UNIVERSITY OF BIRMINGHAM

Investigating Information Leakages Observed Through the Power Consumption Side Channel on the Morello Board

Andreea-Bianca Fraunhoffer
Student ID: 2038849
Supervisor: David Oswald
Word count: 7203
60-credit report submitted for the degree MEng Computer Science/ Software Engineering with Industrial Year

April 7, 2024

Contents

1 Abstract **2**

2 Introduction **4**
2.1 Brief Overview 4
2.2 Aims and Objectives 5

3 Literature Review and Background **8**

4 Methodology **12**
4.1 Stages of Experiment 13
4.1.1 Target Code Development 13
4.1.2 Compatible Compilation 14
4.1.3 Initial Power Trace Analysis 15
4.1.4 Further Power Trace Analysis 17
4.2 Reproducible Final Experiment 18

5 Results and Evaluation **21**
5.1 Results and Discussion 21
5.2 Future Work 25
5.3 Challenges Encountered 26
5.3.1 Scope Changes 26
5.3.2 Equipment Issues 27
5.3.3 Compilation Tooling Issues 28
5.3.4 Time Constraints 28

6 Conclusion **29**

References **31**

Chapter 1

Abstract

The Instruction Set Architecture (ISA) of a machine represents the interface found at the intersection between software and hardware that completes the abstract model of the processor by describing the functionalities that are supported. The assembly code instructions, registers, primitive data types and the organisation of memory are encompassed in an ISA, together with any extensions that provide additional features.

Such an extension is CHERI (Capability Hardware Enhanced RISC Instructions), a novel architecture extension created by the University of Cambridge that enables a fine compartmentalisation of software through its implementation of capability-based memory addressing. This method of managing computer memory encodes additional information into the direct references to memory known as pointers, information pertaining to the level of privilege associated with any given memory reference. The pointer extended in this manner becomes the capability, a new primitive data type.

CHERI proposes a hardware centered solution for the software-level problem of memory safety. Memory safety vulnerabilities comprise the various undefined behaviours of computers that occur when memory outside of the intended scope is being accessed or altered, and they represent a serious threat against the integrity of contemporary software.

Within this context, Arm's Morello Program has produced the Morello Evaluation Board, a prototype System-on-Chip (SoC) whose processor is implementing a CHERI-aware, experimental version of the Arm v8.2-A ISA. The Morello Porgram is currently undergoing a testing phase, during which Arm's industry and research partners conduct their own studies and investigations into the performance of the Morello Board.

In computer security, a side channel represents an aspect of the implementation of a computer that inadvertently leaks information that would otherwise be inaccessible. Side channels can be of the hardware variety, such as power consumption, electromagnetic emanations, sound, temperature, or of the software

variety, such as cache timing and speculative execution.

This project monitors the power consumption side channel of the Morello Board and analyses the leakages discovered, namely the appearance of execution-dependent peaks sufficiently distinct in shape to potentially allow the identification of the instructions executed at a specific point in time.

Chapter 2

Introduction

2.1 Brief Overview

Memory safety issues represent the undefined behaviour of computer systems that occurs when a user, malicious or not, has the ability to read or write to locations in memory that should normally be inaccessible. They often lead to serious consequences such as arbitrary code execution, particularly when exploited through attacks like the buffer overflow. Buffer overflows arise when an attacker takes advantage of an insecure buffer by providing it with an input that is large enough to overwrite the parent process' return pointer.

The control allocated to the programmer by languages with a low level of memory abstractions like C or C++ facilitates the spread of memory safety issues by allowing direct references to memory under the shape of pointer arithmetics (the manipulation of the memory address encoded in the pointer through arithmetic operations) and dynamic memory allocations (the manual management of heap memory allocations that is executed through C functions like *malloc()* or *free()*); regardless, C and C++ are among the most popular programming languages worldwide [11], which suggests that vulnerabilities relating to memory safety could affect a large part of modern software[16].

In 2014, the University of Cambridge together with SRI International have begun a research project titled Capability Hardware Enhanced RISC[1] Instructions, also known as CHERI [13, 17]. CHERI extends the processor model represented by conventional Instruction Set Architectures (ISAs) with support for the capability, a new data type playing a key role in addressing memory.

An ISA comprises the abstract interface that depicts the supported functionality of the processor hardware and its implementation in software. The extension proposed by CHERI seeks to mitigate memory safety issues at the hardware level by allowing a finer compartmentalisation of software through capabilities. Thus, within the CHERI ISA, pointers become capabilities, a data

[1] Reduced Instruction Set Computer

type that encodes within itself the reference to the location in memory previously stored in the pointer, together with additional metadata. The metadata contains information related to the privilege of the capability, such as the permissions granted to the capability or the bounds in memory the capability is allowed to access. By tightly constraining the authority of references to memory locations, the severity of memory breaches is greatly reduced, together with the possibility of attacks like buffer overflows [13, 17].

Arm launched the Morello Program in 2019 in partnership with the researchers behind CHERI and built a prototype System-on-Chip (SoC) that implements the CHERI architecture extension on one of their proprietary processors [15]. Copies of the Morello Evaluation Board, or the Morello Board for short, were shipped to the wide ecosystem of Arm research and industry partners in 2022, as part of the program's testing phase, allowing these partners to conduct their own investigations into the applicability of a system relying on capability-based addressing.

In view of the fact that the Morello Board is a physical hardware implementation of a state-of-the-art mitigation against memory vulnerabilities, determining the extent to which it protects against other types of security threats could represent a worthwhile study. In particular, observing the effects of physical implementation attacks such as side channel analysis attacks begets interest, on account of the hardware-level innovations present on the Board.

Side channel analysis attacks leverage information pertaining to either the hardware or the software implementation of a computer in order to obtain data that would be inaccessible by other means. By monitoring the fluctuations of characteristics such as the power consumption or the electromagnetic emanations of a processor or by recording the elapsed time of program executions or cache fetches (among other things), an attacker can gain insights into values used by the executing software, bypassing the cryptographic measures enforced on the machine (which themselves are often the target of these attacks, as side channel leakages during cryptographic operations have been demonstrated to provide a good starting point for the retrieval of secret keys [6]).

Hardware side channel attacks are considered to be the main focal point of this project because the unique modifications brought to the hardware of the Board could unintendedly prove to disclose information. The power consumption side channel will be monitored during this project in the hopes of observing any tendencies to leak data that the Morello Board might display.

2.2 Aims and Objectives

The main objective of this project is to discover whether the Morello Board leaks information through the power consumption side channel. If so, a secondary objective becomes assessing the severity of the leak by examining what data is leaked, how it pertains to the software executing on the Board, and

whether it would be useful to an attacker looking to exploit this particular side channel.

An experiment is proposed as means of achieving the goals stated above. The experiment entails the monitoring of the power consumption of the Morello Board as it runs capability-aware C code, followed by the further analysis and interpretation of the resulting power traces. Methodologically, the experiment is comprised of four steps:

- The development of the piece of software to be used as the target code;
- The compilation of the target code as a CHERI compatible binary file that will be executed on the Board;
- The initial analysis of the power trace captured during the execution of the target binary with the help of a power measurement device, such as an oscilloscope;
- The further interpretation of the power traces, using processing scripts.

Several outcomes have been predicted to follow from the results of this experiment:

- An unlikely conclusion would be that the Morello Board does not leak any information at all through the power consumption side channel. This is improbable because of the fundamental assumptions about the functionality of modern CPUs that confer side channel analysis attacks their efficiency, as well as because of the fact that side channels were not considered during the assembly of the Morello Board's threat model [19].
- Moreover, it is also rather unlikely that the Morello Board discloses a great amount of information through its power consumption. This is due to the current awareness of the threat represented by the wider spectrum of side channel attacks, which led to general mitigations (such as favouring constant execution paths) being implemented, which can inadvertently address the vulnerabilities used as the points of entry for power analysis attacks.
- The most probable outcome is defined to be that the Morello Board does leak data through the power consumption side channel, although an amount that could only be useful to a skilled attacker in a very particular setting.

Over the course of the project, the experiment revealed the appearance of instruction-dependent peaks on the power traces recorded during the execution of the target code, peaks which seemed to correspond to the assumptions made about the power consumption of each stage of the target code. This finding denotes a potential vulnerability that could allow an attacker to identify the instructions executed by a program by observing the power trace of the

respective execution. The vulnerability would affect software with non-constant execution paths, which could prove to be sufficient to uncover the values of the used operands.

Chapter 3

Literature Review and Background

One of the earliest examples of an attack that exploits the vulnerabilities of computer memory using a buffer overflow dates back to 1988 - the Morris worm [4], which relied on a buffer overflow inside of a network protocol. Soon after, in 1996, the seminal work 'Smashing the Stack for Fun and Profit'[5] exposed the pervasiveness of this vulnerability: at the root of the problem lay the undefined behaviour of some of the most widely-used string processing functions provided by C, such as *strcpy()* and *gets()*. The devastating results of allowing memory to be overwritten painted a bleak picture of the security of the existing software.

In this context, software and hardware mitigations against buffer overflows soon began to appear: Address Space Layout Randomisation randomly rearranges the memory locations of the components of a process with the aim of attempting to stop the intentional misdirection of code execution [7], stack canaries raise segmentation fault errors if it is detected that they were overwritten [8], the NX (no-execute) bit creates a hardware delimitation between locations in memory by marking some as non-executable. Safe alternatives to the vulnerable C functions have been circulated and even new, memory safe programming languages such as Rust have been created as part of the effort to address the danger of memory vulnerabilities [14].

Currently, exploiting buffer overflows is not as simple as it used to be, due to the rapid adoption of the aforementioned targeted defenses, as well as due to the impact of tools like Valgrind [9] or the Address Sanitizer [10], which help programmers actively discover and prevent memory vulnerabilities during the software development process. In spite of the widespread improvement of safeguards, memory safety issues are still rampant: a presentation delivered by a member of the Microsoft Security Response Center in 2019 stated that 70% of the security bugs reported across 12 years are memory problems [16].

The University of Cambridge's project, CHERI, aims to introduce an ar-

chitectural solution to the problem of memory safety by improving software compartmentalisation from the hardware layer of a computer system, thus ensuring a firmer containment of any potential memory breach [13, 17]. This is achieved through the use of capabilities, the new, hardware-supported primitive data type that replaces the memory references known as pointers, whose management is known to create vulnerabilities. The CHERI ISA extends conventional ISAs with capability-aware instructions, providing compatibility with both software built for established architectures and new software that would be built for CHERI systems.

In order to describe a 64-bit pointer, the correspondent capability will use 128-bits, and similarly, for 32-bit pointers, the capability will measure 64 bits.

As depicted in Figure 3.1, the contents of a capability consist of:

- The address of the memory reference itself (the previous pointer);
- The capability bounds, which describe the limited space in memory where the capability is authorised to execute instructions;
- The object type, a value that encodes the type of object the capability is referencing;
- The permissions, which limit the number of instructions the capability is allowed to execute;
- Separately, a one bit validity tag ensures that invalid capabilities cannot perform any instructions.

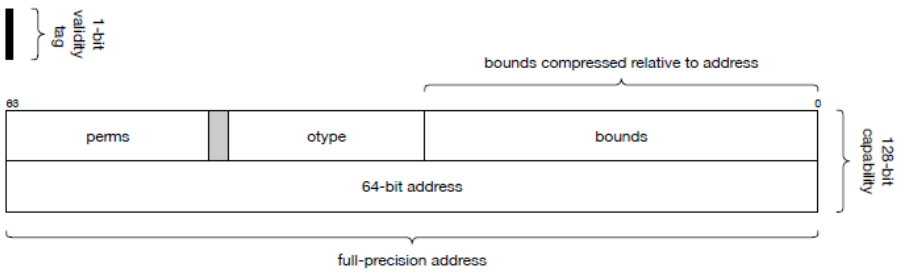

Figure 3.1: The visualisation of a 128-bit capability as per 'An Introduction to CHERI'[17].

CHERI achieves its main goal of software compartmentalisation through several ground assumptions about the functionality of capabilities:

- Provenance validity states that capabilities can only be derived through instructions that came from another valid capability;
- Capability monotonicity states that the privileges (bounds, permissions) of the parent capability cannot be exceeded by the child capability;

- Reachable capability monotonicity guarantees that the set of reachable capabilities cannot increase during arbitrary code execution.

CHERI's fine-grained compartmentalisation of software is reliant on the principles of least privilege and of intentional use. As per the latest version of the CHERI ISA, the principle of least privilege has been implemented through a software design choice that allows each program component to only execute with the lowest level of privilege it needs, whereas the principle of intentional use permits CHERI to avoid 'confused deputy' problems by limiting privileged processes acting on behalf of lesser privileged ones to exclusively use resources the less privileged process could access [20]. Thus, code executed on a CHERI system will have enough privilege to execute what it is intended to, and no more. This collection of rules and generalisations relating to the functionality of capabilities represents the solid base supporting CHERI's strong security assurances.

Arm's Morello Board represents the hardware prototype of a CHERI system, created as part of the Morello Program for the purpose of allowing researchers and industry partners to assess the functionality of an architectural extension featuring capability-based memory addressing [15]. It is based on a standard Arm Neoverse N1 processor implementing the Morello ISA: an experimental version of Arm's proprietary v8.2A ISA that has been extended to provide support for capability-aware instructions. The most significant change to the conventional CPU micro-architecture has been the extended register file which allows the general purpose registers to also hold capabilities [18]. These physical registers have been expanded to 129 bits (the length of a capability derived from a 64-bit pointer, together with its validity tag, as seen in 3.1). Another upgrade to the existing micro-architecture has been to the CPU cache and the system buses, which now carry an extra bit (corresponding to the validity tag) with every 128 bits of data.

The Morello Program, or CHERI behind it, is not the first project to propose a capability-based computer architecture - the concept of a capability was first defined by Dennis and Van Horn in their 1966 paper 'Programming semantics for multiprogrammed computations'[1]. Capability-based addressing has been explored in depth in the late twentieth century, with a number of significant developments occurring mostly between the 1970s and the 1980s [12, 2], notable being the Cambridge-developed CAP computer as a predecessor and a source of inspiration for CHERI[3, 13].

Thus, bearing in mind that passive side channel analysis attacks only became widely known in the late 1990s (Kocher's 'Differential Power Analysis' explicitly described an attack that could break the Advanced Encryption Standard (AES) cryptographic algorithm using information gathered by analysing power traces in 1996 [6]), it could prove to be worthwhile to monitor the leakages of a physical side channel belonging to the Morello Board, a present-day physical system that implements a capability-based addressing architecture, since the periods of active research into both of these topics did not overlap until CHERI.

Additionally, in the 2023 paper 'Arm Morello Programme: Architectural security goals and known limitations' [19], it is stated that side channel analysis attacks were not given further consideration when assembling the hardware threat model. The study depicts the Morello Board's resistance to side channel attacks as being similar to that of the Neoverse N1, a processor which provides limmited protection against this attack vector. Although CHERI focuses on defending software-level vulnerabilities and side channel analysis attacks represent a hardware-level vulnerability, it is also mentioned that contributions investigating the intersection of these topics are welcome.

Chapter 4

Methodology

The aim of the experiments conducted as part of this project was to monitor and interpret the power leakages of the Morello Board. In order to assess whether leakage takes place, a piece of software known as the *target code* (or *attack code*) should be running on the targeted processor. It is expected that information pertaining to the target code, such as the value of operands or whether an instruction was executed or not, would leak. In turn, the processor should be connected to a power measurement device such as an oscilloscope. During the last step, the reading displayed on the oscilloscope as a waveform plotted against voltage and time (known as the *power trace*) would be further analysed, particularly to see whether the expected target code leakage occurred.

Corresponding to the brief outline above, the experimental stage of the project has had four stages:

- Target code development, during which the attack code that would run on the Board was developed;
- Compatible compilation, which tackled compiling the target code as a CHERI-compatible binary and, secondarily, porting the binary file to the board;
- Initial power trace analysis - the superficial analysis of the power trace resulted from the execution of the target binary as displayed on the power measurement device;
- Further power trace analysis - an in-depth analysis of the power trace using data processing scripts, and the development of said scripts.

This will also be the structure used in this report to describe the methodology of the experiments, followed by a description of the final experiment that has yielded the results presented and discussed in *Chapter 5 Results and Evaluation*.

4.1 Stages of Experiment

4.1.1 Target Code Development

The target code refers to the piece of software that is running on the targeted processor during the monitoring of its power consumption. It should include different types of instructions (both power-hungry and power-efficient), such that the resulting power traces capture as wide a variety of behaviours as possible. The target code should reveal significant power consumption differences according to the particular instruction it would execute or the values it would process, potentially to the extent where an attacker could identify what instruction is executed at a certain point in time (which could allow an attacker to break cryptographical protocols that follow different execution paths for different input values[6]) or what value was one of the operands (even the leakage of one bit could lead to the discovery of a key, as seen in the power consumption attacks against AES[6]).

Furthermore, power leakage could be emphasised through contrast. If the power trace resulting from the execution of target code which has high- and low-power instructions alternating in a specific sequence follows said sequence to some extent, the probability of the system being vulnerable due to power leakages increases. The NOP instruction, by definition, does nothing for a clock cycle. It has been used as a contrasting instruction due to its low power consumption, as it would consume an amount of power comparable to that of the system in an idle state.

Finally, considering the scope of the project, the target code should make use of the architectural extensions provided by CHERI. To ensure that the target code will utilise capability-aware instructions in a capability-based memory addressing context, as well as to be able to compile the code as a CHERI-compatible binary file, the attack code must be written in a language targeted by CHERI, and it should include dynamic memory allocations.

Following this brief set of requirements, C was chosen as the attack code implementation language, as it is one of the languages targeted by CHERI precisely because its low level of abstractions permits memory safety issues to occur. C also allows inline assembly blocks, which increases its compatibility with the needs of this experiment, because calling the NOP instruction directly as assembly facilitates its usage.

An initial version of the target code dynamically allocated space for a large *(n=1000000)* array, iterated through it, stored in each array element the value of its index multiplied by an arbitrary number *(m=123)* and then executed one NOP for each entry in the array (as displayed in the pseudocode algorithm 1). When the corresponding assembly code was examined, the majority of instructions were loads (LDR) and stores (STR), followed by additions (ADD). The resulting power trace displayed a sharp, consistent peak (the peak inside the yellow circle in Fig. 4.1).

Algorithm 1 Pseudocode of an early version of the target code

```
1: while true do
2:     int size = 1000000;
3:     malloc() arr[size];
4:     for i = 0, 1, ... size do
5:         arr[i]*=123;
6:         NOP;
```

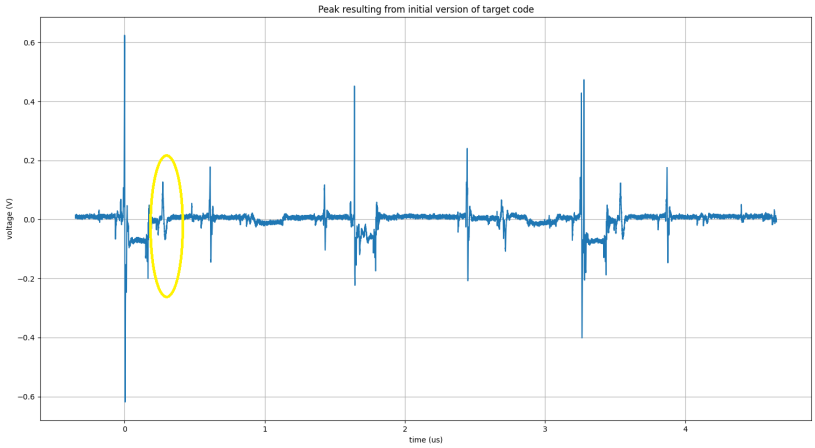

Figure 4.1: Peak visible when running an initial version of the target code

Retrospectively, the trace was constant because of the lack of contrast in the executed instructions' power consumption. The number of NOPs was not large enough to force the processor into an idle state for enough clock cycles for it to 'show' on the trace, possibly because of pipelining. However, the discovery of the execution-dependent sharp peak was promising, leading future iterations of the target code to be built upon this one.

4.1.2 Compatible Compilation

The compilation of the target code as an Arm ISA pure capability executable represented a significant stage of the experimental process, because of the many platform incompatibilities between the personal laptop used for the project and the Morello Board.

Initially, the preferred means of compilation was the Morello Fixed Virtual Platform (FVP), a virtual emulation of the Morello CPU model running inside of a Docker container. Mounting a shared folder enabled communication between the two machines: locally stored C target code files were compiled as Arm

compatible binary files on the FVP before being transferred to the board via the Secure Copy Protocol (SCP) over SSH. It soon proved to be unsustainable due to performance reasons, as the FVP took upwards of 10 minutes to boot, and most of the executed commands ran very slowly.

A key step towards obtaining the principal results of this experiment was beginning to compile files as pure capability with the purpose of fully making use of the capability-based memory addressing provided by the Morello Board. The FVP contained a copy of the Morello Software Development Kit (SDK), which, although functional, was slow enough to warrant the need to find another method of compiling the attack code.

The next approach was to use the Cheribuild toolchain. Cheribuild is a Python-based tool that facilitates building CHERI-compatible software for different target environments, including Morello. Cheribuild is based on CheriBSD (a version of the FreeBSD operating system that provides support for capabilities), whereas the board was booted to run a Debian Linux distribution, leading to a compilation incompatibility that did not permit the binary files to run.

The latest solution to this issue has been to transfer the target code to the Morello Board through SCP and compile it there. This is a viable option because the Board software includes the same SDK found on the FVP. Further detail regarding the different problems encountered during this step of the experiment can be found in *Section 5.3.3 Compilation Tooling Issues*.

Finally, the process of communicating and interacting with the Board also went through several stages across the experiment. The Morello Board was connected to the personal machine through USB, as well as through a network connection to allow SSH. For the largest part of the experiment, the network connection was represented by the Security and Hardware laboratory's wireless network. Unfortunately, the network proved to be unstable later during the course of the project. For quick access to a root shell, communicating through the USB port using a serial port communication program such as Minicom was sufficient, but it delayed the process of transferring files back and forth between the local environment of the personal machine and the remote one of the Board. Eventually, the wireless connection was replaced by a static Ethernet connection which mostly functioned without issues.

4.1.3 Initial Power Trace Analysis

Monitoring the power consumption side channel of a given integrated circuit typically requires particular tools to be used: a power measurement device (for example, an oscilloscope) and a probe. The probe will 'listen' to the power consumption over time of a given location of the circuit (a location that is susceptible to power leakages), which will then be displayed on the oscilloscope as a waveform. The power consumption is a result of the digital logic implemented at the circuit level: setting bits to 1 is equivalent to 'high' from a digital electronics perspective, which means that the respective transistors composing a bit will

require roughly 5V of current to change their state from 0 ('low', corresponding to approximately 0V).

An interesting peak, in the context of this experiment, would be one that appears, or whose shape is altered, only during the execution of the target code. The location chosen for the placement of the probe was in close proximity to the power supply (Figure 4.2). Different places across the board have been taken into consideration, but, upon a quick examination, the resulting traces did not seem to display any interesting peaks.

Throughout the experiment two devices have been used to capture power traces: a regular oscilloscope, and later a PC oscilloscope. The first oscilloscope used in the experiment was a Rigol DS1074Z with the trigger value set to -98 mV and a sample rate of 1.25 G/s. The execution-dependent peaks have been observed at 500 ns/div and 50 mV/div.

The Rigol oscilloscope functioned normally for the most part, save for when the captured power trace data needed to be transferred to a different machine. Sending the trace data to a laptop for further processing has proven to be problematic because of an issue with the Rigol drivers. The oscilloscope provided a USBTMC port for the purpose of data transfer, as well as device drivers to be installed on the receiving end of the USBTMC connection; however, there was no vendor-approved driver for a Linux system, which is the operating system of the personal laptop that has been used in the experiment.

A different approach involved manually simulating the functionality of an oscilloscope device driver with a Python script. The oscilloscope recognised the connection, although, when analysed, the recorded values of the power trace data did not match those displayed on the oscilloscope. Some time was spent on trying to solve this problem, but a solution was not found in a timely manner (further details about the issues encountered with the power measurement equipment can be found in *5.3.2 Equipment Issues*). Because of this, following input from my supervisor, I have switched to a different power measurement device, the Personal Computer (PC) oscilloscope.

The PC oscilloscope was a PicoScope 6403E with the trigger set to 600mV and the same sample rate of 1.25 G/s. The significantly higher trigger value could be attributed to the increased precision of the PicoScope.

The biggest improvement to the experimental setup brought by the PicoScope has been the ease with which the power trace data could be captured and saved. Since the PicoScope is a PC oscilloscope, it was meant to be used alongside a computer: all that was required was a male-to-male USB cable to connect the device to a machine, and a readily available device driver to be installed on the given machine. This connection had to be established before the PicoScope was used to record traces, because it doesn't have a standalone display for the data and it relies on the device driver graphical user interface to show the power trace waveform on the connected computer's screen.

Furthermore, once a suitable trace has been isolated, saving its data only requires a click on the 'Save' button and to choose a preferred file format from a list (the options were CSV, PNG and Psdata, the PicoScope proprietary format). Within the context of this experiment, a suitable trace is a power trace captured over a long enough period of time such that all of the different shapes of the execution-dependent peak appear at least once, and the trace data has been saved as a CSV file.

4.1.4 Further Power Trace Analysis

The final step of the experimental stage consists of processing the CSV file containing the power trace data with the help of a script in order to facilitate interpretation.

Initially, when the Rigol oscilloscope was still a part of the experimental setup, my supervisor had offered a set of Python scripts that were previously of use in similar projects. These provided a workaround for one of the issues surrounding data acquisition, as they implemented a simple device driver by reading the power trace data as bytes coming directly from the oscilloscope before converting it into a list of numbers that could be used to generate a Matplotlib plot. The device driver was discarded after switching to using the PicoScope as the means of power trace measurement, but the idea of using Python together with Matplotlib was taken further.

Given the power trace data's CSV format, a library that efficiently handles reading from CSV files becomes a requirement for the plotting script. Both Python's *csv* module and the Pandas data analysis library were considered, but Pandas was ultimately chosen because of the degree to which the DataFrame object interface facilitated the development process, especially since the trace data was captured as a very large file, and Pandas is commonly used to process large amounts of data.

Multiple representations of the power trace data help interpret the data from different perspectives; because of this, several plotting scripts have been created over the course of the experiment, each requiring different trace inputs and producing different plot outputs:

- *plot_from_csv.py* produces four views of the same large power trace, approximately centered on the different shapes taken by the interesting peak as the target code executes;
- *plot_peaks_compute_frequency.py* takes as input a smaller trace, marks the first peaks of the repeating sequence of the trace and computes the sequence's frequency;
- *plot_together.py* utilises separate traces taken during the different stages of the target code, and one while the Morello Board is idle, and plots the traces together, thus highlighting the differences in the shapes of the interesting peak.

4.2 Reproducible Final Experiment

This section aims to describe the steps taken during the experiment that has yielded the results presented in *5.1 Results and Discussion*, to be potentially used as a reference for future research (see *5.2 Future Work*).

The target code used, *mul_nop_loop.c*, has three stages of execution which become visible on the captured power traces as execution-dependent peaks. It operates in a loop: initialising the elements of a very large array, multiplying them, and then executing an even larger number of NOPs (see algorithm 2).

Algorithm 2 Pseudocode for *mul_nop_loop.c*

```
1:  while true do
2:      int size = 12345678901234;
3:      malloc() arr[size];
4:      for i = 0, 1, ... size do
5:          initialise with some value;
6:      for i = 0, 1, ... size do
7:          perform many multiplications;
8:      for i = 0, 1, ... size * size do
9:          NOP;
10:     free() arr;
```

In order to obtain a pure capability executable, the target code file is transferred to the Morello Board through the Secure Copy Protocol (SCP), where it will be compiled using the sourced copy of the Morello Software Development Kit (SDK) available locally.

A probe connected to an active PicoScope is placed on the Morello Board at the location visible in Figure 4.2. The settings of the PicoScope software are:

- Trigger value set to 600 mV;
- Vertical axis measuring between -1 and 1 V;
- Horizontal axis measuring 200 us/div;
- Sample rate of 1.25 G/s.

The pure capability binary file is executed, and the power trace captured as a CSV file. The CSV file is passed as input to the plotting script *plot_from_csv.py*, which will generate four zoomed-in windows of the trace data (Figure 4.3). The views should roughly correspond to each execution stage, although the bottom-right window needs to be adjusted by moving forward on the plot until $x = 32.8$ *us* is roughly in the centre of the x axis.

Alternatively, by capturing smaller traces (500ns/div) of the execution of a single instruction (and of the idle state of the Morello board), a combined plot

can be created using *plot_together.py* (Figure 5.1). Each one of these traces can also be used as input for *plot_peaks_compute_frequency.py* in order to obtain the frequency at which the sequence containing the peaks of interest is occurring (Figure 4.4).

Figure 4.2: Placement of the probe on the board

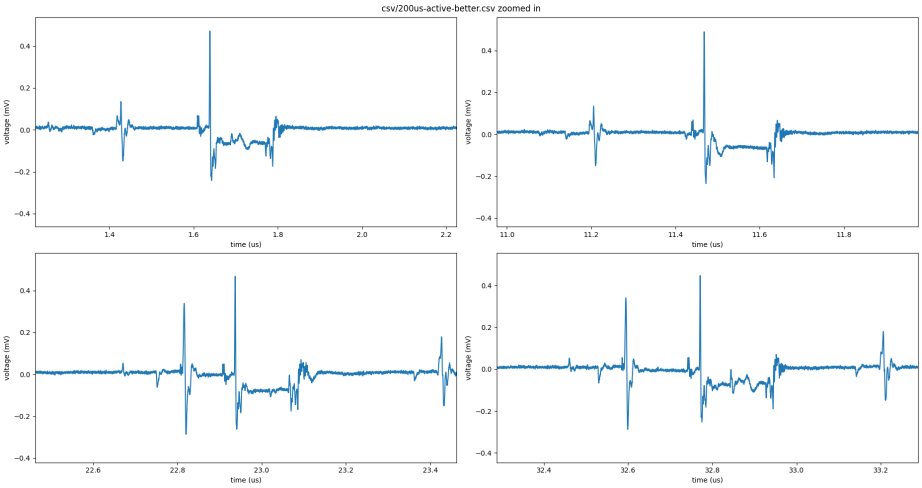

Figure 4.3: Four views of the interesting peak from the same trace: top-left: idle state; top-right: initialisation; bottom-left: in transition between multiplication and NOP; bottom-right: NOP;

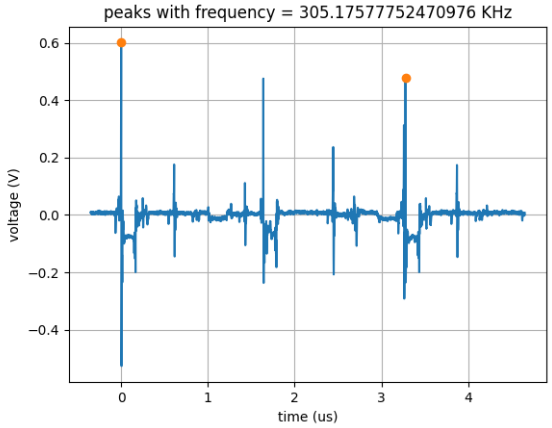

Figure 4.4: The marked repeating peaks of the sequence used to compute a frequency of F = 0.3 MHz

Chapter 5

Results and Evaluation

5.1 Results and Discussion

The results obtained following the experiment are instruction-dependent peaks in the captured power traces, one corresponding to each execution stage of the target code.

Figure 5.1 shows all three shapes of the peak (the array initialisation trace is drawn in orange, the multiplication trace in green and the NOP in red) superimposed on that of the same peak recorded during an idle state (which is drawn in blue), highlighting the difference in appearance between each of them.

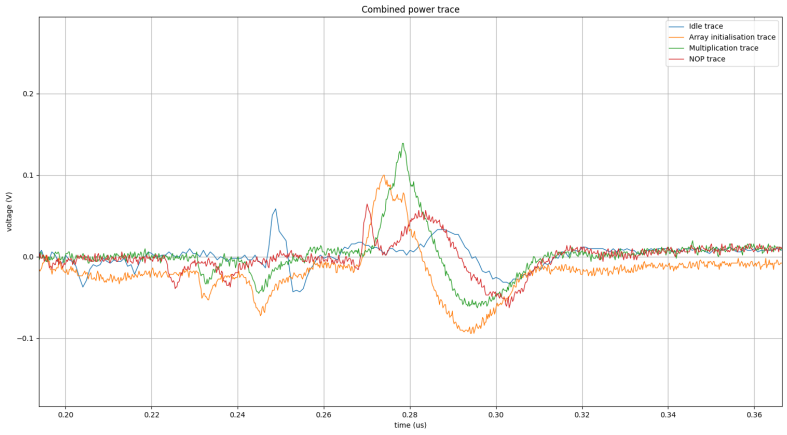

Figure 5.1: Zoomed-in combined power trace

Another view of the results is presented in Figure 4.3, where the four windows

display the different shapes of the peak that have been captured as part of the same power trace.

Figure 5.2 presents a portion of a power trace recorded while the Morello Board was in an idle state, with the peak of interest circled in yellow. Over the course of the experiment, that peak has displayed a regular change in appearance that seems to be closely linked to the instructions executed at a given moment.

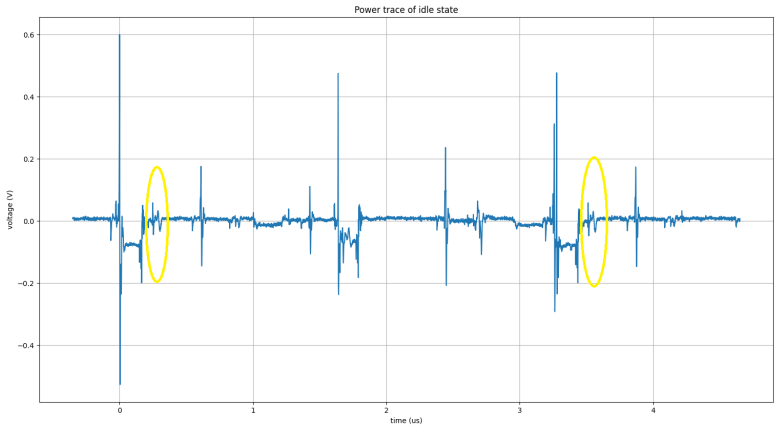

Figure 5.2: Power trace during idle state

The two short peaks of the idle state become a taller, blunt peak when the target code is executing the array initialisation stage, as seen in figure 5.3. This stage is responsible for the execution of a large number of load and store instructions with similarly large operands.

Further, the multiplication execution stage presents a tall, sharp peak (Figure 5.4). This stage is considered to consume the most power, as it also executes many multiplications with large numbers.

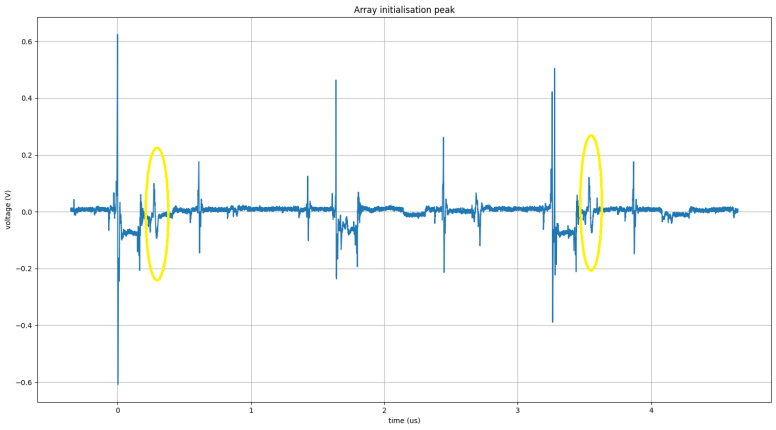

Figure 5.3: Power trace of the Array Initialisation target code stage

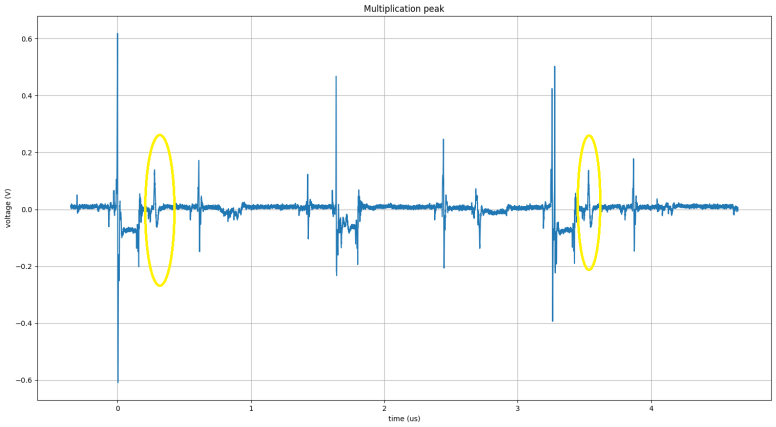

Figure 5.4: Power trace of the Array Multiplication target code stage

The final shape taken by the interesting peak is one somewhat similar to the idle shape: when the target code reaches the NOP stage, which executes a NOP a very large number of times ($n = 1234567891234$ *squared*), the peak becomes short again. Figure 5.5 shows a view of the peak from the same perspective as all the previous plots; the similarity to the idle state peak is more visible when the plot is zoomed in, as is the case for Figure 5.6.

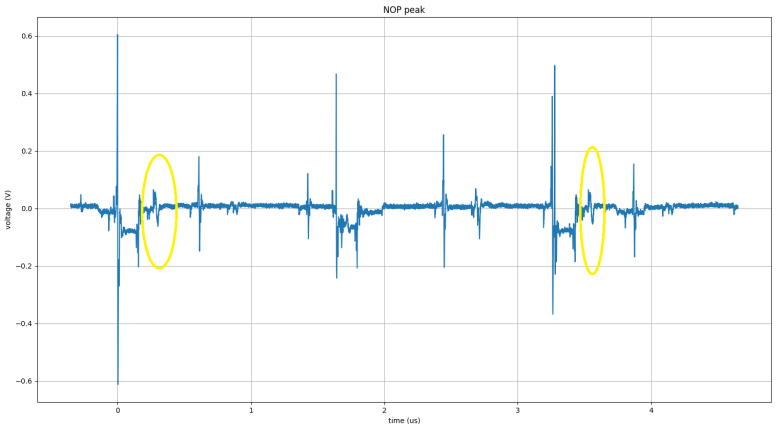

Figure 5.5: Power trace of the NOP target code stage

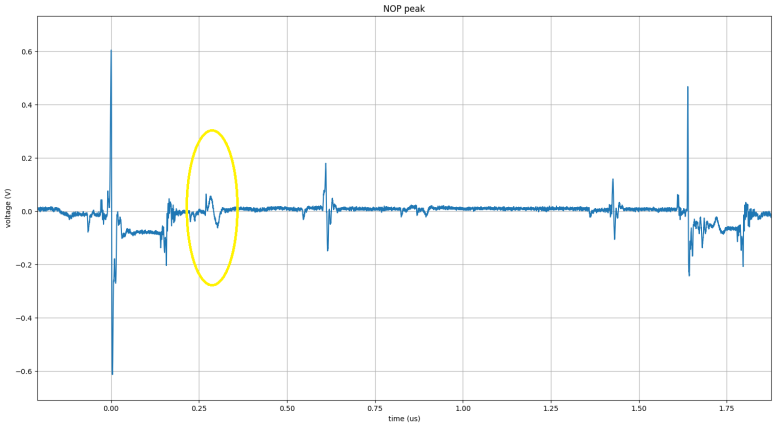

Figure 5.6: Zoomed-in power trace of the NOP target code stage

These findings seem to roughly correspond with the power consumption of the targeted instructions: NOPs should consume little to no power, so it makes sense for the shape of the NOP peak to somewhat resemble the idle state peak. Loads and stores consume more power than a NOP, resulting in the array initialisation peak being taller than the NOP peak, but shorter than the multiplication peak, since they consume less power than multiplications.

The key discovery of the project is that each target code stage produces a

different peak. This hints towards the possibility that, should a more comprehensive attack code be used, the instructions could be identified from 'reading' the power trace which could provide sufficient leeway for an attacker to implement a power consumption side channel attack; particularly, identifying instructions from a power trace represents a vulnerability against attacks targeting cryptographic algorithms that include branches during execution.

An unexpected finding has been the presence of other seemingly execution-dependent peaks elsewhere in the trace (Figure 5.7). These peaks have only been observed during the final stage of further trace analysis. This represents a peculiar discovery, as they are similar in size and shape while appearing at distinctly different moments in time.

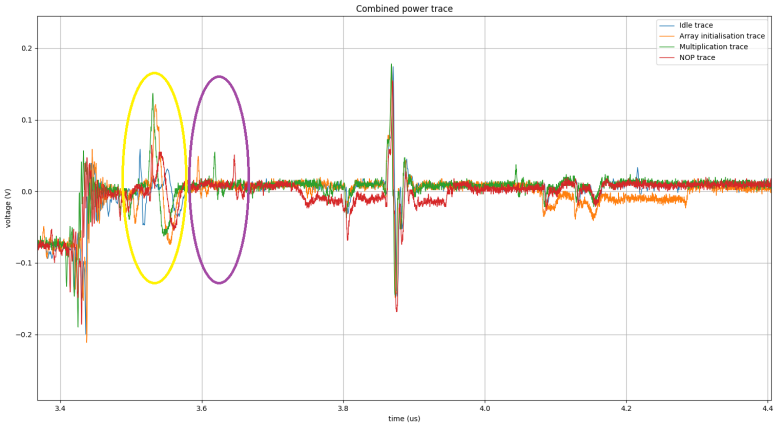

Figure 5.7: Combined power trace zoomed in on unexpected instruction-dependent peaks

Finally, considering the outcomes suggested in the Introduction, the obtained results point towards a potential vulnerability against power consumption side channel attacks owed to a leak of information that could be used to identify instructions.

5.2 Future Work

An interesting future objective could be to directly compare the power leakages between the Morello Board and its conventional architecture counterpart, the Neoverse N1 processor.

The power trace resulting from the execution of same target code, compiled as normal C code, could be compared with the traces captured as part of this

project. If the traces appear to be similar enough, that could be sufficient to denote that the Morello Board is more or less just as vulnerable to power consumption side channel attacks as the Neoverse N1, implying that the hardware support for capability-based addressing does not produce any unexpected information leaks. Conversely, should one of the traces display more execution-dependent irregularities than the other, that could offer a hint towards which architecture is more predisposed to power consumption side channel leakages. The most impacting result would be if the Morello Board would turn out to be less vulnerable to power consumption side channel analysis attacks, as that could potentially present CHERI as one solution that addresses two problems: memory safety, as well as power leakages.

Additionally, another considerable avenue for future research could be attempting to break a naive implementation of a crypto algorithm like RSA, as instruction identification would be sufficient to extract the key during the algorithm's Square-and-Multiply step.

It could also prove worthwhile to further analyse the power leakages of the capability-specific instructions of the CHERI ISA supported by the Morello Board.

Finally, other potential investigations could entail writing the target code in a different programming language, extending the target code with stages corresponding to other instructions, as well as monitoring different side channels (electromagnetic emanations could raise interest for reasons similar to power consumption).

5.3 Challenges Encountered

Over the course of the project certain limitations of various nature and impact have arisen. Each challenge is briefly described below, together with the solution or workaround discovered.

5.3.1 Scope Changes

The proposal for this project presented a larger array of side channels that the Morello Board would be tested against. Primarily, it also included the electromagnetic emanations (EM) side channel alongside power consumption, as well as outlining a plan to actively implement a hardware side channel attack setup against the board for at least four well known attacks: Simple Power Analysis (SPA), Differential Power Analysis (DPA), Simple EM Analysis (SEMA) and Differential EM Analysis (DEMA).

The difference in scope between the proposal and the current project is owed to the significant learning curve that was encountered, particularly in the early stages of the project. The experimental stage has represented the first time in my academic career when I conducted laboratory work; this lack of previous

experience has translated into taking a longer time to get accustomed to the flow of the project. Upon beginning the laboratory tasks in earnest, I understood that the experiment blueprint described in the proposal had been too ambitious, as it had not taken into account that I would need longer to learn the basics and to adapt. My supervisor has been in support of a reduction in scope, as he was also concerned by the difficulty of fulfilling all of the proposal's objectives within the available time frame.

5.3.2 Equipment Issues

As mentioned previously in the Methodology section, one of the most significant setbacks encountered while working on this project has been the issue of obtaining the power trace data from the Rigol oscilloscope.

The oscilloscope had a USBTMC port that was intended for use as a means of transferring data to a computer in tandem with the appropriate device driver. The crux of the problem arose from the limited operating system support of the vendor-approved device driver, as it only supported Windows machines. Although the laptop used as part of the experiment has a dual-boot setup of both Windows and Linux, the experiment required the exclusive use of Linux, and setting everything up again on Windows would have diverted focus away from the next steps for too long.

Another solution could have been to commit to process the power trace data on a different machine, which would run Windows, defaulting to the personal laptop for matters related to the target code. The reason why this avenue has not been pursued is because I only own one laptop, and other ways of getting long-term access to another laptop would have taken far too long.

Eventually, my supervisor and I have agreed on following the manual approach: using the data processing script also as a means of acquiring the data over the USBTMC connection by implementing a device driver. I was provided with the skeleton of such a driver, that had been previously used in similar setups.

The basic functionality of the driver was the ability to send commands to the oscilloscope and to record data. Nevertheless, when the script was run and the power trace data was being recorded, visual cues from the oscilloscope signaled an invalid command input. A closer investigation of the raised error did not reveal any bugs, and it appeared that the data was being read regardless of the error, as the plotting script received a set of data read from the oscilloscope and created a plot. However, the plot did not match the power trace the oscilloscope had captured, and upon further analysis, the data itself was incorrect.

Following these incidents, after analysing the options available, my supervisor and I have decided to switch to monitoring power consumption using a PicoScope because of the intuitive and uncomplicated manner through which a user is permitted to save and export the trace data.

5.3.3 Compilation Tooling Issues

The various components of the CHERI and Morello toolchains have come with many issues over the course of the project. Bearing in mind the recent release of the Morello Program, as well as the fact that the Morello products represent prototypes, and not fully established systems, toolchain difficulties were expected.

Primarily, the Morello FVP, which initially was the first point of contact with the Morello implementation of CHERI, ran very slowly. Considering its nature as a virtual emulation of a different architecture, some slowness was anticipated, but it often negatively affected the flow of the experiments. The FVP was also used as an environment where Arm executables would be ran before being transferred to the board in order to make sure the code superficially executes correctly. The issue was the delayed execution time of the compiled code, which ran slowly enough to make it seem like the FVP was frozen. This gave the impression that the code was faulty, even if it would have executed correctly on the board.

This preliminary execution of the code could not be done anywhere other than the FVP or directly on the board because of incompatible architectures: the machine used as part of the experiment has an x86 architecture, whereas the Morello Board is based on Arm.

Furthermore, other options also turned out to be problematic, as compiling code with the CHERI-aware Clang compiler included in Cheribuild produced pure capability executables that were incompatible with the Morello Board, due to Cheribuild's reliance on the CheriBSD operating system. The Morello Board was running Debian, thus creating another incompatibility.

The solution discussed previously in *4.1.2 Compatible Compilation*, compiling the code directly on the board, proved to be the best way to get fully compatible purecap binary files.

5.3.4 Time Constraints

Two reasons have been the cause of the time constraints of this project: my unfamiliarity with the notions, toolchains and workflows involved, and the time spent trying to find workarounds for the issues mentioned before. This project has been my first exposure to laboratory work, to working with equipment other than a computer (such as the oscilloscope and the PicoScope), to following procedures typical of experiments. Getting accustomed to these activities took longer than was previously expected.

Similarly, the literature review stage also took longer than planned because of the novel nature of the CHERI architecture and because of the recent release of the Morello Board. The body of research work examining various aspects of the Morello Board is slimmer than the one focusing exclusively on CHERI, making it more difficult to research information relating specifically to the board.

Chapter 6

Conclusion

The intention of this project has been to investigate whether the Morello Evaluation Board, a System-on-Chip prototype that implements CHERI leaks information related to code execution through the power consumption side channel. CHERI, or Capability Hardware Enhanced RISC Instructions, represents an architectural extension of conventional ISAs that provides support for capability-based memory addressing, a means of managing memory that increases the degree of software compartmentalisation by limiting the privileges of the memory references known as capabilities. Comprised of the address in memory that is referenced and other relevant metadata that describes the level of authority of said reference, the capability represents a new primitive data type that replaces pointers in the CHERI ISA.

The monitoring of the power consumption side channel of the Morello Board involved the elaboration of a target code to emphasise any potential leakages, the compilation of the code as a pure capability binary file and the analysis of the power trace generated by the execution of the compiled target code. Following this experiment, a leak has been discovered, in the shape of peaks whose appearance is dependent on the instructions executed at a given moment in time. Compared to the power trace obtained as a result of the Morello Board being in an idle state, the peaks present on the active power trace suggest a correlation with the power consumption of the instructions in question: power-hungry instructions produced taller peaks, and low-power instructions lower peaks.

The experiment has only focused on a limited number of instructions - namely loads, stores, multiplications and NOPs - and they seemed to be identifiable on the power traces due to the distinct shapes of the respective execution-dependent peaks. This leakage could prove sufficient to mount a power analysis attack targeting secret values utilised as part of software with a non-constant execution path.

To conclude, this project has met the objective of discovering and investigating an information leakage obtained through the monitoring of the power

consumption side channel of the Morello Board, and the observed results could prove to be a solid stepping stone for further research concerning the side channel analysis of capability-based memory addressing ISAs such as CHERI.

References

[1] Jack B Dennis and Earl C Van Horn. "Programming semantics for multiprogrammed computations". In: *Communications of the ACM* 9.3 (1966), pp. 143–155.

[2] Robert S. Fabry. "Capability-based addressing". In: *Communications of the ACM* 17.7 (1974), pp. 403–412.

[3] Roger M. Needham and R. D. H. Walker. "The Cambridge CAP computer and its protection system". In: *Symposium on Operating Systems Principles*. 1977. URL: https://api.semanticscholar.org/CorpusID:16198472.

[4] Eugene H Spafford. "The Internet worm program: An analysis". In: *ACM SIGCOMM Computer Communication Review* 19.1 (1989), pp. 17–57.

[5] Aleph One. "Smashing the stack for fun and profit". In: *Phrack magazine* 7.49 (1996), pp. 14–16.

[6] Paul Kocher, Joshua Jaffe, Benjamin Jun, et al. "Introduction to differential power analysis and related attacks". In: (1998).

[7] PaX Team. "PaX address space layout randomization (ASLR)". In: *http://pax.grsecurity. net/docs/aslr. txt* (2003).

[8] Mark Dowd, John McDonald, and Justin Schuh. *The art of software security assessment: Identifying and preventing software vulnerabilities*. Pearson Education, 2006.

[9] Nicholas Nethercote and Julian Seward. "Valgrind: a framework for heavyweight dynamic binary instrumentation". In: *ACM Sigplan notices* 42.6 (2007), pp. 89–100.

[10] Konstantin Serebryany et al. "{AddressSanitizer}: A fast address sanity checker". In: *2012 USENIX annual technical conference (USENIX ATC 12)*. 2012, pp. 309–318.

[11] Tegawendé F. Bissyandé et al. "Popularity, Interoperability, and Impact of Programming Languages in 100,000 Open Source Projects". In: *2013 IEEE 37th Annual Computer Software and Applications Conference*. 2013, pp. 303–312. DOI: 10.1109/COMPSAC.2013.55.

[12] Henry M Levy. *Capability-based computer systems*. Digital Press, 2014.

[13] Jonathan Woodruff et al. "The CHERI capability model: Revisiting RISC in an age of risk". In: *ACM SIGARCH Computer Architecture News* 42.3 (2014), pp. 457–468.

[14] Abhiram Balasubramanian et al. "System Programming in Rust: Beyond Safety". In: *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*. HotOS '17. Whistler, BC, Canada: Association for Computing Machinery, 2017, pp. 156–161. ISBN: 9781450350686. DOI: 10.1145/3102980.3103006. URL: https://doi.org/10.1145/3102980.3103006.

[15] Arm. 2019. URL: https://www.arm.com/architecture/cpu/morello.

[16] Matt Miller. *Trends, challenges and strategic shifts in the software vulnerability mitigation landscape*. 2019. URL: https://github.com/microsoft/MSRC-Security-Research/blob/master/presentations/2019_02_BlueHatIL/2019_01%20-%20BlueHatIL%20-%20Trends%2C%20challenge%2C%20and%20shifts%20in%20software%20vulnerability%20mitigation.pdf.

[17] Robert NM Watson et al. *An introduction to CHERI*. Tech. rep. University of Cambridge, Computer Laboratory, 2019.

[18] Richard Grisenthwaite et al. "The Arm Morello Evaluation Platform—Validating CHERI-based Security in a High-Performance System". In: *IEEE Micro* 43.3 (2023), pp. 50–57.

[19] Jessica Clarke et al. Robert N. M. Watson Graeme Barnes. *Arm Morello Programme: Architectural security goals and known limitations*. Tech. rep. University of Cambridge, Computer Laboratory, 2023.

[20] Jonathan Woodruff et al. Robert N. M. Watson Peter G. Neumann. *Capability Hardware Enhanced RISC Instructions: CHERI Instruction Set Architecture (Version 9)*. Tech. rep. University of Cambridge, Computer Laboratory, Sept. 2023.