tfm.git
cd tfm
git submodule init
git submodule update
```

We then need to build the attacks, the selected project uses a simple make file so we only need to enter the attack project folder and run make.

```
cd deps/meltdown
make
```

The next step in order to generate the dataset is to build the recording software. In our case, as a build system we used cmake which does not come preinstalled with many of the linux distribution it will therefore need installation. After installing cmake building the project is as simple as the following commands.

```
mkdir build
cd build
cmake ..
make
```

After having all programs built we can already run the dataset generation. However, it is recomended to first run the event listing program in order to list all available hardware counters. This can be done by running the built script named "list".

```
./build/src/list
```

Once we have selected the desired events to record we can run the script to record a single attack dataset. For example to record a dataset that records three events the total number of instructions, the L2 cache accesses and the L2 cache misses we run the following command.

```
python scripts/run_tests.py --vuln ./deps/meltdown/test \
    --events PAPI_TOT_INS PAPI_L2_TCA PAPI_L2_TCM
```

The result of running this command will be 2 files in CSV format. First, an intermediary file named "out.csv" that contains the output of the dataset recording software. Then, a file named "out_labeled.csv" contains the dataset itself comprised by the columns representing the event counts, the timestamp column and the label column.

2.6.1 Format

The dataset takes the form of a table with a variable number of columns and a more or less set number of rows. The number of rows is bound by the recording software that records 500 samples spaced more or less 10 milliseconds apart. The number of columns has a minimum of 3 and a maximum only bound by the specific CPU used.

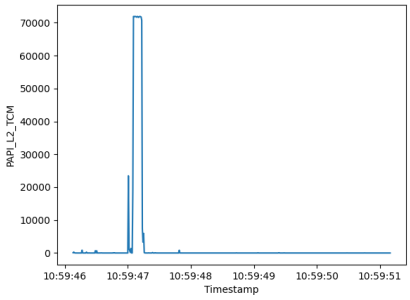
The first mandatory column of the dataset that is the "timestamp" column. This column should not be used when training a model. The main reason we advise against using it is that this column does not contain any information that relates with the attack. This column is however very useful when plotting the dataset as it can be used as the X-axis in the plot. This column data is the UNIX timestamp in milliseconds.

The second mandatory column is the column containing the label. This column is binary in nature as it contains a 0 if no attack was present and 1 if the attack program was running. This column is created by using the previously described column. When the attack is launched the starting timestamp is recorded, after that we wait for the attack to end and get the end timestamp. With the start and end timestamps we can then label a record in the dataset as 1 if its timestamp is between the two.

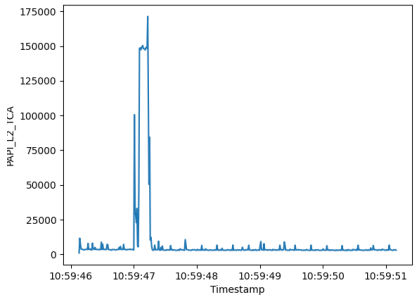
The rest of the columns are comprised of the recorded events. They contains the number of times a particular event took place since the last timestamp. For the dataset to be useful it is recommended to use at least two columns. One of them as a baseline metric (e.g. total number of instruction) and a metric that is affected by the attack (e.g. total cache misses).

2.6.2 Plots

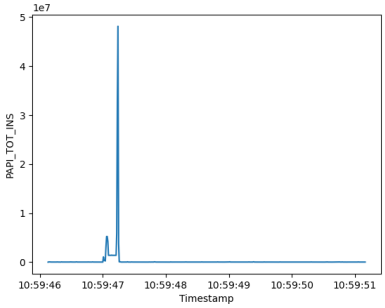
If we plot the first dataset, generated by running the test program with the recording software running in the background we obtain what is shown in Figure 2.4. This plots show all the non timestampt columns against the timestamp column. If we take a look at both Figures 2.4a and 2.4b we can see that they clearly spike when the attack is being performed. This is reasonable as the attack uses the cache as a side channel. In order to use as a baseline we selected the total number of instructions executed, as shown in Figure 2.4c its value remains stable during the attack except for a spike at the end.



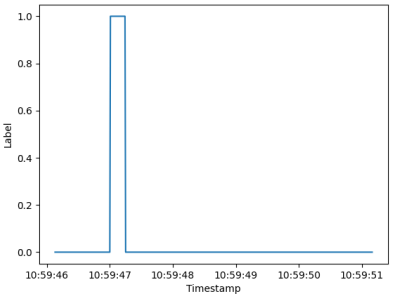
(a) Total L2 Cache Misses



(b) Total L2 Cache Accesses



(c) Total executed instructions



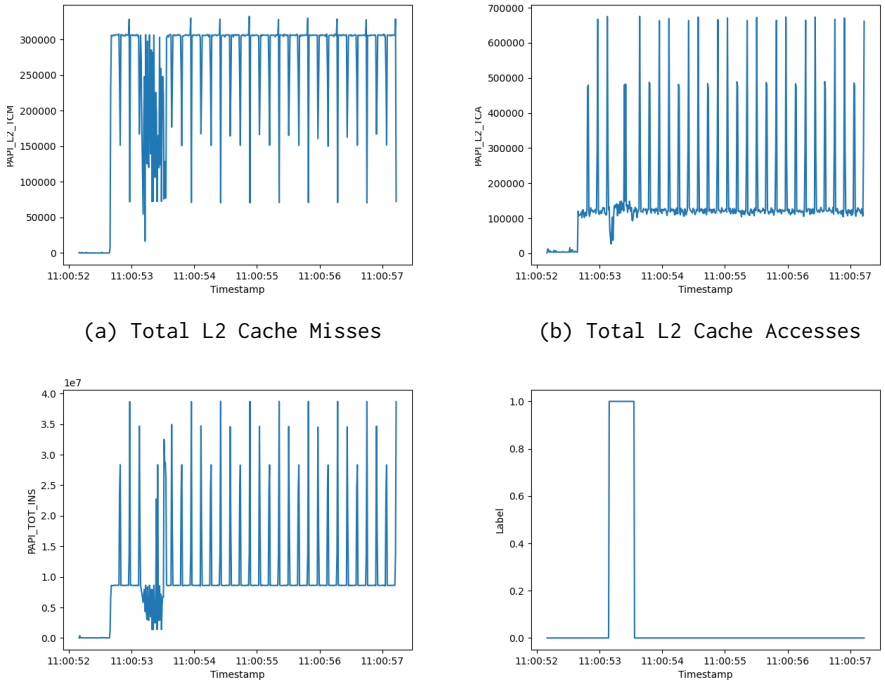
(d) Label

Figure 2.4: Columns of the generated dataset for the test attack

The previous data however was generated under ideal conditions. In that case the detection is trivial as a simple trigger algorithm would detect it. In order, to clearly expose the limits of this approach we performed the same exact test with on key difference. This time a memory stress program was also running during the attack. The generated dataset in this context can be seen in Figure 2.5.

By taking a closer look at Figure 2.5d, we can see that the stress

software does not only take an effect on the metrics obtained but also slows down the attack. In the Figure this can be see by the time where the label has value 1 being larger. Even under stress conditions the attack can be clearly detected in Figure 2.5a. This time however automated detection is more difficult as a trigger algorithm would fail. We speculate that a more sophisticated detection approach like Deep Learning could succeed.



(a) Total L2 Cache Misses

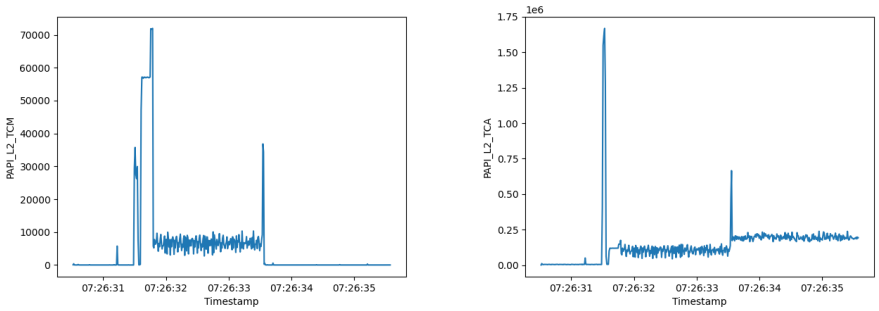
(b) Total L2 Cache Accesses

(c) Total executed instructions

(d) Label

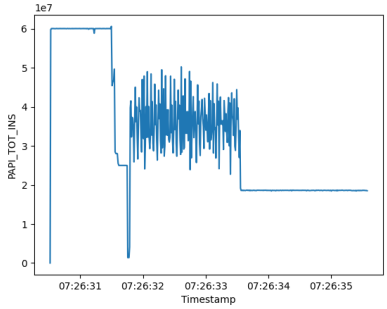
Figure 2.5: Columns of the generated dataset for the test attack under stress

In order to further provide proof that the methodology works we provide a dataset with a more covert version of the attack. This attack slowly steals physical memory. In Figure 2.6 we can see the result of such dataset generation. In Figure 2.6a, we can see that the footprint of the attack in the amount of cache misses is smaller making detection harder. However, it still seams to have a particular patter that would allow detection. With this stealthier attack detection with only the cache accesses (Figure 2.6b and total instruction (Figure 2.6c is unfeasible.

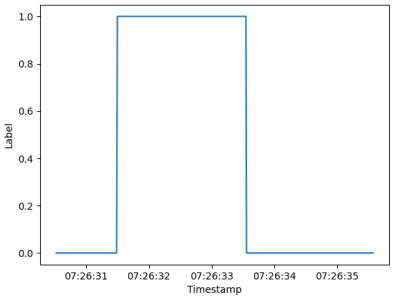


(a) Total L2 Cache Misses

(b) Total L2 Cache Accesses



(c) Total executed instructions



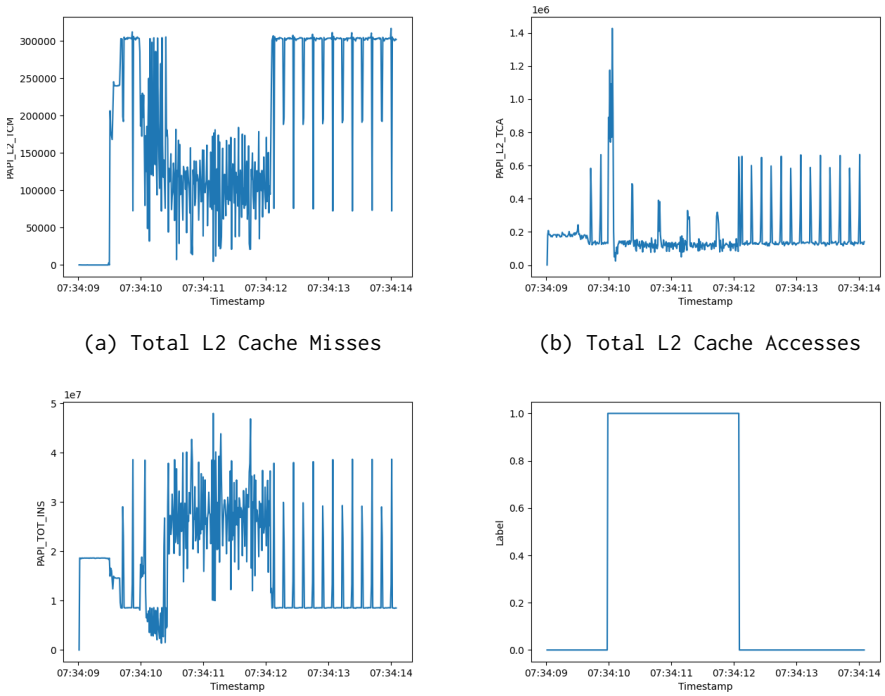
(d) Label

Figure 2.6: Columns of the generated dataset for the memory dump attack

Once again, we are interested on how a considerable load on the system, spetially the memory would affect the attack and the generated dataset. For this reason we follow the same exact procedure as previously. We add a stress process to the execution.

The resulting dataset of this last approach can be seen in Figure 2.7. In this case no comment can be made about the duration of the attack as the attack would dump the entire memory cyclicly if allowed two. The attacking process is killed in order to fit in the capturing window.

This time we can see that in the amount of cache misses of the attack is overshadowed by the stress process in Figure 2.7a. However even if the attack a much lower footprint we can see that the same pattern seen in previous recordings still stands, as the attack is performed in much of the same way. For this reason, once again it is reasonable to speculate that a top of the line Neural Network would have not much issue detecting an attack.



(a) Total L2 Cache Misses

(b) Total L2 Cache Accesses

(c) Total executed instructions

(d) Label

Figure 2.7: Columns of the generated dataset for the memory dump attack under stress

Chapter 3

Business

3.1 Cost analysis

Item	Quantity	Unit cost	Cost
Research	60h	12.5 eur/h	750 eur
Programming	180h	12.5 eur/h	2250 eur
Experimentation	120h	12.5 eur/h	1500 eur
Reporting	120h	12.5 eur/h	1500 eur
Hardware	5 units	800 eur/unit	4000 eur
Total			10000 eur

Table 3.1: Project costs

As expected from a Master's project there where no available resources to cover this costs. For this reason, the costs where taken care of with alternative methodology.

The labor costs where taken care by the project developers without the cost specified in Table 3.1. For this reason, the project could be developed without the need of paying the salary.

The hardware was obtained by the project coordinator from older unused hardware found at the university. It was a challenging task as most of the hardware is in use. The added difficulty of the task was that the hardware needed to be vulnerable to the exploit. The main issue with this requirement is that it forced the hardware to be reasonable new and therefore harder to get.

3.2 Sustainability analysis

A sustainability analysis is an essential part of every project as it allows us to asses and give precise quantification of the impact on nature, society and economy of any project. With software project this analysis is often skip as it is though that by not having a physical component the project does not have much impact. This is a narrow mindset, at the core of the sustainability revolution is software. For this reason we think that it is essential to perform for any project this analysis.

3.2.1 Environmental

One of the main motivations for this project is the environmental impact that the current mitigation provide over the performance. We previously stated that this impact can be as high as 20%. This impact in performance is tightly coupled with the environmental cost of computing. By cutting performance by 20% we require more assets in order to cover this performance loss therefore causing an increase in the hardware requirements and energy requirements. Increasing hardware has a very high cost on the environment as the silicon industry has a very high energetic cost.

Our project aims to up the efficiency of secure systems, and by consequence, reducing the environmental footprint of computing centers with vulnerable hardware. It also allows for the vulnerable hardware, to not be discarded directly which will have an gigantic negative impact on the environment.

3.2.2 Social

The social impact is twofold, in a big scale it allows companies to run in cloud providers, in a smaller scale it allows end users to safely use their vulnerable hardware.

On the one hand, most of the services developed by companies are running in servers hosted by cloud providers as they allow easier and seamless scaling. Many of this services are basic for our everyday lives (e.g. online banking). Some allow us to more efficiently sustain our most basic needs (e.g. social networks allow us to fulfil our need to socialize). I believe that by making this industries more profitable by reducing costs our project will positively impact society.

On the other hand, our project allows users that have vulnerable hardware at home to keep using their hardware without being exposed to the attacks. As for most users replacing the hardware is not an option, having a security patch with as low performance hit as possible will allow to safely continue to use their hardware for longer.

3.2.3 Economic

It is a common saying in the cybersecurity space that security is not investment, it is rather a cost for companies as it does not generate value. However, in the case of cloud providers patching the studied vulnerability is a must if they do not want to exclude a part of their potential clients. For this reason, an alternative solution to the currently applied one with higher efficiency would highly benefit the cloud providers.

Chapter 4

Conclusion

From the project results presented, we can conclude that the project goals have been completed. We have built a solid framework that enables the generation of a generic dataset of cache side-channel attacks.

To provide a proof of concept of how such a dataset is generated, we provided four examples of datasets generated using our tools. Not only that but we introduced a methodology on how to relevantly stress the system during the recording in order to provide the worst case conditions.

The tools developed have a very important role often overlooked in the field of machine learning, they provide a common ground to test the different proposed techniques. Without a common dataset the results provided on a paper, even if reproducible can not be compared to the ones obtained by another paper.

For this reason, this project is a necessary step that will allow to further the research in the field of detection of cache side-channel attacks through hardware counters to ensure a minimal (if any) performance hit for users.

A proof that attacks like Meltdown and Spectre will not soon disappear is that the new Apple M1 still suffers from a similar attack.

All the code for the generated tools alongside with the necessary instructions to use them are publicly available at:

https://github.com/musergi/tfm

Bibliography

[1] Mark D. Hill, Jon Masters, Parthasarathy Ranganathan, Paul Turner, and John L. Hennessy. On the spectre and meltdown processor security vulnerabilities. IEEE Micro, 39(2):9–19, 2019.

[2] Intel performance hit. https://www.extremetech.com/computing/291649-intel-performance-amd-spectre-meltdown-mds-patches. Accessed: 2022-02-15.

[3] Maria Mushtaq, Ayaz Akram, Muhammad Khurram Bhatti, Maham Chaudhry, Muneeb Yousaf, Umer Farooq, Vianney Lapotre, and Guy Gogniat. Machine learning for security: The case of side-channel attack detection at run-time. In 2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS), pages 485–488. IEEE, 2018.

[4] Jonas Depoix and Philipp Altmeyer. Detecting spectre attacks by identifying cache side-channel attacks using machine learning. Advanced Microkernel Operating Systems, 75, 2018.

[5] Philip J Mucci, Shirley Browne, Christine Deane, and George Ho. Papi: A portable interface to hardware performance counters. In Proceedings of the department of defense HPCMP users group conference, volume 710. Citeseer, 1999.

[6] John M May. Mpx: Software for multiplexing hardware performance counters in multithreaded programs. In Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001, pages 8–pp. IEEE, 2001.