UNIVERSITATEA LUCIAN BLAGA DIN SIBIU

Doctoral School of Engineering Sciences and Mathematics

PhD field: Computer Science and Information Technology

PHD THESIS

CPU Thermal Signatures for Security Monitoring and Behavioral Analysis

PhD Candidate:
Teodora VASILAS

PhD Supervisor:
Prof. Remus BRAD, PhD

SIBIU 2025

UNIVERSITATEA LUCIAN BLAGA DIN SIBIU

Școala Doctorală de Științe Inginerești și Matematică

Domeniul de doctorat: Calculatoare și Tehnologia Informației

TEZĂ DE DOCTORAT

Semnături Termice ale Procesorului pentru Monitorizarea Securității și Analiza Funcționării

Doctorand:
Teodora VASILAS

Conducător de doctorat:
Prof. Dr. Ing. Remus BRAD

SIBIU 2025

Abstract

This thesis presents a comprehensive analysis of existing research and improvements in numerous aspects of anomaly detection via CPU thermal sensors, addressing important difficulties and novel solutions.

In today's digital age, it is crucial to recognize both the advantages, such as convenient access to information, and the challenges, including security and privacy concerns as well as the spread of misinformation. Recently, organizations across different sectors have embraced computer technology, reshaping their operations through digitization. Information, services, personal data, and goods now exist primarily in digital form. In some countries, public institutions are even prohibited from requiring physical documents to provide their services.

Although the transition to the digital world offers numerous benefits, it also brings significant risks. Cyberattacks occur for various reasons and are driven by diverse motivations. Therefore, understanding system vulnerabilities is crucial to preventing such attacks and strengthening their resilience. Naturally, mitigating these threats requires ongoing efforts, as technology continues to evolve.

This thesis mainly aims at the thermal dissipation in the CPU. It starts with a detailed explanation of attacks and countermeasures in the last decade, covering only software-based attacks that do not require physical access to the system. Attacks and countermeasures were divided into different categories, presented chronologically, together with overview tables and statistical graphs.

Taken together, the trends show a clear trajectory: early emphasis on cache attacks, followed by diversification into speculation, TLB (Translation Look-aside Buffers), prefetcher, power, and thermal vectors. Over time, defenses have shifted away from broad mitigations toward detection, while attackers continue to refine foundational techniques such as Flush+Reload and Prime+Probe. The result is an ongoing race, with Intel platforms, Linux systems, and AES (Advanced Encryption Standard) encryption standing out as the primary battlegrounds.

Next, using Intel digital sensors on the CPU, the *Hot-n-Cold* technique is presented. It can detect anomalies in the temperature of Linux commands. Using the temperature footprint of the command, together with the Pearson correlation, between the original version of the command and a version that was augmented with additional code, it can detect a possible attack.

The key idea behind *Hot-n-Cold* method is that one can detect deviations in thermal behavior that may indicate the presence of injected or malicious syscalls. The experiments focus on two commonly used Linux commands, `ls` and `chmod`, and compare their thermal profiles in normal operation against versions augmented with syscalls extracted from the known CVEs (Common Vulnerabilities and Exposures Database). The results reveal a strong positive correlation of up to 0.95 between the thermal traces of original and modified commands, with a heat difference of up to 1 °C, demonstrating that CPU temperature can be used to characterize and distinguish legitimate executions from potentially compromised ones.

Subsequently, *Beat-the-Heat* is an extension of the above technique, which includes

five different types of noise, to check the resilience of the technique in noisy environments. The same two Linux commands were used in these experiments. The introduction of noise aims to reproduce behavior similar to that of normal usage of a user, such as browsing the internet, pressing keys, listening to music, etc. A major improvement is the noise that introduces significant variability in system behavior, which offers a better evaluation of the robustness of the technique in noisy environments. Despite this added complexity, *BeatTheHeat* method confirms the resilience and viability of thermal-based anomaly detection outside controlled laboratory settings. The experiments were performed on three types of CPUs which show high correlations, approaching 0.96, between the original command and an augmented one.

Next, *WhenThingsHeatUp* technique is another extension of *Hot-n-Cold*, which includes server-like noise and different workloads, to further check the resilience of the original technique in more noisy and real-life environments. The noise introduced aimed to reproduce server workloads by repeatedly loading HTML pages and downloading different sizes of files. Similarly as before, the two Linux commands `ls` and `chmod` are used, and the results include a positive Pearson correlation above 0.9.

Furthermore, a thermal fingerprint is created for cryptographic processes linked to ransomware attacks, together with the fingerprints of authentication processes. In addition, Machine Learning is used to detect commands based on their thermal traces, with an accuracy of up to 90%. Moreover, many other experiments based on heat dissipation are conducted on different kinds of Intel CPUs to better understand some insides of these processors.

The results presented in this thesis demonstrate that thermal monitoring constitutes a feasible and complementary mechanism to improve the security of the system. Monitoring processor temperature data for intrusion and anomaly detection introduces promising directions for future research. I consider these techniques to be the foundation for increasing the security of processors using their heat. In fact, more research is needed to incorporate such approaches into real-time monitoring frameworks and improve data protection.

Abstract

Această teză prezintă o analiză cuprinzătoare a cercetărilor existente și a progreselor realizate în multiple aspecte ale detecției anomaliilor prin intermediul senzorilor termici ai procesorului, abordând dificultăți importante și soluții inovatoare.

În era digitală actuală, este esențial să recunoaștem atât avantajele, precum accesul facil la informație, cât și provocările, incluzând problemele de securitate și confidențialitate, precum și răspândirea informațiilor false. În ultimii ani, organizațiile din diverse sectoare au adoptat pe scară largă tehnologia informatică, remodelându-și activitățile prin digitalizare. Informațiile, serviciile, datele personale și bunurile există acum în principal în formă digitală. În unele țări, instituțiilor publice le este chiar interzis să solicite documente fizice pentru furnizarea serviciilor.

Deși tranziția către lumea digitală oferă numeroase beneficii, aceasta aduce și riscuri semnificative. Atacurile cibernetice apar din motive variate și sunt motivate de factori diferiți. Prin urmare, înțelegerea vulnerabilităților sistemelor este esențială pentru prevenirea acestor atacuri și pentru consolidarea rezilienței lor. În mod firesc, diminuarea acestor amenințări necesită eforturi continue, pe măsură ce tehnologia evoluează.

Această teză vizează în principal disiparea termică în procesor. Ea începe cu o explicație detaliată a atacurilor și contramăsurilor din ultimul deceniu, acoperind doar atacurile bazate pe software care nu necesită acces fizic la sistem. Atacurile și contramăsurile au fost împărțite în categorii diferite, prezentate cronologic, împreună cu tabele de sinteză și grafice statistice.

În ansamblu, evoluțiile evidențiază o traiectorie clară: o concentrare timpurie a atacurilor asupra memoriei cache, urmată de o diversificare către speculație, TLB (Translation Look-aside Buffers), prefetcher, consum de energie și vectori termici. În timp, apărărarea împotriva atacurilor a trecut de la măsuri generale la soluții orientate spre detecție, în timp ce atacatorii continuă să perfecționeze tehnicile fundamentale, precum Flush+Reload și Prime+Probe. Rezultatul este o cursă continuă, în care platformele Intel, sistemele Linux și criptarea AES (Advanced Encryption Standard) se remarcă drept principalele câmpuri de luptă.

În continuare este prezentată tehnica *Hot-n-Cold*, care utilizează senzorii digitali Intel ai procesorului. Aceasta poate detecta anomalii în temperatura generată de comenzile Linux. Folosind amprenta termică a comenzii, împreună cu corelația Pearson dintre versiunea originală a comenzii și o versiune augmentată cu cod adițional, tehnica poate identifica un posibil atac.

Ideea principală a metodei *Hot-n-Cold* este că se pot detecta deviații în comportamentul termic, care ar putea indica prezența unor *apeluri sistem* injectate sau malițioase. Experimentele se concentrează pe două comenzi Linux utilizate frecvent, `ls` și `chmod`, comparând profilele lor termice în condiții normale de funcționare cu versiuni augmentate, conținând *apeluri sistem* extrase din baza de date CVE (Common Vulnerabilities and Exposures). Rezultatele arată o corelație pozitivă mare de până la 0,95 între trasările termice ale comenzilor originale și celor modificate, cu o diferență de temperatură de până la 1 °C, demonstrând că temperatura procesorului poate fi folosită pentru a caracteriza și diferența execuțiile legitime de cele potențial compromise.

Ulterior, *Beat-the-Heat* reprezintă o extensie a tehnicii de mai sus, incluzând cinci tipuri diferite de zgomot, pentru a verifica reziliența metodei în medii zgomotoase. Aceleași două comenzi Linux sunt utilizate în aceste experimente. Introducerea zgomotului are scopul de a reproduce comportamente similare cu utilizarea normală a sistemului de către un utilizator, cum ar fi navigarea pe internet, apăsarea tastelor, ascultarea muzicii etc. O îmbunătățire majoră o reprezintă zgomotul care introduce o variabilitate semnificativă în comportamentul sistemului, oferind o evaluare mai bună a robusteții metodei în medii zgomotoase. În ciuda acestei complexități suplimentare, metoda *Beat-the-Heat* confirmă reziliența și fezabilitatea detecției anomaliilor bazate pe temperatură și în afara condițiilor controlate în laborator. Experimentele sunt efectuate pe trei tipuri de procesoare, care au prezintă corelații ridicate, apropiindu-se de 0,96, între comanda originală și cea augmentată.

În continuare, tehnica *WhenThingsHeatUp* este o altă extensie a *Hot-n-Cold*, care include zgomot specific serverelor și diferite tipuri de sarcini de lucru, pentru a verifica reziliența metodei originale în medii mai zgomotoase și mai apropiate de condiții reale. Zgomotul introdus are scopul de a reproduce sarcini specifice serverelor, prin încărcarea repetată a paginilor HTML și descărcarea fișierelor de diferite dimensiuni. La fel ca înainte, sunt utilizate cele două comenzi Linux `ls` și `chmod`, iar rezultatele includ un indice de corelație Pearson de peste 0,9.

Mai mult, este creata o *amprentă termică* pentru procesele criptografice asociate atacurilor de tip ransomware, împreună cu amprentele proceselor de autentificare. De asemenea, este utilizată tehnica Machine Learning pentru a detecta comenzile pe baza trasărilor lor termice, cu o acuratețe de până la 90%. În plus, sunt efectuate numeroase alte experimente bazate pe disiparea căldurii pe diferite tipuri de procesoare Intel, pentru a înțelege mai bine anumite aspecte ale acestora.

Rezultatele prezentate în această teză demonstrează că monitorizarea termică constituie un mecanism fezabil și complementar pentru îmbunătățirea securității sistemului. Monitorizarea temperaturii procesorului pentru detecția intruziunilor și a anomaliilor deschide direcții promițătoare pentru cercetări viitoare. Consider că aceste tehnici reprezintă fundamentul creșterii securității procesoarelor prin utilizarea căldurii generate. Bineînteles, sunt necesare cercetări suplimentare pentru a integra astfel de abordări în cadrul sistemelor de monitorizare în timp real și pentru a îmbunătăți protecția datelor.

Acknowledgments

First off, if you're reading this, it means I survived. That's already a miracle worth celebrating.

The completion of this thesis was made possible by the many individuals who offered their support, guidance, and expertise throughout the process.

To my supervisor, Prof. Remus Brad, PhD, thank you for your patience, wisdom, and for not giving up on me even when my research plan looked like a conspiracy theory. Your ability to guide me through academic chaos with calm, precision, and steady encouragement made this journey not only possible, but genuinely enjoyable.

To my committee members, Prof. Adrian Florea, PhD, Assoc. Prof. Macarie Breazu, PhD, and Prof. Arpad Gellert, PhD, I appreciate your thoughtful feedback and challenging questions. You kept me on my toes, and occasionally up at night.

To Assoc. Prof. Florina Ciorba, PhD, who welcomed me with open arms during my research mobility and helped launch this entire academic adventure: thank you for your kindness, encouragement, and for believing in my potential from the very beginning. You were the spark that lit the fuse, and I'll always be grateful for your professional and life advice.

To my PhD mate and friend Costin, thank you for the memes, the mutual venting sessions, the never-ending reviews, and the shared existential dread. You made the chaos bearable and the victories sweeter.

To my parents, Rodica and Cornel, thank you for pretending to understand what I was working on and for always asking, "So when will you be done?" with just the right mix of curiosity and concern. Your love and support were the real foundation of this thesis.

To my sister, Iulia, and her husband, Radu, thank you for being my personal cheer squad, occasional therapists, and reliable providers of food. Your support and well-timed distractions (little Emma) were exactly what I needed, especially when I was ready to throw my laptop out the window. I'm lucky to have you both by my side.

To my friends Marius, Bianca, and Adrian, thank you for being the steady rhythm in the background of this long, chaotic symphony. Whether it was a quick message to check in, a shared laugh when I needed it most, your quiet belief in me from afar, or the encouragement that pushed me to grow, your presence made the hard days lighter and the good days even better. I'm deeply grateful for your friendship.

And finally, to all who supported, encouraged, or contributed to this work, thank you. Your involvement has played a meaningful role in reaching this milestone.

List of Tables

List of Figures

Code Listings

List of Equations

Contents

CHAPTER 1

1 Introduction

1.1 Background and Motivation

Over the past decade, organizations have increasingly adopted digital technologies, shifting information, services, and personal data away from physical formats. Although this digital transformation offers great advantages in efficiency and accessibility, it also exposes systems to numerous security threats. Cyberattacks today are motivated by a wide range of objectives, including information theft, financial gain, espionage, competitive advantage, personal retaliation, and large-scale manipulation. As technology evolves rapidly, understanding the vulnerabilities of a system and strengthening their resilience has become a necessity.

Modern computing environments often process sensitive data while prioritizing performance over security. As these systems become increasingly interconnected, ensuring the confidentiality, integrity, and availability of data is more critical than ever. Traditional security mechanisms, such as firewalls and intrusion detection systems, provide valuable protection but can struggle to detect subtle or low-level attacks. In addition, they typically overlook the security insights that can be gained from the physical behavior of the hardware itself.

A promising yet underexplored method for increasing security lies in the use of on-chip thermal sensors, which are standard components of modern processors. These sensors were originally intended to prevent overheating and maintain stable operation, but they also capture temperature fluctuations caused by system-level activity. The CPU temperature correlates with electrical properties, such as applied current and voltage, which means that different workloads or modified versions of the same workload can produce distinct thermal signatures even when they follow similar execution patterns.

Monitoring CPU temperature therefore presents an opportunity to detect anomalous behavior. If malicious code introduces additional computations, activity bursts, or irregular resource usage, these changes can be reflected as deviations from the normal thermal profile of a program. By recording thermal signatures and comparing them with expected behavior, attacks can be detected at runtime. Machine learning techniques can further enhance this process by learning what constitutes normal operation and identifying temperature patterns that fall outside this boundary.

In this work, a novel security approach is proposed that uses Intel's Digital Thermal Sensors (DTSs) to implement temperature-based anomaly detection. Thermal signatures are shown to be effective for security monitoring, even in noisy environments, providing an additional layer of defense that complements existing software-based mechanisms.

1.2 Objective of the research

This thesis aims to address the under-developed field of thermal side-channel attacks and countermeasures. The key objectives are the following.

- Perform a comprehensive analysis of the existing attack and countermeasure techniques, categorize them, and detect trends.
- Propose a novel detection method using the thermal sensors of the Intel CPUs.
- Test the proposed method in various environments and on different processor types.
- Improve potential attack detection using Machine Learning techniques.
- Increase the knowledge of how the CPU works, through thermal analysis.

1.3 Overview of the thesis structure

The structure of this thesis is as follows:

Chapter 2 presents a comprehensive review of the literature on attack and countermeasure techniques in the last decade. Divides the techniques according to the components they target and present the trends.

Chapter 3 proposes the *Hot-n-Cold* detection technique and presents the results of this method on an HPC system.

Chapter 4 proposes two extensions of the above technique, namely *BeatTheHeat* and *WhenThingsHeatUp*, which test it in different noisy scenarios. Moreover, this chapter presents different experiments that lead to better knowledge about the termal dissipation of CPU.

Chapter 5 concludes the work, summarizing the contributions and the direction for future work.

CHAPTER 2

2 A State of the Art in CPU's Vulnerabilities

2.1 About Attacks & Countermeasures

The release of confidential data can be categorized by different models, such as access to information, threat models, principles, and the nature of its state. By accessing information, leaks can be based on weak points in software or hardware. To exploit *hardware-based channels*, the attacker requires a measurement tool and physical access to the system, or at least access in the room where the system is, to extract hardware parameters. Whereas to exploit *software-based channels* the spy needs only software access that can be done remotely.

Some examples of attacks using hardware-based channels are: gaining knowledge about the information processed by a communication equipment by reading LED status indicator sequences [2], finding passwords from heat traces resulting from authentication using PINs or patterns on mobile phones [3], recognizing the key pressed on keyboards using the emitted sounds [4] and emanations of displays with cathode-ray tube and liquid crystal [5], or covert channel communication between two computers situated in one room using heat emissions [6]. Even though these attacks are very easy to visualize and very palpable, their limited functionality does not make them the object of my study, so hereafter the focus will be on software-based attacks.

The threat models involve unintended or unauthorized transmission of information, but with different features, and are mainly two: *Covert channels* and *Side-channels*.

In covert channels, there is deliberate and unauthorized exchange of information between two entities in a manner that prevents detection or monitoring by security mechanisms. Covert channels appear when one program alters the state of the system based on a specific protocol, while another program monitors these changes to interpret the transmitted data.

However, in a side channel, a trusted party accidentally discloses confidential data, which is a side effect of the normal operation of the system. The unintentionally disclosure of secret data is done in such a way that the victim remains unaware of the alterations during execution. These channels frequently leverage physical characteristics or implementation attributes of the system, such as power consumption [7], electromagnetic emanation [8], timing variations [9], or memory access [10], [11], [12], and can work bidirectionally.

If we examine the nature of a channel's state, both *persistent* and *transient* states can be identified. Persistent-state channels introduce a vulnerability related to the storage capacity of a microarchitectural resource. For example, accessing specific data in cache can determine a connection between the victim's data and the attacker's data. To use these types of channels in attacks, techniques such as Prime+Probe [9] [13] are needed.

The transient-state channels introduce a vulnerability associated with high parallelism

in new, state-of-the-art CPUs that execute instructions using out-of-order or speculative ways. The risk of leakage arises when speculative execution proves incorrect, as the side effects introduced by microarchitectural state persist beyond the transient execution, which can be recovered by an attacker. A chain of widely recognized ”speculation” attacks is Spectre [14], Meltdown [15], Foreshadow [16] and other attacks in this area, such as [17], [18] and [19]. These attacks force the speculative execution to read secret data that the attacker will use as a side channel to deduce values [20].

In terms of what the channels are based on or their principle, there are three main categories of *access-based*, *timing-based*, and *trace-based* channels.

Access-based attacks involve taking advantage of vulnerabilities or weaknesses by directly reaching certain information on the addresses that the victim accesses. For these attacks, techniques such as Prime+Probe [9] [13] are needed, which use a process that probes the cache and calculates the latency, to establish if there was a hit or a miss regarding the data [21].

Timing-based attacks are stealing valuable information by using the method of observing cache hits and misses by measuring the time of different operations of the victim. Therefore, these types of attacks are effective in determining the entire execution time, facilitating the recovery of secret information associated with the victim. For such attacks, techniques such as Evict+Time [9] are needed. Timing-based measurements are very difficult to perform in state-of-the-art CPUs because of the multitude of performance optimizations implemented in those processors. [22]

Attacks exploiting **trace-based channels** are considered hardware attacks by the authors of [23] and [24]. They state that traces of cache hits and misses are difficult to obtain from software [23] or that physical access is needed to obtain traces for power, for example [24]. Recently, those statements are no longer valid because research claims that both the power and the temperature of a CPU can be read from software. Therefore, remote power [25] [26] and thermal [27] attacks are possible, which are the key insights in this work.

2.1.1 Scope

This survey focuses on the numerous ways to exploit side- and covert-channels that can appear in modern CPUs, taking into consideration the processor and its nearby components. Whereas hardware-based attacks are easily visualized and tangible, their restricted functionality does not align with the chapter's objectives. Henceforth, the emphasis will shift to attacks made from software in the last decade.

Selecting articles in the last 10 years ensures relevance for modern systems, as cache attacks started to gain visibility around year 2014. Emerging technologies such as cloud computing and multicore architectures have introduced new attack surfaces not covered in older studies, making the 10-year time frame essential to address current attack techniques and countermeasures.

The articles were selected using Google Scholar using relevant keywords. Articles with more than 100 citations were selected to ensure the inclusion of widely recognized and influential studies. In areas where the research was less developed, the articles with the highest citation counts were selected to ensure the inclusion of the most impactful studies.

In the following, the techniques, classified according to the aforementioned categories or other relevant schemes, are presented in chronological order.

2.2 Common targeted encryption algorithms

Encryption uses complex algorithms to transform data from a readable state to an unreadable one, a state which can then be decrypted using a key provided by the sender of the message. Symmetric key algorithms use identical keys for both encryption and decryption steps, whereas asymmetric encryption uses two keys, public and private, respectively. These algorithms are widely used to protect all types of sensitive information in all domains related to data, such as storage, communication, authentication, etc. Naturally, if the important data are secured, the attacks will aim the encryption algorithm to access the confidential information. This is done by extracting the keys, bypassing the encryption, or taking advantage of the implementation weaknesses. The most widely used cryptographic algorithms for attacks are presented in the following.

AES (Advanced Encryption Standard) AES is a symmetric system that consists of four steps:

- Substitute: Plaintext is replaced with encrypted text using a cipher.
- Shift: All the rows, with the exception of making the first one, are shifted by one.
- Mix: The columns are mixed using the Hill cipher.
- Further encryption: A part of the encryption key is utilized to further encrypt the data block.

RSA (Rivest-Shamir-Adleman). RSA is an asymmetric system consisting of the following steps:

- Two prime numbers are selected.
- Their modulus is computed as the public key.
- The Carmichael totient function is computed.
- An integer that is coprime to the Carmichael totient function result is selected.
- The modular multiplicative inverse of the modulo between the two coprime numbers above is computed as the private key exponent.

ECDSA (Elliptic Curve Digital Signature Algorithm) ECDSA is an asymmetric system that needs a double number of bits from the elliptic curve for the signature, following the steps:

- Key generation: A private key is generated, together with its correspondent public key, using the elliptic curve base point.
- Sign: The message is hashed, a random integer is used to compute an elliptic curve point, and all of them are used to calculate the signature components.
- Verification: The message is hashed again and the signature components are validated using the public key and elliptic curve calculations.

ElGamal ElGamal is a hybrid symmetric-asymmetric system that consists of the following steps:

- Key generation: Select a large prime number and a base number (generator) together with a secret number (private key), and use it to compute a public key.
- Encryption: Change the message to a number format. Choose a random number, then use it with the public key and generator to create two encrypted values.

- Decryption: Use the private key and one of the encrypted values to find a shared secret. Use the shared secret to convert the other encrypted value back into the original message.

2.3 Common used techniques for cache attacks

The most common attacks are those using the cache and are based on access or timing, and they are usually using some specific techniques which will be presented next.

2.3.1 Timing-based attacks

EVICT+TIME (E+T) [9] is a technique that aims to evict cache lines along the measurement of the whole execution time, following the steps:

1. The attacker stimulates the victim to execute some code in such a way that it will initialize a set and determine the medium execution time.
2. Evict: the attacker evicts the targeted line and determines the victim to execute again.
3. Time: the attacker establishes the time in which the victim executes. A difference in execution time indicates that the targeted line was accessed. A longer time determines a cache miss, and a shorter time shows a cache hit.

There are several limitations associated with this approach.

- It assumes knowledge of the mapping from virtual to physical addresses;
- It assumes the capability to initiate encryption and exert control over its initiation and conclusion;
- The timing measurements during the encryption process have low accuracy;
- It has significant timing noise from scheduling, misses in the page table, and other causes;
- It leads to a significant amount of time spent to fully recover the key.

2.3.2 Access-based attacks

PRIME+PROBE (P+P) [9] [13] is a technique that detects if the victim evicts the attacker's working set, following the steps that are also presented in Figure 1:

1. Prime: the attacker fills one or more sets in the cache with data.
2. It allows the victim to execute.
3. Probe: the attacker measures the time of the access to the same previous accessed lines to check if any of them was evicted. A bigger time leads to the conclusion that the victim has accessed an address that maps to that set.

Some advantages consist of the speed of the method and the difficulty of being influenced by random factors both related to the attacker measurements or how long it takes for their own actions to execute. Additionally, since multiple cache sets can be examined concurrently during the probing stage, fewer victim executions are necessary.

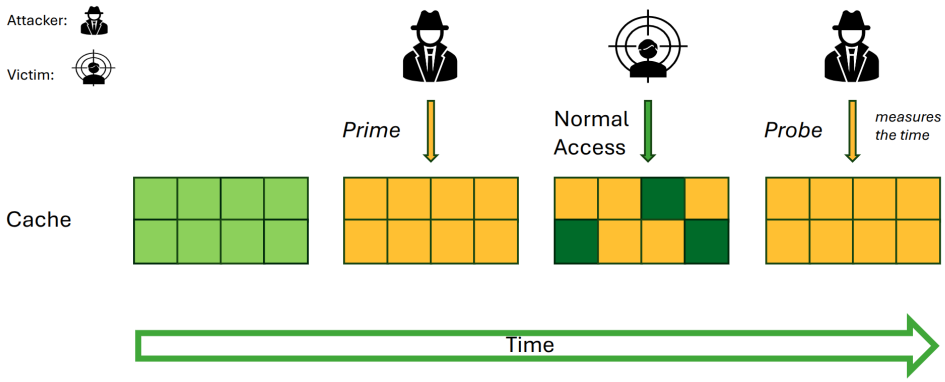


Figure 1: Prime+Probe method.

FLUSH+RELOAD (F+R) [10] is an extension of the [21] technique, which is the inverse of the above, and is based on shared virtual memory, following the steps that are also presented in Figure 2:

1. Flush: the attacker flushes a line in cache that is shared, using a specific instruction or using the contention method.
2. It allows the victim to execute.
3. Reload: the attacker times the reloading of the evicted line while touching it. If the time is small, it means that the victim touched the line, and if it is big, the victim did not touch it.

The advantage over the above method is that the attacker can aim at a certain cache line, rather than an entire set in cache.

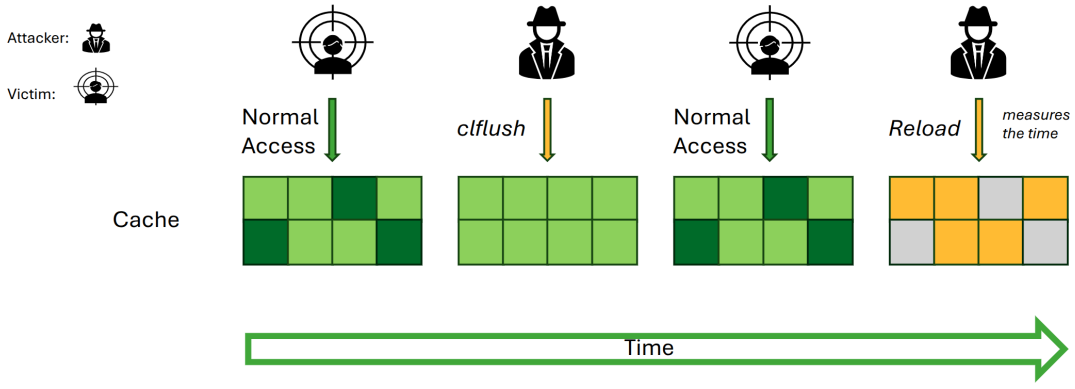


Figure 2: Flush+Reload method.

FLUSH+FLUSH (F+F) [11] is a variant of the Flush+Reload technique that provides similar results; however, it has the advantage that it is harder to be detected as it does not access the memory and does not create such a large number of cache misses. As

its name suggests, it continuously flushes a memory line and establishes the time of the flushes to check if the data were accessed, as seen in Figure 3.

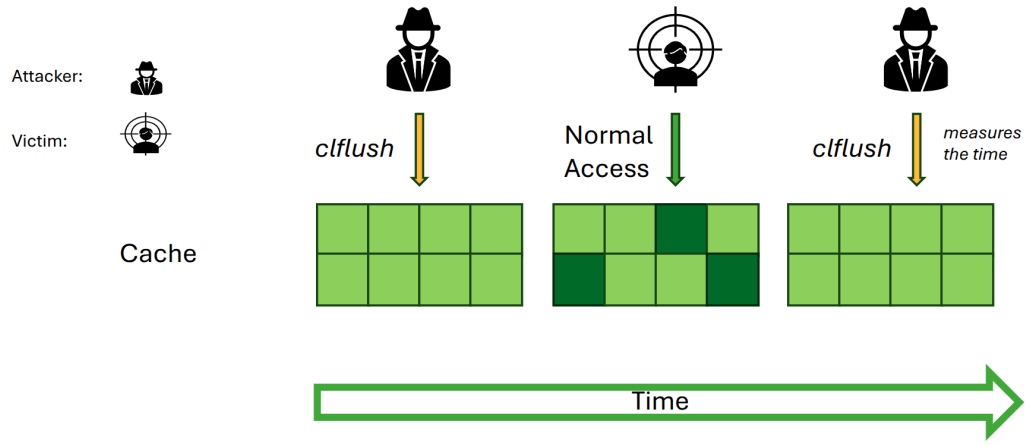


Figure 3: Flush+Flush method.

EVICT+RELOAD (E+R) [28] is a variant of the Evict+Time technique. However, the major changes made to it lead to a transformation in a trace-driven one. E+R works as follows, as seen also in Figure 4:

1. Evict: the attacker evicts some lines in the cache.
2. It allows the victim to run, remaining in an idle state.
3. Reload: the attacker executes a reload and establishes the time to observe if there was a cache hit or miss.

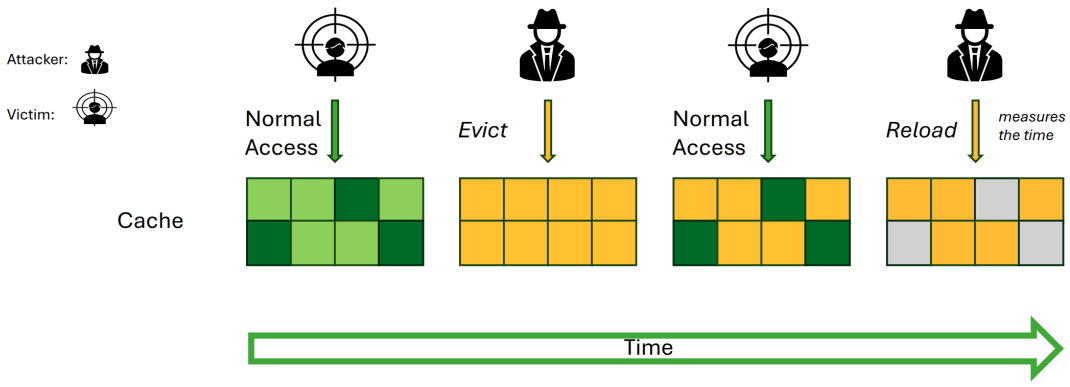


Figure 4: Evict+Reload method.

PRIME+ABORT (P+A) [29] is a technique that, unlike the others, does not use a

threshold when detecting the eviction set, but takes advantage of Intel hardware properties, as follows:

1. Prime: the attacker uses its own set of data to prime the cache.
2. Waits until the victim executes and evicts the primed data.
3. Abort: the eviction of primed data will set a flag which will cause the system to start the mechanism of aborting.

The abort mechanism helps the attacker determine when the cache was accessed by the victim, which makes the technique very effective.

INVALIDATE+TRANSFER (I+T) [30] is a technique that leaks secret information across processors (on multiprocessor platforms) bypassing the CPU cache separation. It works based on the idea that platforms need to maintain cache consistency in the case of invalid locations by synchronization between CPU caches, which leads to timing differences. Steps:

1. Invalidate: the attacker evicts the shared part of the cache.
2. Waits for the execution of the victim.
3. Transfer: the attacker establishes the time. In this way, it will know if the memory that was evicted is synchronized from the cache across processors.

RELOAD+REFRESH (R+R) [31] is a technique that monitors the accesses to memory without evicting victim's data, based on the replacement policy. The block ages indicates the replacement candidate that the attacker will force to be the data inserted by them in cache.

1. Waits for the execution of the victim and its access to those data. If the victim uses the data that was placed by the attacker, the age will decrease and it will no longer be the replacement candidate.
2. Reload: the attacker can check the cache state by reading the last item of the eviction set, forcing the replacement of the candidate. By measuring the time, they will know if the data was used by the victim (slow) or not (fast).
3. Refresh: the attacker reverts the cache to the original state.

COLLIDE+PROBE & LOAD+RELOAD [32] are two techniques used to identify the victim's accesses to the memory on AMD processors.

Collide+Probe (C+P) works using the following steps:

1. Collide: the attacker uses a precalculated address to trick the system into updating its cache prediction.
2. The victim's tasks are scheduled. If the victim does not use a specific address, the attacker's data remain available in the cache. However, if the address is used by the victim, the attacker's data becomes inaccessible.
3. Probe: the attacker checks how quickly they can access the precalculated address. If the targeted data was not accessed by the victim, access is fast. Otherwise, access becomes slow.

Load+Reload (L+R) works using the following steps:

1. Load: the attacker loads a different address with the same physical tag as the target address. This makes the line of cache containing the target address inaccessible

from the fastest cache level for other threads.
2. The victim process is scheduled. If it tries to access the targeted cache line, it experiences an L1D cache miss, meaning it must fetch the data from a slower cache level, which invalidates the attacker's cache line.
3. Reload: the attacker then measures how quickly they can access the address loaded in the first phase. If the targeted cache line was targeted by the victim, the attacker will notice a miss in the cache and a slower access time. Otherwise, the attacker sees a faster access time. This helps the attacker determine whether the target address was accessed by the victim or not.

PRIME+SCOPE (P+S) [33] is a cache contention monitoring technique that operates without the need for windows. It permits an attacker to track the accesses of a victim to a specific address, called target, in two steps:

1. Prime: the attacker first removes the address that is targeted from the cache, by using an eviction set, and then manipulates the cache to ensure that the chosen line remains in a specific cache level while being available in lower-level caches as well.
2. Scope: continuously retrieves the chosen line in the lower level of caches and establishes the latency of the access, without disturbing the cache state.

This technique effectively maintains the cache state in both the lower-level and higher-level caches throughout the monitoring process, thereby overcoming the observer effect.

2.4 Caches

Caches are compact, swift, and costly memory units positioned between the main memory and the processor. They contribute to reducing the time required to fetch data from the main memory by storing data that are very likely to be used by the CPU in the near future. When the caches reach full capacity, a replacement policy is used to determine which location should be vacated to create space for new data. The most commonly utilized replacement policy algorithm is the Least Recently Used (LRU). Caches are typically organized into three levels, with the first level (L1) being faster and smaller than the last level (L3 / LCL). Each cache level is further divided into sets, in which each memory location can be mapped. The caches have two or more ways that determine the number of locations a set can have. Due to their design, caches can potentially leak information, often through timing- or access-based methods.

2.4.1 Cache-based Attacks

The cache-based attacks use mainly the attack techniques presented above. The following is a short description of the most important attacks in the last 10 years. Additionally, for a more optimized overview, Table 1 shows the techniques used for each attack, together with the processor on which the attacks were run, the attacked algorithm, the reliability of the attack, and the performance overhead introduced.

2014. The Flush+Reload [10] attack extracts the private key of the RSA algorithm, recovering 96.7% of the bits of the secret key, tested firstly, using two processes running in a single OS and secondly, using two processes running on separate VMs. In [34],

Table 1: Cache-based attacks

Attack	Year	Technique	Targeted Algorithm	System	Reliability	Overhead
[10]	2014	F+R	RSA	Intel i5-3470	96.7% of bits	A single decryption
[34]	2014	F+R	ECDSA	Intel Core i5-3470	Full	200 signatures
[35]	2014	F+R	AES	Intel i5-3320M	Full	400 encryptions
[36]	2014	F+R	Activities of victim	PaaS Cloud	No. of items in shopping cart	-
[37]	2014	F+R	ECDSA	HP Elite 8300	571 bits	1 signing process
[38]	2015	F+R	ECDSA	Intel Core i5-3470	Full 256-bit, 521-bit key	13 signatures (256-bit), 40 signatures (521-bit)
[28]	2015	F+R, E+R	Keystroke, AES	Intel i5 Sandy Bridge	Over 90% of key presses; 64bits AES	16-160 encryptions
[39]	2015	P+P	AES	Intel i5-650	Full	650 encryptions
[40]	2015	P+P	ElGamal, RSA	Intel Xeon E5 2690, i5-3470, Dell R720 (S)	Full 3072-bit Key	79,900 observed exponentiation
[41]	2015	P+P	Safari, Tor, Firefox	Intel Core i5, i7 Sandy & Ivy Bridge	82.1% accuracy	5000-element vector
[42]	2015	F+R	AES	Intel i5-2430M	Full	30,000 encryptions
[43]	2016	F+R	ECDSA	Intel Core i5-3470	Full	6 signatures
[44]	2016	P+P, F+R, E+R, F+F	AES	Krait 400 Cortex-A53, A57	Full	3,707 encryptions
[45]	2016	F+R	DSA	Intel X5660, AMD Opteron 2435	Full	260 SSH, 580 TLS handshakes
[46]	2016	P+P	Cloud, RSA	Amazon EC2, Intel Xeon E5-2670v2	Full 2048-bit key	4000 decryptions
[47]	2016	P+P	AES	Intel Sandy Bridge	Full	-
[11]	2016	F+F	AES	Haswell Intel Core i7-4790	64 bits	-
[30]	2016	I+T	AES, ElGamal	AMD Opteron server	Full	20,000 encryptions
[48]	2017	F+R	RSA	Intel i5-3470	Full 1024-bit Key	-
[29]	2017	P+A	AES	Intel Skylake i7-6600U	outperforms P+P	-
[49]	2017	P+P	RSA	-	70% RSA-2048 key	300 decryptions
[50]	2017	P+P	RSA	Intel Core i5-6200U	Full 4096-bit key	11 traces
[51]	2019	E+T	AES	Intel Core i7-3770, E5-2640 v4	Over 50% of the key	2^{19} encryptions
[31]	2020	R+R	AES, RSA	Intel Core i5-7600K	Comparable with F+R and P+P	-
[52]	2020	P+P	SSH	Intel Xeon Silver 4110	85% accuracy	-
[53]	2021	P+P	AES	Apple A10 Fusion	Reduce AES-128 security by 50 bits	2× reduction of traces
[33]	2021	P+S	AES	Intel Core i7-7700K	First-round recovers 64 bits	Fewer traces than P+P
[54]	2023	S2C	AES	Apple M1	Full 16-byte key	Similar with F+F & P+S

the authors combine Flush+Reload with lattice techniques to recover the secret keys for an 256-bit elliptic curve algorithm. In [35] is presented an attack across cores and virtual machines in a realistic server setting. It combines Flush+Reload technique with the advantages brought by the dedeuplication mechanism, and it recovers the full key of AES. Flush+Reload technique is used in cache-based attacks on PaaS environments as well [36], where information about products in a shopping cart is retrieved, together with attacks that reset passwords and break encryption of authentication applications. In [37], the attack against elliptic curve cryptographic algorithms implemented using the Open SSL Montgomery ladder recovers most of the bits of the k scalar.

2015. The attack in [38] greatly reduces the number of signatures needed to restore an entire private key. Cache Template Attacks [28] is a generic technique that can extract in an automatic way, the keystrokes of a user with a rate of 90%, using Flush+Reload and Evict+Reload methods. Moreover, it improves the amount of necessary encryptions to extract the AES secret key to less than 200. Another version of Prime+Probe attacks is [39], which extends the technique limitation of using only L1 cache to L3 cache, with a performance slightly lower than Flush+Reload's. This attack recovers the AES key in a cross-VM setting and presents a major risk for cloud servers. Another Prime+Probe attack on the LLC is presented in [40], implementing a covert channel with a high bandwidth of 1MB/s, extracting ElGamal and RSA keys. An extension of this technique is [41], which is created to extract user behavior in web browsers when the victim surfs on an untrusted web page controlled by the attacker. Another attack that recovers the full AES key is [42], which exploits the memory deduplication optimization technique, attack which works in a fully asynchronous way, easy applicable in real-world scenarios.

2016. A performance degradation attack that amplifies the Flush+Reload side channel attack, is used in [43], leading to a high reduction in the amount of signatures required to extract the ECDSA private key. In [44], the first cross-core cache attack is introduced on non-routed Android smartphones running on ARM platforms. They implement all attack techniques P+P, F+R, E+R, and F+F outperforming all existing covert channels. The first cache-based side channel attacks on the TLS and SSH protocols are presented in [45], which recovers the full key due to a vulnerability in the implementation of the DSA signature scheme. A Prime+Probe attack is presented in [46], used to recover a RSA secret key from a colocated instance. Another Prime+Prime attack is presented in [47], which recovers the AES key and does not relay data sharing between the attacker and the victim, having a comparable effectiveness with the Flush+Reload technique. Flush+Flush attacks [11] are categorized as fast and sneaking attacks in comparison with Flush+Reload and Prime+Probe as they are 6.7 times faster than the other covert channels and cannot be detected using the detection methods based on hits and misses in cache. Invalidate and Transfer technique [30] relies on the data sharing across cores, regardless of where they are positioned within or across processors or across CPUs in socket servers, and is used to recover the full AES and ElGamal keys.

2017. Prime+Abort [29] is another cache attack technique on Intel TSX that is three times faster than LLC Prime+Probe on Skylake architecture. The full 1024-bit RSA key can be recovered if taking in account the direction of the sliding window, as shown in [48]. The authors also state that the left-to-right method leaks more information than the right-to-left method. An attack that can extract secret information from SGX enclaves is [49],

where the full RSA-2048 key can be extracted, making it more effective than the previous ones. Another attack on SGX enclaves is [50], which can be applied on native machines or Docker containers, and can extract the full RSA-4096 key.

2018. In 2018 the Spectre [14] and Meltdown [15] attacks appear, but Iwill not include them in the cache attacks chapter, but in the speculation one.

2019. Another attack on LLC is presented in [51], which increases the speed and accuracy of the Evict+Time techniques, and is applied on 4- and 10-core CPUs, where the spy threads run in parallel and in an optimal way.

2020. The Reload+Refresh attack technique [31] is founded on the policy of cache replacement in the CPU, hard to be detected since it can track victim accesses to cache without creating additional evictions, with results comparable to the other existing attack methods. NetCAT [52] is a Prime+Probe network-based cache attack on a remote machine, that can create a covert channel among a sandboxed server process and a network client and gain secret information in a SSH connection.

2021. The Prime+Scope technique [33] uses cache contention to recover the AES key, outperforming the other cache attack techniques by two orders of magnitude. An adaption of Prime+Probe used to attack Apple iPhones is presented in [53], using reverse engineering on the memory hierarchy, the access-driven attack can reduce the AES security with 50 bits, using half of the traces needed in similar attacks.

2023. An attack on Apple M1 is [54], which is based on the synchronization instructions to create changes in the state of the microarchitecture when an L1 cache eviction appears. Together with reverse engineering they are able to reconstruct full AES-128 keys.

2.4.2 Cache-based Countermeasures

In this subchapter, the countermeasures for attacks on caches are divided into seven categories, keeping in mind the timeline of techniques, as seen in Table 2.

Constant time techniques A method to protect cryptographic code involves ensuring that its actions are not influenced by data. This means that factors as cache accesses or branching sequences remain constant regardless of the key or plaintext. This tactic is widely used to mitigate vulnerabilities associated with execution time, which can be exploited in remote attacks. However, due to the hardware complexity, these techniques are hard to be implemented without reducing the performance as well.

2014. Another disadvantage of this technique is that it is dependent on the platform as shown in [55], as is not efficient for the ARM platforms.

2015. Through time, several analysis tools and frameworks were created to make the developer use more constant-time code, for example CacheAudit [69].

2016. Another option is to change the *clflush* instruction to become a constant-time one, so that the time remains persistent for both hits and misses in cache as suggested by the authors in [11].

2017. Further, the dependence of the platform is visible in CacheBleed [70], as it was created for Sandy Bridge processors, and not working on processors that do not include multithreading.

Injecting noise techniques The concept of fuzzy time [90] works by introducing noise into observable events of a process, serving as a covert channel countermeasure. The

Table 2: Cache-based attack countermeasures

Technique/ Year	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024
Logical hardware isolation and partitioning	[55]	[39]	[56] [57]	[58]	[59] [60]	-	[61]	-	-	-	-
Randomization	[62]	-	-	-	[63]	[64] [65]	[66]	[67]	-	-	[68]
Constant time	[55]	[69]	[11]	[70]	-	-	-	-	-	-	-
Partition time Cache flushing	[71]	-	-	[72]	-	-	-	-	-	[73]	-
Inject noise	[55]	-	-	-	-	-	-	-	-	-	-
Scheduler-based	[74]	-	-	-	[75]	-	-	-	-	-	-
Auditing, Detection and Evaluation techniques	-	-	[76] [77]	[78] [79]	-	-	-	[80]	[81] [82] [83]	[84] [85] [86] [87] [88]	[89]

idea is to stop the abuse of a timing channel by introducing so much noise in the attacker's measurements, such that they become effectively useless.

2014. However, in [55] it is stated that simple noise injection is not efficient to obtain security, and they propose an anti-correlated noise technique that closes the channel completely, yet it degrades the system performance very much.

Partition time techniques This category includes attacks that depend on either simultaneous or sequential access to shared hardware and the countermeasure will either offer exclusive access in time slices (for simultaneous accesses) or managing the transition between time slices cautiously (for sequential accesses). In this category, the most used method is cache flushing.

2014. Flushing the cache during a context switch can serve as a mitigation for side-channel attacks that target the relatively small L1 cache. The first state-cleansing mechanism for the L1 cache was proposed in [71].

2017. Kernel address space isolation is used in [72] to fully isolate the user address space from the kernel in a way that exceptions will not lead the path to the kernel address space.

2023. FlushTime [73] is a technique created for ARMv8 architectures that forces the timer and cache flush instructions to execute and pull together only in the kernel space. This method can resist to all attacks based on flushes.

Logical/physical partitioning and isolation techniques Simultaneous attacks can be mitigated by dividing hardware resources between cores or threads. This category includes: disabling the hardware threading, the page or physical memory sharing, hardware partition the cache, cache coloring, partition the spacial network or migrate the VMs.

2014. Cache coloring is a software approach that divide the memory in some colored pools that are secured. Physical frames will not be mapped to an identic cache set if their addresses do not correspond to the colored bits. These techniques are more effective for

cores with much simpler structures, instead of complex ones, as stated in [55].

2015. To create support for multiple colors [39] reverse engineering the hash functions for different processor models.

2016. Preventing physical memory sharing between security domains was used in CacheBar [57] countermeasure, which creates a copy of the physical memory page and protects the system against Flush+Reload and Prime+Probe.

Hardware cache partitioning was used in CATalyst [56] to partition the LLC into a cache that can be managed from both software and hardware in a hybrid manner. It establishes two partitions: a secured one and a non-secured one. The secure partition exclusively accommodates cache-pinned pages that are secured, while the partition that is non-secured operates akin to conventional cache lines controlled by replacement algorithms. This method mitigates LLC timing attacks stemming from cache line conflicts, securing RSA algorithm with a very low performance impact.

2017. Cloak [58] leverages Intel TSX to defeat attackers by monitoring sensitive data or code through misses in cache. When a TSX transaction is in progress, if the cache location containing data that is security sensitive is evicted, the respective transaction will be completed.

2018. In [60], the introduction of data-oblivious extensions for ISA, named *OISA* is suggested, to enhance resistance against side-channel attacks. OISA utilizes a method of way partitioning to establish a memory partition that is oblivious, to be a segregated spot within the first level of data cache. DAWG [59] introduces minimal hardware modifications to completely segregate cache hits/misses and updates of metadata through protection domains within a set of cache.

2020. HybCache [61] applies partitioning in a selective way, exclusively for execution domains that are isolated, enhancing the flexibility and efficiency of cache resource sharing.

Scheduler-based techniques Scheduling techniques are effective strategies in mitigating timing channel attacks. Different scheduling methods can help minimize the interference of attacking VMs with the memory accesses of victims, thereby limiting information leakage. Reducing the overlap time between VMs is one approach; however, it adds a considerable performance overhead due to recurrent context switching.

2014. To address this, the hypervisor can introduce noise before each VM's timeout, disrupting the transmission of data to attacking VMs via timing side-channels. Noise can be added in memory accesses, as presented in [74], thus the attacker will not be able to differ if the signal comes from the victim or from the supervisor.

2018. Shuffler [75] is another technique that distributes the CPU time with equal probability reducing the vulnerabilities with low performance overhead.

Randomization Randomization techniques are used to create confusion regarding the patterns of cache accesses.

2014. A cache with random replacement policy was designed in [62], where in case of a miss in cache, the data requested is delivered to the processor without necessitating the allocation of a line in cache. As an alternative, the mechanism assigns lines through randomized fetches in an window with configurable neighborhood, surrounding the memory line that is missing.

2018. Ceaser [63] is a method of encrypting and remapping the addresses of the lines

in memory that can reduce the likelihood of conflicts resulting from misses in cache.

2019. An improvement of it is Ceaser-S [64], used to partition the cache into multi-way partitions and implement an algorithm of aleatory placement through these partitions, thereby augmenting the ambiguity in the mapping of memory to cache. ScatterCache [65] converts both the addressing of memory and information process into a aleatory cache set indicator, ensuring that every cache way possesses its unique indicator.

2020. PhantomCache [66] suggested an innovative localized randomization approach, where a loaded memory block is randomly placed within its predefined mapping range.

2021. In [67], a versatile model is presented for randomization-based caches, accompanied by a thorough analysis.

2024. The first set-associative cache that is randomized dynamically was implemented in [68], using a hash login on one cycle to randomize the cache indexes. Additionally, a remapping is done cyclically at a certain time through a relocation scheme to reduce the time interval available for an attack.

Auditing and Detection techniques The auditing techniques are based on monitoring systems to collect online data, particularly to evaluate bandwidth consumption. Through traffic analysis, a system can develop a network security supervisor that is capable of detecting any ill-natured use of shared networks. Detecting an attack allows users to become aware of an attacker's presence and take appropriate countermeasures. When a side-channel attack is identified, implementing and applying an appropriate countermeasure becomes essential. The detection techniques can be based on signature or based on anomaly detection, or even both.

Side-channel attacks often result in abnormal cache references and misses, which can be detected using Hardware Performance Counters (HPC), for example. The counters are implemented using registers that store different information regarding the CPU behavior, such as cache hits and misses, branch misses, clock cycle, and more.

2016. HPC together with neural networks and machine learning are used in [77] to detect Flush+Reload attack (also suggested in [11]), where the time needed by this method is less than the attack time. Another system for detecting cross-VM cache side channel attacks using HPC is CloudRadar [76] successfully used against Flush+Reload and Prime+Probe.

2017. Some techniques for detecting side-channel attacks in Intel SGX using the TSX feature are presented in [78] and [79].

2021. Another detection method that uses the performance counters for Spectre attacks is presented in [80], with an accuracy of 90%, furthermore it detects evasive Spectre attacks with an accuracy of 80%.

2022. Predator [82] is a technique that can detect five different attacks using cache performance counters even in environments with noise, and can tell which kind of instruction is accessing the secret data in the RSA and AES cryptosystems. Kingsguard [81] is a more complex technique that can detect an attack at runtime and apply the mitigation specific for the respective kind of attack. It can detect an attack with an accuracy higher than 95% and mitigate techniques such as Flush+Reload, Flush+Flush and Prime+Probe on RSA and AES algorithms. CacheHawkeye [83] is used to monitor the events in the HPC and create some characteristics of the attacker based on memory events. It can detect Flush+Relaod and Flush+Flush attacks with good accuracy and portability to other

systems.

2023. Another detection and classification of attacks method is SCAGuard [84], which is using a dynamic time wrapping algorithm to check the similitude in cache transition states between the attack and the victims. Frime [85] is using different deep learning methods to detect basic cache side channel attacks such as Flush+Reload and Prime+Probe, but also Prime+Abort. A combination of two neural networks trained to extract the best performance counters combination to be used to detect attacks is presented in [88]. A framework designed to identify microarchitectural vulnerabilities is [87] and another is presented in [86] which works within trusted execution environments.

2024. LmSpec & LmTest [89] is a framework used to find and test leakage models for specific implementation of cryptographic algorithms in different libraries.

2.5 TLBs

TLBs (Translation Lookaside Buffers) are components that play a crucial role in nowadays systems that use virtual memory mechanisms. They function as buffers and are used to store mappings or translations between physical and virtual addresses, resembling set-associative caches. Sharing parts of hardware between privileged and unprivileged modes/users. The storage of mappings within a known set can result in leakages similar to those observed in caches.

ASLR (Address Space Layout Randomization) is a security method that helps mitigate attacks that exploit vulnerabilities related to memory. The idea of this technique is to arrange the virtual memory in random ways, either upon each new code execution initiation or on system boot, making it significantly demanding for attackers to predict the location of code or data within the memory. This technique was implemented by all major operating systems, with recent inclusion of support in mobile operating systems too.

KASLR (Kernel Address Space Layout Randomization) is another security method that helps to randomize the memory layout of the kernel, making it harder for attackers to locate kernel objects and execute kernel-level exploits.

The techniques that I have considered important are summarized, together with Table 3, where an overview of each attack is presented. It includes the system and the operating system on which the attack was performed, the algorithm or component attacked, the reliability of the attack and some possible mitigations.

2.5.1 TLB-based attacks

Probably the first attacks against ASLR implementation is presented in [91]. They devise three methods to accurately deduce the locations of the address of the kernel modules using the timing measurement techniques. The fundamental idea of their research involves reconstructing the map of allocations of the entire kernel space. This is achieved by accessing the kernel space pages from user mode by inducing a page fault. Following the initial page fault, a subsequent access is performed, and the duration of this access is measured. A quicker access is indicative of an allocated kernel page, facilitated by a TLB hit. The implementation underwent testing on distinct CPUs, including both Intel

and AMD, and was executed on various operating systems such as Windows 7, Linux, or virtual machines. The study concludes that these attacks are practically feasible.

Table 3: TLB-based attacks

Technique Name/ Article	**Year**	**OS**	**System**	**Targeted Algorithm/ Component**	**Reliability**	**Possible Mitigation**
Double Page Fault [91]	2013	32 & 64-bit Windows 7 Enterprise; Ubuntu 11.10 Native & VMs.	Intel i7 Bloomfield, Lynnfield; Intel i7-2600 Sandy Bridge; AMD Athlon II X3 455;	CPD algorithm	Reconstructs kernel space 94,92% of all pages on each machine.	Isolated caches; Changes in execution time for page fault handler; Disable the *rdtsc* instruction in user-mode;
Telling Your Secrets Without Page Faults [92]	2017	Ubuntu 15.10, Linux k v4.2.0; Use Graphene; Hyperthreading disabled.	(L) Skylake Intel i7-6500U.	Libgcrypto EdDSA v1.6.3, v1.7.5	Recovers full 512-bit EdDSA session key; 485-bit of secret EdDSA key in 1 run of victim enclave	Déjà Vu [78] for frequent enclave preemtion
AnC [93]	2017	Ubuntu 16.04.1 LTS	Intel i7 Ivy Bridge; Skylake; Haswell; i3 Broadwell; Celeron N2840, Atom C2750; E56/L56/X56 Westmere; Allwinner A64 ARMv8-A; Nvidia Jetson, Tegra ARMv7; AMD FX-8320.	Firefox; Chrome	ASLR derandomization. Reduce the entropy to 1 bit. Recover the last bit. (Fireox); Recover 33/35 bits. (Chrome)	Detection using hardware performance counters. Cache coloring. Reduce accuracy of timers. Separate cache for page tables.
TLBleed [94]	2020	Linux	Intel Skylake i7-6700K, Coffeelake i7-8700; (S) Intel Broadwell Xeon E5-2620 v4	Curve 25519 EdDSA in Libcrypt 1.6.3 not constant time; RSA in Libcrypt constant time	Reduce entropy of ASLR with 8 bits; Reconstruct private key in 97% of cases; Success rate 99.3%; Recover 92% of RSA keys; Success rate 97%.	Disable hyperthreading; Partition the TLB; Extension of Intel CAT to provide TLB partitioning.
TagBleed [94]	2020	Ubuntu 18.04 Kernel v4.19; Ubuntu 18.04.1, Kernel v4.15	Intel Core i7-4650U Haswell; i7-6700K Skylake	-	KASLR derandomization. 94% success rate, reduce KASLR entropy to 1 bit.	Spatial and temporal partitioning of the TLB. Isolate page table cache.
TLB;DR [95]	2022	Linux kernel v5.14	Intel i7-7700K Kaby Lake, frequency scaling disabled	Eviction sets.	TLB desyncronization. Error rates $<$1% on sTLBs and $<$8% on dTLBs.	-

2017. In [92] is stated that an operating system that is not trusted can monitor page accesses in the interior of an enclave, without page faults, leading to different types of

side-effects of the address translation process. The procedure of the attack begins with the creation of a separate spy thread within the untrusted user space runtime. This spy thread operates in kernel space through a modified Graphene-SGX driver, providing the attacker with the capability to send interrupts, examine page table entry attributes, and monitor page table memory. As the victim thread enters the enclave, the spy interrupts it, and the interruption handler determines the pattern of accesses for the aimed page set, using the Flush+Reload technique. The resulting access patterns are recorded by the attacker and analyzed post-processing. To assess the effectiveness of the attacks, EdDSA session keys are extracted from Libgcrypt cryptographic library. The testing was conducted on Intel processors running the Linux operating system.

AnC [93] technique uses an Evict+Time attack, targeting the virtual address translation of the processor. This attack exploits the phenomenon where page-table walks cause the caching pages in the last level of the cache. As a consequence, this attack undermines the randomization of virtual addresses associated with a victim's code and data. The first bits are derandomized by gaining information about the offsets of the cache lines within a page, containing the Page Table Entries (PTEs) stored in case of a Page Table (PT) walk (TLB miss). The last bits are derandomized through identification of PTE offsets and mapping cache sets to PT levels via accessing pages, named the sliding method. AnC was applied on Firefox and Chrome browsers and successfully derandomized ASLR.

2018. TLBleed [96] is a method capable of divulging detailed confidential information regarding the actions of the victim, despite the presence of cache side-channel protections such as CAT and TSX. This technique depends on measuring the latency of unprivileged memory access to differentiate between TLB misses and hits. They explored Elliptic Curve Cryptography (ECC), involving a duplication operation and an addition operation. To distinguish between the two operations, a classifier is trained to identify which function the victim is executing. TLBleed can leak the secret key of 256-bit EdDSA with a 98% success rate. Moreover, TLBleed reconstructs 92% of RSA keys from a secure against Flush+Reload attacks version of RSA algorithm, all from a single trace capture.

2020. TagBleed [94] is a new side-channel attack on tagged TLBs, which can break KASLR, even though the latest mitigations are in place. The technique is based on forcing the kernel to do the simplest possible operation, namely, accessing its own memory. The most important idea is that memory accesses occurring in the same set in tagged TLBs, can be observed by the attacker. Using kernel's uncontrolled but valid memory accesses the KASLR can be broken. For this, the tagged TLB architecture for big pages is reverse engineered, using Evict+Time technique in combination with AnC attacks, which leaded to de-randomizing the KASLR. To make the attack practical, the noise is reduced using TLB flushing techniques. The success rate of this attack is 94%.

2022. TLB;DR [95] paper introduces a TLB desynchronization technique to increase the performance of TLB-based attacks by introducing optimized eviction sets. The authors found out by reverse-engineering method, that the tested systems use non-inclusive, non-exclusive TLBs; the dynamic partitioning takes place during idle states, and the entries are never reintroduced into L2 cache following eviction from L1 caches. They also found a new, unknown replacement policy for sTLB for some Intel CPU families, named (MRU+1)%3PLRU4. The speed benefits introduced by the optimized eviction sets reach up to 20% less time to break ASLR in Page Translation side-channels like AnC [93], 12%

Table 4: TLB-based attack countermeasures

Technique Name/Article	Year	OS	System	Mitigated Techniques	Performance Impact/ Overhead	Performance Evaluation Algorithm/ Benchmark
KAISER [72]	2017	Linux Ubuntu 16.10	Intel Core i7-6700K Skylake	Double page fault attacks [91]; Intel TSX-based Attack [98]; Prefetch side-channel attacks [99];	Overhead: 0.28% runtime; 1MB memory; 0.08-0.68% performance	PARSEC 3.0 [100]; pgbench [101]; SPLASH-2x [102].
Secure TLBs: SA TLB, SP TLB, RF TLB [103]	2019	Linux	RISC-V	SA TLB prevents 10 types of timing-based vulnerabilities; SP TLB prevents 4 more; RF TLB prevents TLB internal collision [91], TLB Flush+Reload, TLB Evict+Time, TLB Prime+Probe [96], TLB Bernstein Attack, TLB Evict+Probe, TLB Prime+Time	SP TLB IPC is 0.5% better than standard TLB; RF TLB IPC is 1.4% higher compared to SA TLB; 10% performance overhead	SPEC 2006 integer + floating point on FPGAs; RSA implementation from TLBleed attack [96]
TLBcoat [104]	2022	Linux kernel v5.12	RISC-V	Protects against state-of-the-art attacks including Prime+Prune+Probe.	negligible	PARSEC 3.0 [100]; Gem5 [105]

shorter median hammer time in Page Translation Rowhammer attacks, like Pthammer [97], and doubled sample rate improvements for L2, in TLBleed [96] technique.

2.5.2 TLB-based countermeasures

The countermeasures for the TLB-based attacks are described next. Additionally, in Table 4, each countermeasure has associated the system and the OS on which was run, together with its performance impact, the benchmark used for evaluation and information about the attack that is mitigated with it.

2017. KAISER [72] is a method that creates a practical kernel address isolation in a way that hardware does not store data about kernel addresses during user mode execution. It provides security against double page fault attacks, introduced in [91]. The technique is using the so-called shadow address spaces where an address space contains the mapping of the user space, but not the kernel. The kernel is mapped into a second address space, which leads to no memory look-ups. Additionally, instead of a complete TLB flush, for complete elimination of the leakage, they disabled the global bits with the advantage of negligible performance overcome.

2019. In [103], two new hardware designs of TLB are presented, that increase security, namely, Static-Partition (SP) TLB and a Random-Fill (RF) TLB. The SP TLB operates on the principle that specific ways are designated for a victim process, while other ways are allocated for all remanent processes, typically considered as potential spying processes by default. The process ID is employed as a means of distinguishing among the victim and the potential attacker. The RF TLB has the capability to uncorrelate the required memory

access from the TLB entries, introducing non-determinism into the observations of the attacker. In the event of a TLB miss, a aleatory address translation is retrieved in the TLB, whereas the address that was initially requested is sent to the CPU without populating the TLB. From a security perspective, the suggested TLBs offer the same level of security as normal TLBs while providing protection against varied attacks.

2022. It was found that the mitigation techniques created for timing cache-based attacks do not work for TLB side channels. Thereby, TLBcoat [104] is proposed, a technique that introduces the re-randomization method, based on a miss threshold and preventing the leakage of Process-Context Identifiers (PCIDs) information as well. The main idea involves employing a miss counter for a process, using a threshold value which is dependent of the TLB's associativity and size. Each miss in the TLB decreases the counter by one. In case of the counter reaching the value of zero, a random value is assigned to the rid, and the miss counter register returns to the threshold value. As the alteration of the mapping function, the translations that are stored previously are highly likely to no longer match during a lookup, clearing entries from the TLB for the present process. Within TLBcoat, the randomization is individualized for each process, which makes impracticable for adversaries to create simple eviction sets by replicating the virtual addresses or generating addresses that correspond to the identical TLB index within processes. This indicates that conventional Prime+Probe attacks are no longer viable.

2.6 Prefetchers

Prefetchers are created to increase the speed of program execution by speculatively fetching data into the cache, data that are likely to be used in the future. Predicting based on locality, the prefetcher brings anticipated data into the cache. When the program accesses the cache, the required data is already present, reducing the latency for accessing the main memory. The prefetching technique can be executed either via software [106] or hardware [107]. Hardware prefetchers employ diverse strategies for predicting and fetching data, proving advantages depending on the specific application being executed on the hardware [108], [109], while software ones let programmers include prefetching instructions within the code [110], [111].

However, there is the possibility that some of the data brought into the cache by the prefetcher may go unused, which can leave behind traces of the data accessed by potential victims. When conducting a time measurement, it may be observed that the data utilized during runtime was already present in the cache, indicating that it was fetched by the prefetcher. If the operational method of the prefetcher is understood, usually done by reverse engineering [112], [113], [114], it becomes possible to trace the previously filled cache locations.

2.6.1 Prefetcher-based Attacks

The attacks on the prefetchers are briefly explained next, alongside their overview in Table 5, where for each attack the system, OS, and the attacked algorithm or component are presented, together with the reliability of the attack and some possible countermeasures.

Table 5: Prefetcher-based attacks

Technique Name/ Article	Year	OS	System	Targeted Algorithm/ Component	Reliability	Possible countermeasure
Flush+Prefetch [99]	2016	Linux Ubuntu, Amazon EC2 PVM, Windows 10	(L) i5 Sandy Bridge, Ivy Bridge; (D) i7 Haswell; Skylake; (L) i3 Broadwell; Amazon EC2 VM; Xeon E5-2650 Sandy Bridge; (M) Exynos 7420 ARMv8-A;	CPU	Defeat ASLR Link between virtual and physical addresses, KASLR exploit	Stronger kernel isolation
PAPP [113]	2019	Android OS	Intel Atom Z3580, ARM Intel Xeon, Haswell, Sandybridge, Skylake	LLC-based AES; ElGamal cipher; Random cache lines	0.81-0.95 CSV [115] vs 0.2-0.48 CSV for traditional Prime+Probe	-
Prefetch+Reload Prefetch+Prefetch [111]	2022	-	Intel Core i7-6700 Skylake; i7-7700K Kaby Lake; Intel Xeon Platinum 8124M Skylake; 8151 Skylake	Square-and-multiply algorithm	Accuracy 96.2%	Write permission checks and constant time execution for PREFETCHW
Prefetch+Time Prefetch+Power TLB-Evict+Prefetch [116]	2022	Ubuntu, CentOS, Linux kernels v3.10-v5.9	(M) Ryzen 5 2500U; (D) Ryzen Threadripper 1920X; 5 3600; 7 3700X (D) A10-7870K (C) EPYC 7402 (C) EPYC 7571 Intel Core i7-8565U	CPU	Recover secret bytes with a success rate of 96.7%	Page table isolation; FLARE[117], Prefetch configuration MSRs; Restrict access to energy measurement
Exploiting the microarch. leakage of prefetching activities [114]	2023	Linux kernel v5.15.74	Intel i7–10700 hyperthreading disabled	AES-128	Recover the full secret key of AES-128	-
PrefetchX [112]	2024	AWS EC	Intel processors	Square-and-Multiply RSA	Recover RSA key	Prefetcher partitioning

2016. Prefetch Side-Chanel Generic Attacks introduced by [99], are, to the best of our knowledge, the first prefetching-based attacks. These techniques enable an attacker to obtain address information and circumvent security protocols such as Supervisor Mode Access Prevention (SMAP), Supervisor Mode Execution Prevention (SMEP), and kernel Address-Space Layout Randomization (ASLR). The prefetching method can bring data into caches from privileged memory locations in the Intel x86 architecture, and also leaking virtual memory translations for both Intel x86 and ARMv8-A. Three attacks have been created. The initial attack recovers an accurate resemblance of the process' paging hierarchy, effectively overcoming both user and kernel spaces' ASLR. The second one sorts out virtual to physical addressing, providing a means to avoid SMAP. The third attack demonstrates how to overcome the ASLR of the kernel on Windows 10, using ROP attacks.

2019. The PAPP [113] technique involves reverse engineering of the prefetcher and the replacement policy. It establishes a prime and probe strategy that withstands interference from intrusive prefetchers on CPUs using in-order techniques. This approach results in a two-fold increase in data leakage in contrast with conventional prime and probe implementations.

2022. In [111], it is noted that the *prefetchw* instruction, designed to accelerate ulterior writes by altering the data's coherence state, reveals two security vulnerabilities. Specifically, this instruction can operate on data that have read-only permissions, and the amount of time taken to execute of this instruction discloses the actual coherence state of the data that was targeted. Building on these findings, they introduced *Prefetch+Reload* and *Prefetch+Prefetch* cross-core attack techniques utilizing caches. These covert-channel attacks achieve very high transmitting rates (782 KB/s and 822 KB/s), whereas in the side-channel scenario, the attacks can observe the victim's pattern of accesses on the respective processor, recovering encryption keys with nearly zero error rate. In terms of speed, these attacks enable the leakage of approximately twice as many secret bytes in contrast with Flush+Reload [10].

In [116], two exploitation techniques are presented, namely, Prefetch+Time and TLB-Evict+Prefetch, which are used to break KASLR on AMD CPUs via timing channels. The leakages consist in page-table level knowledge and the knowledge of an address is cached in the TLB or not. A higher execution time for a *prefetch* instruction means that it is not cached in the TLB. The first technique gains information about the page-table level by measuring the execution timings. The second technique is using the RAPL interface to measure the power consumed by a *prefetch* instruction and gains information about the the page-table level, as well. The last technique gains information about the TLB state by flushing the TLB entry before applying the above techniques. These methods are applicable in real-life scenarios and can break KASLR, monitor the activity of the kernel, and create covert channels.

2023. In [114], the researchers conduct a reverse engineering analysis of the IP-based stride prefetcher and introduce a side-channel attack to retrieve the AES-128 secret key. They strategically delay the probing phase and observe minimal interference. Given this observation, they adjusted the probing schedule to occur after the entire encryption process was completed, as opposed to the initial round where the attacker would typically stop the encryption execution. Ultimately, the attack recovers all the AES-128 secret key.

2024. PREFETCHX attack [112] facilitates side-channel attacks that traverse cores without relying on caches as the source of leaks. This method involves extracting activities based on pages performed by a victim. Furthermore, a thorough investigation of the XPT prefetcher has been conducted, allowing the recovery of the RSA key and the establishment of cross-core covert channels with low error rates.

2.6.2 Prefetcher-based Countermeasures

In addition to the countermeasure description that follows, in Table 6 the countermeasures for the attacks on the prefetchers are summarized, based on the system, OS and benchmark used to test the technique, together with the performance overhead and the attack that was mitigated.

Table 6: Prefetcher-based attack countermeasures

Technique Name/ Article	Year	OS	System	Mitigated Techniques	Performance Impact/ Overhead	Performance Evaluation Algorithm/ Benchmark
Prefetch side channel attack [99]	2016	Linux; Windows; Amazon EC2 VM	Intel x86 ARMv8-A	Double page fault attacks [91]	0.06–5.09% performance overhead	pgbench [101]; apache [118];
KAISER [72]	2017	Linux Ubuntu 16.10	Intel Core i7-6700K Skylake	Double page fault attacks [91]; Intel TSX-based Attack [98], Prefetch side-channel attacks [99]	Overhead: 0.08-0.68% performance; 0.28% runtime; 1MB memory	PARSEC 3.0 [100]; pgbench [101]; SPLASH-2x [102]
Flush+Prefetch [119]	2020	Linux Ubuntu 16.04 LTS	Intel Xeon E5-2643 fixed clock of 3.4Ghz	Flush+Relaod [120]	10% execution time improvement	Square-and-Multiply implementation of RSA cryptosystem
PREFENDER [121]	2022	-	Intel x86 2GHz out-of-order CPU	Spectre attacks using F+R, E+R and P+P techniques	Maintain or improve performance up to 2%	SPEC 2006 [122]

2016. In [99] they suggested a preventive measure against their novel attacks, recommending enhanced kernel isolation. This involves ensuring that syscalls and unrelated kernel threads operate in separate address spaces from user threads. The implementation of this mitigation needs only a small number of adjustments to operating system kernels, with the associated performance overhead being minimal, approximately 0.06–5.09%.

2017. KAISER [72], which was mentioned in cache countermeasures, provides a similar security method against prefetch side-channel attacks as the one presented in [99].

2020. Flush+Prefetch [119] is a method designed to obscure the memory access patterns of a secure application by employing *prefetcht* and *clflush* instructions to access the secure application's memory and generate noise. Consequently, the cache access model of the victim application is encrypted in a manner that thwarts an attacker attempting to collect the real cache access pattern through a Flush+Reload attack. This technique was developed specifically to counteract the Flush+Reload category of attacks and does not necessitate any hardware modifications. The results against the Flush+Reload indicate a reduction in information leakage of 22.3% in one round of decryption, and the information leaked appears scattered without discernible patterns that could aid an attacker in establishing the secret key through multiple iterations. Furthermore, the performance demonstrated a 10.2% improvement.

2022. PREFENDER technique [121] incorporates a DST and an APT to disrupt the second and third phases of an attack. The DST is employed to forecast the eviction of cache lines during the victim's execution, while the APT is utilized to predict the patterns of an access during the measurement of the attacker. The efficacy of PREFENDER in enhancing security was assessed through various side-channel attacks. Leveraging com-

binations of the DST and APT, the hardware prefetcher introduces eviction cache lines into the cache, thereby confounding potential attackers. The experiment demonstrates the defensive efficacy of this technique against attacks such as Flush+Reload, Evict+Reload, and Prime+Probe. Furthermore, performance evaluation indicates that PREFENDER can obtain a speedup, tested on SPEC CPU 2006 benchmark.

2.7 Speculative and Out-of-order Execution

Speculative execution is a technique used to boost performance by predicting instructions and executing them ahead of time, before it is certain that they will be needed. This technique allows the processor to make more efficient use of its resources by keeping them busy. This can lead to significant performance improvements, especially in environments with long pipelines and multiple execution units.

When the flow control of a program relies on a value that is uncached from the physical memory, there is usually a delay of several hundred clock cycles before the CPU can utilize this value. Instead of remaining idle during this time, the CPU employs speculative execution by predicting the direction of control flow. First, it will save the register state and it will run the program speculatively along the predicted route. After the data from the memory arrive, the CPU checks if the original prediction was correct. In the case the prediction was incorrect, the processor clears the speculative execution by reversing the state of the register to the saved one, leading to performance equivalent to idling. However, if the prediction was accurate, the results of the speculative execution are committed and will increase performance.

Out-of-order execution is a performance optimization mechanism that is used to improve instruction-level parallelism and overall efficiency. In a traditional processor, instructions are executed in the order they appear in a program. However, in an out-of-order execution processor, instructions are run immediately after their operands are available, regardless of their original order in the program. Out-of-order execution can significantly improve processor performance by enabling the CPU to execute independent instructions concurrently and reducing idle time.

Branch prediction is another technique used in computer processors to improve the performance of conditional branch instructions, which determine the flow of execution based on certain conditions. It works by predicting the outcome of branch instructions before their actual resolution, based on historical data or patterns. If the prediction is correct, the CPU continues to execute instructions based on the predicted outcome, thereby avoiding stalls in the pipeline. However, in the case where the prediction is not correct, the processor must throw away the speculatively executed instructions and resume execution from the correct branch target.

Transient execution is a microarchitectural optimization mechanism utilized to further enhance performance by speculatively executing instructions beyond traditional control flow boundaries. It includes techniques like speculative execution, out-of-order execution, and others, but it specifically focuses on executing instructions that are not part of the normal program flow, often to predict and preload data or instructions that might be needed in the future.

Although transient execution can significantly improve processor performance by keep-

ing resources busy and reducing idle time, it also introduces security vulnerabilities. For example, Spectre [14], Meltdown [15] and Foreshadow [16] vulnerabilities are the result of exploiting transient execution mechanisms to access privileged information or leak sensitive data from memory.

To mitigate the security risks associated with transient execution, various hardware and software countermeasures have been developed, including microcode updates, operating system patches, and changes to compiler optimizations. These countermeasures aim to prevent unauthorized access to sensitive information while still allowing processors to benefit from the performance improvements offered by transient execution techniques.

2.7.1 Speculative and Out-of-order Execution Attacks

In Table 7 a summary of speculative attacks can be seen, where for each technique the OS and processor, together with the reliability and a possible countermeasure are presented. In the following, each attack is shortly described.

2018. Meltdown attacks [15] force the side effects of out-of-order execution to extract the complete kernel address space, together with the physical memory mapped to it. Shows the fusion of execution in an out-of-order way with a microarchitectural covert channel, enabling the transmission of data from an external receiver through a covert channel. If the out-of-order execution is disposed of, neither the memory contents nor registers are committed. However, memory contents that are already cached persist in the cache. It utilizes the Flush+Reload [10] method, that allows to know wether a particular memory location is cached, enabling its transformation into a microarchitectural state to make it visible. The attack consists of two essential components: first, inducing the processor to execute instructions that deviate from the typical execution path, and second, transferring the resulting microarchitectural side effect. This attack prevents the occurrence of an exception and redirects the control flow following the execution of a sequence of transient instructions. The memory address accessed by these transient instructions becomes cached for subsequent accesses. To create the covert channel, the recipient calculates the latency of an execution of an instruction on the identical execution port. A delayed response time indicates that the sender transmits a bit of 1, while a low latency indicates that the sender transmits a bit of 0. By executing these steps for various locations of memory in an iterative way, the attacker can extract the kernel memory.

Spectre [14] attacks deceive the CPU into executing in a speculative way sequences of instructions that would not typically be executed in a correct program flow. By controlling which transient instructions are executed in a speculative way, the attack can extract data from the memory address space of the victim. This attack merges speculative execution with data extrusion using covert channels. That is, a series of instructions is inserted into the process address space to create a covert channel. Subsequently, the attacker induces the CPU to speculatively execute an instruction sequence in an erroneous mode, thereby facilitating the leakage of information through the established covert channel using Flush+Reload [10] and Evict+Reload [28] techniques. Erroneous speculation occurs through two methods: utilizing conditional branches and employing indirect branches. In the case of conditional branches, deliberate mispredictions are induced during training, resulting in speculative errors that persist without reverting cache contents, allowing the

Table 7: Speculation-based attacks

Technique Name	Year	OS/Kernel	System	Reliability	Possible Countermeasure
Meltdown [15]	2018	Ubuntu 16.10, Linux kernel, v2.6.32-4.13.0, KASLR active; OS X; Windows 10; Docker, LXC, and OpenVZ.	(L)(D): Celeron G540, Core i5-3230M, i5-3320M, i7-4790, i5-6200U, i7-6600U, i7-6700K; (C): Xeon E5-2676 v3, Xeon E5-2650 v4.	Dump physical memory and kernel with up to 503 KB/s with error rate 0.02%; Not working on AMD or ARM.	KAISER prevents Meltdown to a large extent. Permission checks.
Spectre [14]	2018	Linux; Windows; both 32- & 64-bit	Intel Ivy Bridge i7-3630QM, Haswell i7-4650U, Broadwell i7-5650U, Skylake Xeon on Google Cloud, i5-6200U, i7-6600U, i7-6700K, Kaby Lake i7-7660U; AMD Ryzen; Qualcomm Snapdragon 835, Kyro 280; Samsung Exynos 7420Octa Cortex-A57 & Cortex-A53.	10KB/s with $<$0.01% error rate	Disable speculative execution; Serializing instructions; Prevent access to secret data via speculations; Prevent indirect branch poisoning;
FORE SHADOW [16]	2018	Linux	(D) Dell Optiplex 7040 with Intel i7-6700 Skylake	Success rate 99.92%. 100% success rate extracting the full 128-bit key in Launch Enclaves and Quoting Enclaves	Microcode updates; Set logical cores in a HyperThreading to execute into the same enclave; L1 cache is flushed at each exiting event from an enclave.
ret2spec [123]	2018	Windows 10	Intel Core i5-4690 Haswell	Retrieved byte was correct in 80% of cases.	Disable speculative execution; Hardened compilers; Mtigations against accurate timing measurements
SpectreRSB [19]	2018	Linux kernel v4.15.0-22, v4.4.117	Intel Core i7-6700 Skylake; Intel Xeon E5-1620	Not implemented	Microcode patches to secure the speculation using the RSB; Refill the RSB at thr entrance of the SGX enclaves.
Meltdown Prime; Spectre Prime [18]	2018	MacOS Sierra v10.12.6	Macbook with Intel Core i7	99.95% accuracy correctly leaked characters in the secret message for SpecterPrime	Same as Specter & Meltdown.
Thrash+ Reload [17]	2019	Linux, macOS & Windows	(L) Intel Core i5-4200M, i5-6200U, i7-8550U; (D) i7-6700K, i7-8700K; Skylake-based Intel Xeon CPU in Google Cloud; ARM Cortex A75.	Local network: leaks 1 bit in 4 minutes; Cloud network: leaks 1 byte in 8h; 1 byte in 3h AVX-based covert channel	*lfence* for speculation barier (Intel & AMD); Detection of vulnerable code paths (Microsoft); Code analyzers (Linux); Artificial noise in the network; DDoS protection.
SGXPECTRE [124]	2019	Linux SGX SDK v2.1.102; Rust-SGX SDK v0.9.1; Graphene-SGX.	(L) Intel Core i5-6200U	-	Restrict speculation to indirect branches; Retpoline.
Transient Forwarding forwards & Store-to-Leak [125]	2019	Windows; Linux Ubuntu 16.04	Intel i7-6600U Skylake (without KAISER), i7-7600 Kaby Lake, i9-9900K Coffee Lake includes fixes for Meltdown and Foreshadow.	F1-score is 0.96-0.98 for derandomizing KASLR. Recover the last kernel store with 50–80% success rate.	Microcode updates for *verw* instruction to write on top of the contents of the store buffer.
SMoTher Spectre [126]	2019	Ubuntu 16.04, Linux kernel v4.15.0. OpenSSH v7.2	Intel Core i7-6700K, i5-6200u	OpenSSH: 95% confidence Student's t-test, BTI success rates: 16%-25%. OpenSSL: 95% confidence Student's t-test, BTI success rates: 80%.	Disabling SMT; Retpolines.
RetBleed [127]	2022	Ubuntu 20.04 with Linux kernel 5.8.	Intel Coffee Lake; AMD Zen 2.	Break KASLR with 97.5% accuracy. Covert channel with 5.9 kB/s bandwidth and 83.1% accuracy on Intel Coffee Lake, and 20.6 kB/s bandwidth and 96.7% accuracy on AMD Zen 2.	Prevent speculation; Use of eIBRS.
Downfall [128]	2023	Ubuntu 20.04.5 LTS, Linux kernel v5.15.0-48-generic	Intel Core i7-1165G7 Tiger Lake, i7-8650U Kaby Lake; Xeon Silver 4314 Ice Lake Server; Xeon Gold 6230 Cascade Lake;	Attack 100% successful for AES-128 and 86% for AES-264. Covert channel bandwidth 5870.3 bytes/s.	Disable SMT; Disallow certain instructions; Disable the gather instruction; Prevent transient forwarding.

attacker to analyze and extract data. Similarly, with indirect branches, the Branch Target Buffer (BTB) is manipulated to mispredict branches within indirect branch instructions. This misprediction, followed by a nonreverting action on the cache, perpetuates the leakage of information.

MeltdownPrime and SpectrePrime [18] are two attacks that implement the original Spectre [14] and Meltdown [15], using Prime+Probe technique instead of Flush+Reload. The main difference is that MeltdownPrime and SpectrePrime stem from the speculative release of write requests within a system employing an invalidation-based coherence protocol, while Meltdown and Spectre exploit cache pollution during speculation.

Foreshadow [16] exploits an identical vulnerability in the CPU as the Meltdown [15] attack but adopts a distinct attacker model, targeting the compromise of protection domains within intra-address space enclaves, i.e. Intel's Software Guard eXtensions (SGX) [129]. This attack can disrupt enclaves and retrieve the contents of CPU registers, or can gather secret information in real-time without interrupting the enclave of the victim, or even bring secrets in the L1 cache executing the victim. In the initial stage of the attack, enclave data is fetched into the L1 cache by executing the enclave of the victim. Subsequently, a dereference to the enclave secret occurs, followed by speculative execution of a sequence of transient instructions that fill up an entry of an "oracle" buffer into the cache. In the case the CPU detects that it was wrong in the speculative execution of the transient instructions, the register changes that were not yet committed are thrown away and a page fault is triggered. Upon fault catching, the OS invokes the exception handler of the attacker. The time taken to reload each oracle slot is measured, and using the Flush+Reload technique the secret enclave byte is established.

Ret2spec [123] attack is similar to the SpectreRSB [19] attacks, both exploiting return stack buffers (RSB) to trigger mispeculations. In the first attack, the CPU is fooled into executing instructions that would not typically occur in a sequential execution. The objective is to divulge secret information through speculation, such as caching a specific memory area that becomes detectable during normal execution. The attacker induces mispredictions in the return address predictor, redirects speculative execution toward a controlled code sequence, and alters the state of the architecture in speculation. This manipulation can then be identified externally using the Flush+Reload [10] method. In the second attack, the attacker positions itself in close proximity to the victim in the same core. They clear the entries of the shared addresses and contaminate the address space of the victim within the Return Stack Buffer (RSB) with the address of a gadget. Subsequently, the attacker relinquishes to the victim the control of the processor, which will execute a return instruction. This triggers execution in a speculative way to the attacker's injected address on the RSB. Finally, the attacker measures the resulting leakage.

2019. In Fallout [125] it has been shown that even though patches for Meltdown were introduced, Meltdown-like attacks are still possible. These attacks are based on the store buffer, namely, Write Transient Forwarding sends information from stored data to later loads, even if the locations are different, and Store-to-Leak take advantage of the interaction between the store buffer and the TLB to expose data regarding stored addresses. The attacks can leak data, recover control flow, and attack the ASLR.

NetSpectre [17] implements the first version of Spectre [14] attacks on the network, using a remote variant of Evicte+Reload technique, called Thrash+Reload, using an AVX-

based covert channel instead of a cache covert channel. The NetSpectre attack is based on sending a large number of network packets towards the victim. The attack is mounted in one step where the attacker executes constantly to mistrain the processor, by passing alternative valid and invalid network packages. To implement the covert channel, the response time of a request through the network is measured. The time taken by the response is affected by the state of the variable used in the attack, which will lead to a variation in the duration of the response in the range of nanoseconds, thus, averaging over a large amount of network packets will be needed.

SGXPECTRE [124] its a Spectre attack on the SGX [129] enclaves, leverages speculatively executed memory references, and exploits branch resolution latency to generate observable cache traces through side-channel methods, thereby enabling the reading of memory content. Unlike other attacks, SGXPECTRE uses Flush+Reload [10] side channels to leak information. This attack compels an interaction between the speculative execution of memory references within enclaves and memory locations outside the enclave used by them. This is done by flushing an array of memory prior to the attack, reloading every entry, and establishing the time for a reload to discern whether the entry was accessed by the enclave code during the speculative execution. There are two possible end-to-end attacks. The first attack involves the attacker accessing arbitrary register values within any enclave program. This exploit is feasible because, during Asynchronous Exit (AEX), the register values are preserved in the Secure Storage Area (SSA) before exiting the enclave. Since the SSA resides within the enclave and maintains a fixed address upon enclave loading, it enables the adversary to disrupt enclave execution frequently with AEX and capture snapshots of its SSAs. This capability facilitates the trace of the register values step by step during execution, rendering the attack powerful. The second attack is capable of extracting the keys from enclave memory while they are used in encryption procedures. To decipher the encrypted data, the attacker may extract the encrypted key and subsequently apply a decryption algorithm.

SMoTherSpectre [126] is an attack for multithreading processors that uses a tool to automatically discover gadgets in popular libraries and leak information in real-life situations when port contention takes place. As a consequence, an unauthorized enemy process can discern if a co-located process of the victim is executing an instruction on a specific port. A real-world attack was created on OpenSSH and OpenSSL where a local attacker initiates a TCP connection, and calls the Branch Target Injection (BTI) gadget where the private host key is referenced by the pointer. It will affect the forking of new processes by the server when the host keys are loaded from the disk. The cryptographic values stored in memory will be utilized for the SSH handshake later on, whereas the buffer employed to read files from disk will be cleared and released.

2022. RetBleed [127] hijack the return instructions found in the kernel that lead to gaining speculative code execution in the context of the kernel. The kernel data are leaked by controlling the registers and the memory when the victim executes the return instruction. They found approximately 1000 vulnerable return instructions, of which approximately 60 can be used in a Flush+Reload attack and leak kernel data.

2023. Downfall [128] presents three cross-core covert-channel attacks based on Gather Data Sampling (GSD) using three instructions to transfer data *vmov, rep mov,* and *aes*. The attack against kernel employs two processes: one for *ioctl* calls in user space and

Table 8: Speculation-based attack countermeasures

Technique Name/ Article	Year	System/ Simulator	Mitigated Techniques	Performance Impact/ Overhead	Performance Evaluation Algorithm/ Benchmark	Hardware Overhead
InvisiSpec [130]	2018	Gem5 [105]	Spectre-like attacks	53% performance overhead. Reduce overhead introduced by *fence* defense against Spectre attacks with 21%.	SPECInt2006 and SPECFP2006; Multi-threaded PARSEC;	0.0174-0.0176 mm^2
SpecShield [131]	2019	Gem5 [105]	Meltdown, Foreshadow, Spectre-v1, v2, v4	21% for ERP and 10% for ERP+	SPEC 2006 [122]	-
SafeSpec [132]	2019	MARSS x86	Spectre & Meltdown	3% improvement	SPEC 2017 [133]	2-17%
Conditional Speculation [134]	2019	Gem5 [105]	Spectre attacks	12% for Cache-hit Filter; 6.8% for Cache-hit + TPBuf Filters.	SPEC 2006 [122]	3.5%
SpecCfi [135]	2020	MARSS x86	Spectre BTB, RSB	1.9%	SPEC 2017 [133]	0.1%
Detecting Spectre using HPC [80]	2021	Intel Core i3-3217U	Spectre-IV	-	-	-

another on a sibling CPU thread for GDS. By manipulating the index, the attacker guides the victim into prefetching memory offsets, leaking one dword at a time. The gather value injection (GVI) attack combines GDS with Load Value Injection (LVI), where the attackers seek gather instructions in the victim code followed by data-dependent operations, similar to Spectre gadgets. The technique introduces various GVI gadgets and utilize them to pilfer out-of-bounds data from a victim process. Additionally, the attack recovers the AES-128 and AES-256 keys from AES encryption executed using OpenSSL and leak data from an Intel SGX enclave, stealing the sealing keys.

2.7.2 Speculative and Out-of-order Execution Countermeasures

In Table 8 is the summary of countermeasures for speculation-based attacks, containing the system and the benchmark used for the evaluation, the performance and hardware overhead and the name of the mitigated attack. Next, a description of countermeasures will be presented.

2018. InvisiSpec [130] is a defensive strategy employed in multicore processors to mitigate hardware speculation attacks, rendering speculation undetectable within the data cache hierarchy. It implements a speculative buffer (SB) where unsafe speculative load data is read to avoid modifications in the cache memory, by obtaining the requested cache line. By avoiding the cache, data is invisible to other threats. When the loads become safe, a request to the cache line is made again which will modify the cache and bring the respective data into the cache, making it visible. Two mitigations are tested, implementing an SB for L1 cache and for LLC.

2019. SpecShield [131] protects against Spectre, Meltdown and Foreshadow-like attacks, that leak speculatively executed data. This technique is based on delaying the

instructions until it is no longer speculative to stop the leakage and disrupt the transmitting phase of a covert channel. The delay is done by modifying the timing at which speculative data that are loaded into the pipeline of the CPU can be utilized by dependent instructions. Three designs are implemented, *SpecShieldERP+* having the best performance. Here, speculation can be retrieved only by instructions that are not useful in a covert channel. This will reduce performance overhead by reducing the amount of deferred forwarding of speculative data.

SafeSpec [132] is a countermeasure technique against Spectre and Meltdown attacks that isolate speculations from committing states, protecting caches and TLBs in a single-core system. This method introduces a so called shadow state that acts as a barrier between the speculative state and the initial processor structure, named committed state. A temporary structure will hold the line that needs to be filled up in cache after a speculative execution. In case the speculation is correct, the line will be moved to L1 cache, whereas if it was wrong, the structure will be removed and the vulnerability channel will be closed. There are two possibilities to move back from speculative to committed state. The first one is called wait-for-branch (WFB), where the move is done when all the branches related to an instructions are solved, and the second one is called wait-for-commit (WFC), which waits for the instruction to do all the commits needed. The WFB mitigates all the Spectre attacks, and the WFC mitigates Meltdown.

Conditional Speculation [134] is a hardware-based mitigation against Spectre attacks using a hazard detection module. This module works by marking instructions that access memory and which can affect the security with flags that indicate a suspect speculation. The flag is removed only if the speculative execution of those instructions is not writing data in new cache lines without privileges. To reduce the performance overhead, two buffers are implemented, the Cache-hit Hazard Filter which aim speculative instructions that hit the cache and Trusted Page Buffer which spots the safe speculative instructions. Instructions that get through the filtering process are permitted to undergo aggressive speculation, thereby enhancing performance within the realm of security.

2020. SpecCFI [135] is a defense against Spectre RSB and BTB, that prevent attackers to launch branch target buffer attacks using Control-flow integrity (CFI) solution. CFI prevents against diverting the execution of the victim's program and force the execution to follow only legal paths that are included in a control-flow graph. This defense consists of two parts; one computes the graph, and the second maintains the execution of the expected path.

2021. Hardware Performance Counters (HPC) were used in [80], to detect the Spectre-IV attack. Different events related to caches and branches are monitored during an attack, and the data is introduced in an ML classifier, which detects an attack with an accuracy of more than 90%.

2.8 Power-based attacks and countermeasures

Power side-channel attacks analyze fluctuations in the power consumption of a device to extract confidential information; however, there are a small number of attacks that can be executed without direct physical access to the targeted system.

RAPL. Running Average Power Limit (RAPL) is an interface that reports the energy

Table 9: Power-based attacks

Technique Name	Year	OS/Kernel	System	Targeted Algorithm	Reliability	Possible Countermeasure
Platypus [25]	2021	Ubuntu 16.04 to 20.04; kernels 4.15.0 to 5.4.0	(M) Intel Core i7-6600U Skylake; i3-7100U Kaby Lake; i7-8650U Kaby Lake-R; (D) i5-3230M Ivy Bridge; (D) i7-6700K Skylake-S; (D) i9-9990K Coffee Lake-R; (S) Xeon E3-1240 v5 Skylake; (S) Xeon E3-1275 v5 Skylake; (S) Silver 4214 Cascade Lake;	RSA; AES; KASLR	Full 2048-RSA key; 15/16 bytes of AES key; kernel address space derandomization; Covert channel with transmission rate 18.7 bit/s and error rate of 0.89 %	Restrict access to the RAPL counters; Limit the resolution of the counters; Intel microcode updates.
Red Alert for Power Leakage [26]	2021	Ubuntu 16.04 to 20,04, CentOs 7, macOS Catalina 10,15,3	Intel Core i7-7700, i7-8550U, i5-8500B; Xeon D-1548, Xeon Gold 6130	Chrome, Safari, Tor	Accuracy of 81% for Tor and 99% for the others	Restrict access to the RAPL; Remove the *powercap* interface; Implement fuzzy time.
Prefetch+ Power [116]	2022	Linux kernel 3.10, 5.4, 8.5	AMD EPYC 7402P, 7571; AMD Ryzen 5 3600; AMD Ryzen Threadripper	KASLR	Break KASLR	Restrict access to *amd_energy* driver, limit userspace access.
Collide+ Power [137]	2023	Ubuntu 20.04.5 LTS, Linux kernel 5.4	Intel Core i5-2520M Sandy Bridge; i3-7100T Kaby Lake; i7-10510U, i7-10710U, i7-10510U, i7-10710U Comet Lake; i7-1185G7 Tiger Lake, Core i9-12900K Alder Lake; Core i9-9900K, i7-8700K, i9-9980HK, Xeon E-2176M Coffee Lake; Ryzen 5 2500U Zen, 3550H Zen+; AMD EPYC 7252 Rome; Ryzen 9 5900HX Zen 3	Memory locations	99% success rate	HW mitigations, eliminate prefetcher gadgets in the kernel, limitations in reading performance counters.

consumption across different power domains [136]. It was designed to ensure that the CPU stays in the desired thermal and power limits. The RAPL interface is available on both Intel and AMD processors.

2.8.1 Power-based Attacks

The power-based attacks are described in the following, together with Table 9, where the OS and system used for attacks are mentioned for each technique, together with the targeted algorithm, the reliability of the attack, and possible countermeasures.

2016. The power-based analysis attacks debuted with [138] which could only retrieve the information of taken branches and cached data, on account of the low sampling rate of the RAPL interface.

2021. Later on, Platypus [25] takes advantage of unprivileged access to the RAPL

Table 10: Thermal-based attack countermeasures

Technique Name	Year	OS/Kernel	System	Reliability	Performance Impact/ Overhead	Mittigated Techniques
Seeing the Unseen [139]	2015	Android	Samsung Galaxy S3, LG Optimus 4X HD P880	70-96% accuracy detection	negligible	Covert channels
MAD-EN [140]	2023	Ubuntu 20.04 LTX, kernel 5.11.0-46	Intel i7-10610U	98% accuracy-detection	9%	F+F, F+R, P+P, PortSmach, TLBleed, Spectre PHT, Spectre BTB, Spectre RSB, Spectre STL, BHI, Medusa, ZombieLand, Fallout.

interface, which reveals details associated with power consumption of the CPU, thus creating a side channel. This technique uses statistical analysis, to detect fluctuations in power consumption, technique which allows to differentiate between various instructions and calculate the Hamming weights of the memory loads and operands, which leads to the cryptographic key extraction. The results include recovering RSA and AES keys, breaking the KASLR, and setting up covert channels.

Another paper that presents an attack using RAPL interface is [26], where a covert channel using *AVX* instruction is created, together with a website fingerprinting technique that will discover the web pages visited by an user in different browsers.

2022. The first power-based side-channel attacks on AMD are presented in [116], implementing the Prefetch+Power technique which uses an interface similar to Intel RAPL. It uncovers variations in power consumed by the *prefetch* instruction and successfully break the KASLR.

2023. Collide+Power [137] is a generic attack that analyzes power consumption in the memory hierarchy of a CPU using a model function for leakage. It leaks data from memory locations in MDS or Meltdown styles, together with single-bit differences in the memory with a speed of 4.8 bits/h.

2.8.2 Power-based Countermeasures.

Similar as above, Table 10 shows the summary of power-based countermeasures, containing the system and operating system, the technique's reliability and performance overhead and the mitigated attack.

2015. Covert channels on Android devices were detected using Android APIs to read instant voltage of a smartphone [139]. The authors used neural networks combined with decision trees to detect data transmission in seven covert channels, with an accuracy between 70% and 96%.

2023. RAPL interface can also be used to detect attacks, MAD-EN [140] is a tool that can detect anomalies of the system in real time. In the offline phase, traces are collected during attacks and are used in the training of the anomaly detector, which will be used in

an online phase. It can detect 10 different attacks with an accuracy of 98% and has 69% less overhead introduced, compared to the other detection methods which are based on hardware performance counters.

2.8.3 Other types of attacks.

2018. Checking the security in the green IT sector, [141] shows that the use of energy consumption can be used to attack RSA implementations using *powercap* file system in Linux.

2022. Hertzbleed [142] is another paper that uses power consumption to extract frequency variations on Intel and AMD CPUs to extract full cryptographic keys.

Moreover, frequency throttling is another type of attack [143] based on power management, which can extract the full AES key.

2.9 Thermal attacks and countermeasures

The thermal attacks can be created via side channels or covert channels and they exploit the electronic device's heat emissions to get access to secret information. These attacks can be made by having physical access to the hardware or via software, without physical access. These attacks work based on the idea that the heat which is generated while the CPU is executing or accessing different memory locations is measured by the thermal sensors placed on the CPU. The temperature variations are recorded by the attacker over specific amounts of time, and the thermal traces are used to extract secret data.

2.9.1 Thermal-based attacks

The thermal attacks are described below, together with an overview table (see Table 11) which contains for each attack, the OS and the system used, together with the targeted algorithm, the reliability of the attack and a possible countermeasure.

In [144], the authors used ARV and PIC microcontrollers in a hardware-based side-channel attack, where they exploited the idea that if the temperature of a device exceeds a specific threshold, a fault is impelled. Using this scheme, it was possible to recover the private key of an RSA algorithm.

In [145], the authors propose a hardware-based thermal side-channel attack. Here they come with a big improvement by using Correlation Power Analysis (CPA) techniques modified for thermal traces, together with Deep Learning (DL) techniques. The results consist of full recovery of the RSA key on devices based on AVR.

2015. In the software-based area of attacks, most papers present only covert channels. In this case is [146], which exploits the thermal comportment of multi-core systems based on remnant heat and heat propagation using Intel digital thermal sensors. The authors have created two types of covert channel attacks, using the temporal and spatial partitioning on the x86 systems. To send the data through the covert channel, they use different applications from the MiBench [147] benchmark suite, in an *ON-OFF* scheme. To transmit the value of *1*, the application is run intensively for a period of time, while when sending a value of *0* the application is stopped (is in idle state) for the same period of time.

Table 11: Thermal-based attacks

Technique Name	Year	OS/Kernel	System	Targeted Algorithm	Reliability	Possible Countermeasure
Thermal CC on multi-core platforms [146]	2015	Suse Linux	(S) Intel Xeon	-	Throughput of 12.5 bps with an error rate of up to 10%	Restrict access to thermal sensors
On the capacity of thermal CC in multi-cores [148]	2016	Ubuntu 14.04.2	(L) Intel Core i7-4710MQ; (M) Samsung Exynos 5422 with Cortex-A7 and A15; with frequency, fan speed set to maximum	-	Throughput of 45 bps on the same-core channel and 5 bps on the 1-hop channel, with less than 1% error probability	Restrict access to thermal sensors; Increase refreshing time and reduce sensor resolution
Improving efficiency of TCC in multi/many-core systems [149]	2018	-	Many-core C++-based simulator, with DSENT & Hotspot for power model and temperature	-	Throughput of 160 bps and error rate 0%, in a chip with the transmission rate stable	-
Thermal AttackNet [150]	2021	UbuntuMate v14.04, K 3.10	Samsung Exynos 5422	AES-256	100% prediction accuracy	Mask peak temp. during encryptions
ThermalBleed [27]	2022	Ubuntu 18.04, kernel v 4-5; KPTI enabled	(L) Intel Core i7-7200U Kaby Lake; i7-10510U Comet Lake; (D) Intel Core i7-6700K Sky Lake; i5-7400 Kaby Lake; i7-8700, i5-9600K Coffe Lake; i7-10700, i9-10900 Comet Lake; (S) Intel Xeon E3-1270 v6 Kaby Lake, Silver 4210 Cascade Lake;	KASLR; RSA	Break KASLR with 100% accuracy on Intel Core i9-10900 and Xeon E3-1279 v6, and with 88% and 76% accuracy on Intel Core i7-7200U and i7-10510U; Not able to distinguish RSA instructions during decryption	Restrict access to thermal sensors
TooHot ToHandle [151]	2024	Linux kernel v 5.4.0	Intel Core i7-10th gen, i5 9th gen with RAPL filtering enabled	CNNs; Chrome and Tor browsers	Recognize CNN families; Thermal footprint various websites using Chrome in Incognito mode; Distinguish between Shallow Web and Deep Web on Tor.	-
ThermalScope [152]	2024	Ubuntu 20.04.5, 22.04.1; Kernel v 5.15.0, 5.8.0	Intel Core i5-8250U, i7-9750H, i7-14700K	DNN model architectures; KASLR	Throughput of CC 0.1 bit/s with BER=0.2%; Fingerprint DNN models with accuracy of 90% Break KASLR	-

The heat of the running core will be propagated to the nearby cores, from where it is collected and analyzed. The 1-hop (reading the temperature from the nearby core) covert channel implemented by them has a throughput of 1.33 bps (bits per second) with 11%

error rate in spatially partitioned systems and a throughput of up to 12.5 bps in temporally partitioned systems.

2016. A study of the capacity of thermal covert channels in the multi-core systems is presented in [148], together with an improvement of the communication scheme proposed in [146]. The authors improve the *ON-OFF* scheme to a *Encoding-Decoding* one, which transmits and encodes text in order to avoid the average temperature from wandering up and down. The throughput of this scheme is greater than 5 bps for a 1-hop channel, with an error probability of less than 1%, higher than the one above.

A further improvement of the above techniques is presented in [149], where, compared to [148], the error rate is reduced by 30% on average, and the throughput increased by 50%. Additionally, they overcome the thermal interference by proposing a technique that selects the transmission frequency and makes it able to create multiple transmitters to one receiver covert channel. The attack they propose can reduce the error rate by 75% and can improve the transmission throughput with 370%, leading to a transmission rate on a stable channel of 160 bps and a low error rate of 0%, on a simulator.

2019. In [153], the authors explore the principles of the Bluetooth Low Energy protocol and combine it with a huge load of CPU computations to create a thermal covert channel between two devices, reaching throughput similar to other works.

2021. ThermalAttackNet [150] is a visual-based CNN model used to create thermal side-channel attacks on mobile devices that have multiprocessors systems-on-chip (MPSoC). The most used passwords by users were employed to encrypt a text file with AES-256 encryption, and the temperature of the CPU was recorded during this operation. Several pre-trained popular models were used for the CNN, together with different Linux governors. However, to reduce the power and memory consumption ThermalAttackNet was redesigned, having similar performance with the popular existing CNNs. It is the first CNN that uses graphic representations of the thermal traces to create a side-channel attack and determine the passwords, with a prediction accuracy of 100%.

2022. Thermal Covert Channels were used also in 3-D multi-core processors in [154], with a throughput of 200 bps using the SPLASH-2 [155] benchmark.

ThermalBleed [27] is the first thermal software-based side-channel attack, which overcomes the limitation of covert channels. It uses Intel's digital sensors to establish the temperature of the processor in various conditions. The authors studied the sensors in different situations and were able to differentiate between cache hits and accesses to the physical memory. Moreover, they were able to identify TLB address translations. Additionally, they constructed a side-channel attack in which they were able to break the Kernel Address Space Layout Randomization (KASLR) by repeatedly executing *prefetch* instructions on the kernel base addresses. Simple thermal analysis was used to differentiate between a hit or a miss in the dTLB.

2024. TooHotToHandle [151] establishes the connection among power consumption and thermal dissipation to create a side-channel attack and avoid the countermeasures of RAPL filtering against power side-channel attacks. Using the thermal information of the CPU, the authors were able to distinguish between different CNN models. Additionally, using thermal traces, they were able to extract the history of websites accessed from a Chrome browser used in Incognito mode and distinguish between the Shallow Web and Dark Web in Tor browser.

ThermalScope [152] is another side-channel attack based on thermal emissions. It is composed of three attacks, based on event interrupts, which are correlated with different processes running on the CPU, creating side-channels. A covert channel with a throughput of 0.1 b/s and a Bit Error Rate (BER) of 0.2% is implemented, together with fingerprinting different DNN model architectures, and breaking the KASLR in 8.2 hours.

2.9.2 Thermal-based countermeasures

Similar, the overview of the thermal-based countermeasures are presented in Table 12, containing the system, reliability and performance overhead for each technique, preceded by a short description for each technique.

2016. In [156] a technique that camouflages the activities in the CPU in order to prevent against side-channel attacks is presented. It uses on-chip controllers that will track odd activity patterns and generate patterns with an algorithm for dynamic shielding to mask the activity and inject noise. It can reduce the Side-channel Vulnerability Factor (SVF) for side-channels under 0.05 and the Spatial Thermal Side-channel Factor (STSF) under 0.59.

2020. In [157] a countermeasure against Thermal Covert Channel (TCC) attacks using Dynamic Voltage Frequency Scaling (DVFS) is detailed. It can detect 98% of attacks and increase the BER to 92%. The three steps of the technique include a detection part, where the temperature of each core will be recorded using a specific frequency for sampling; a selecting part, where the core with the highest signal amplitude will be selected as suspicious; and a blocking part, where the frequency of the suspicious core is altered by scaling it down, which will change the temperature of the core and increase the BER.

2021. In [158] a countermeasure against TCC attacks using task migration is outlined. First, it detects the cores that are affected by the attack by scanning the frequency. Next, the receiver and transmitter are migrated to cores as far away as possible, to decrease the thermal correlation between them. The packet error rate (PER) is increased to 84% and the power consumption and execution time are reduced compared to existing countermeasures.

In [159], a scheme that included three steps against thermal covert channels was introduced. In the first step, to identify a possible attack, they analyze the workloads of the CPU in terms of frequency. Then they identify the cores on which the thermal covert channel is mounted. Finlay, they use the Dynamic Voltage Frequency Scaling technique to block the attack on the respective cores. With this scheme, they were able to detect 97% of the thermal covert channel attacks and increase their bit error rate (BER) with 70%, in this way making them useless, with a performance loss of only 3%.

2022. A countermeasure scheme for adding noise in the thermal behavior of the processes is presented in [160]. It is based on increasing the duration while the transmitter sends a bit of 1, which will change the thermal pattern of the transmission, and the real value will not be discovered by the receiver. Using this technique, the BER of a covert channel attack is increased to 94% with a low addition in the power consumption.

Another countermeasure against TTC attacks is [161], which can detect an attack with an accuracy of 89% and stop it by increasing the BER to 50%. It uses frequency scanning to detect possible attacks and applies intense noise to the frequency band to

block transmission.

Table 12: Thermal-based attack countermeasures

Technique Name	Year	System	Reliability	Performance Impact/ Overhead
TASCS [156]	2016	GEM5 [105] with McPAT and Hotspot	Reduce: Side-channel Vulnerability Factor under 0.05; Spatial Thermal Side-channel Factor under 0.59	Power overhead of 5.7%
On countermeasures against the TCCA targeting many-core systems [157]	2020	Sniper with McPAT and Hotspot	Detect 98% of the TCC attacks; Increase BER to 92%.	Performance penalty of 3%
Defending against TCCA by task migration in many-core systems [158]	2021	Event-driven C++ simulation platform with DSENT and Hotspot	Increase PER to 84%.	Reduce power consumption introduced by other counter measures by 57.3% and execution time by 68.5%.
Countermeasure against TCCA in Many-core Systems [159]	2021	Sniper-v7.2 with McPAT-v1.0 and Hotspot-v6.0	Detect 97% of the attacks and increase error rate of the attack with 70%.	3%
Selective Noise Based Countermeasure against TCCA [160]	2022	Sniper-v7.2 with McPAT-v1.0	Increase BER to 94%	Power consumption did not increase significantly
Combating Stealthy TCC Thermal Signal Transmitted in Direct Sequence Spread Spectrum [161]	2022	Sniper with McPAT and Hotspot-v6.0	Detection accuracy of a TCC is 89% and increase BER to 50%	Energy consumption overhead of 0.32%
Detection of TCC Based on Classification of Components of the Thermal Signal Features [162]	2022	Sniper-v7.2 with McPAT-v1.0 and Hotspot-v6.0; Intel Core i7-7700HQ, i7-6700U	Detection accuracy is 99%.	Runtime overhead 0.187% and energy overhead 0.072%
Smart Detection of Obfuscated TCCA in Many-core [163]	2023	HotSniper with McPAT and HotSpot	Detection accuracy is 99%.	Reduce the overhead of the existing countermeasures by 14 times.
Hot-n-Cold [164]	2023	Intel Xeon E5-2640 v4	Detection correlation of 0.8	-
Beat the Heat [165]	2024	Intel Core i7-10700F, i5-4590; Intel Xeon W3550	Detection correlation 0.96; 88% detection accuracy using ML	-

A detection method is presented in [162], which uses a neural network to classify the TCC signals. The inputs contain a vector with amplitude signals of the TCC or noise and the output is one of the values *0* or *1*, which specifies if the signal is noise or is part of a covert channel. Its accuracy is 99% with an overhead of 0.187% for runtime and 0.072% for energy.

2023. Dotecca [163] is a detection technique that can detect attacks with a precision of 99%. It uses time-domain windows of measurements and every core is analyzing its

own sensors to detect a possible attack and its location. This mechanism of detection at the core level helps to diminish the overhead brought by the countermeasures using the Fourier Transform by 14 times.

2.10 System call attack countermeasures

System call monitoring is an effective approach to study attacks because they produce traces that are not created in normal circumstances [166]. To achieve efficient anomaly detection, it is essential to track a feature that remains stable during normal, authorized activities while it is affected by attacks.

Most studies monitor system calls by utilizing a combination of static and dynamic analyses. Dynamic analysis involves extracting system calls from an executable during its real-time execution. While effective, this technique has a significant disadvantage: it cannot identify all system calls because not all execution paths are covered during runtime, as it only captures the system calls made during the specific execution scenarios that are observed. To address this limitation, static analysis is employed. Static analysis examines the entire codebase without executing it, ensuring that all potential execution paths are covered. This comprehensive approach can identify all possible system calls within the program. However, static analysis comes with its own drawback: it tends to generate false positives, identifying system calls that might not actually be executed in a real-world scenario, and it is time consuming. Dynamic analysis cannot retrieve the full list of system calls due to the nested libraries calls [167]. Despite this, the combination of both static and dynamic analyses provides a more thorough and accurate tracking of system calls than either method used alone [168] [169].

Sysverify [168] uses an approach that merges static and dynamic analysis to minimize the attack area in cloud docker containers. The static analysis generates a connection path from the system calls to the library API, whereas the dynamic analysis creates a connection path for syscalls resulting from indirect calls or for the syscalls that are seldom used. The combination of the two connection paths will show information about the security level of the syscall's generation. Sifter [170] constructs security filters for kernel modules in Android platforms, which are used to reject syscall patterns that are strange, taking in consideration the order of execution, the arguments, and their length or if the system call is out of use. The Tailor toolkit [169] creates a system call whitelist for the Seccomp filter. The whitelist is created by combining two partial whitelists. The first one is created using three profilers which map the system calls originating either from functions, from the environment, or libraries. The second one is created by tracking system calls coming from assembly instructions or syscall functions, using static analysis. However, a fully automated machine learning-driven framework for detecting cryptomining activities is presented in [171], which utilizes syscall patterns to identify anomalous events in comparison to the normal algorithms used in cryptocurrency operations. Other filtering techniques are SysXCHG [172] which can adapt during execution, Nimos [173] which explores syscall sequences, and Optimus [174] which identifies the syscalls which are essential for containers, increasing their security.

2.11 Discussion about the literature review

In this subchapter, it will be presented some trends in attacks based on the papers selected for the literature review.

Taking into consideration all the processors used in the aforementioned attacks, one can see, based on Figure 5, that Intel is by far the most used CPU, regardless of the category in which the attack falls. The next CPU used is AMD, closely followed by the ARM-based CPU and other platforms. Apple is the least attacked, maybe because its architecture is not public, reverse engineering is not practical, and the number of units on the market, other than personal use, is low.

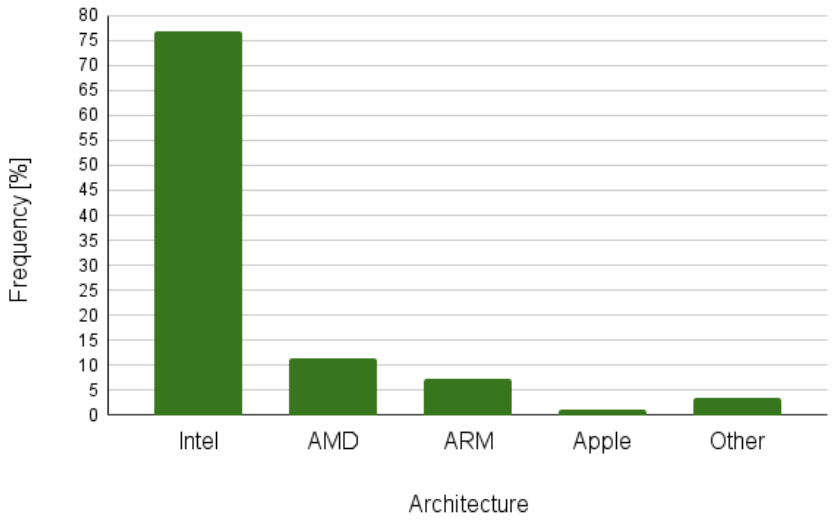


Figure 5: Attacked CPU architectures

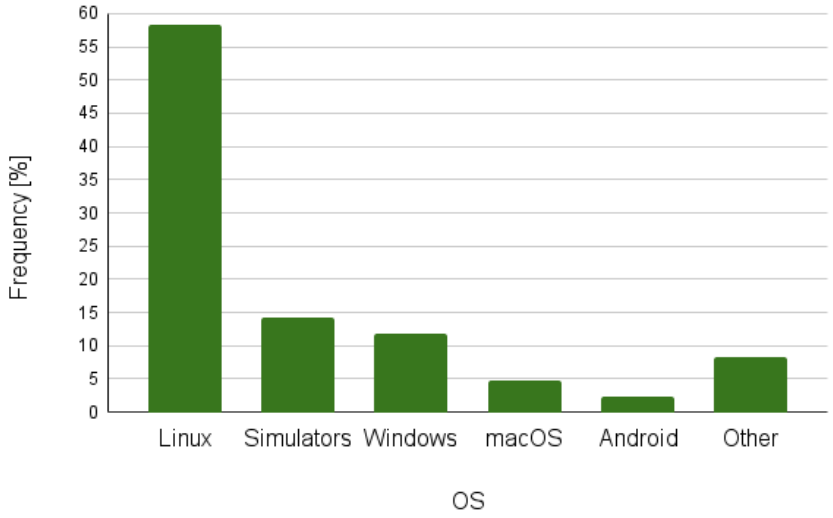


Figure 6: OSs used in attacks

If one looks at the most used Operation Systems (OSs) in attacks, from Figure 6 it can be seen that the most attacks are mounted on Linux.

Taking into account the category of cache attacks, the technique most used to attack is Flush+Reload, followed by Prime+Probe, as can be seen in Figure 7. In course of time, as chronologically presented in Table 1, one can see that more and more attack techniques appear, however, F+R and P+P are by far the most used in the actual attacks, due to their practicality and the extent of information provided.

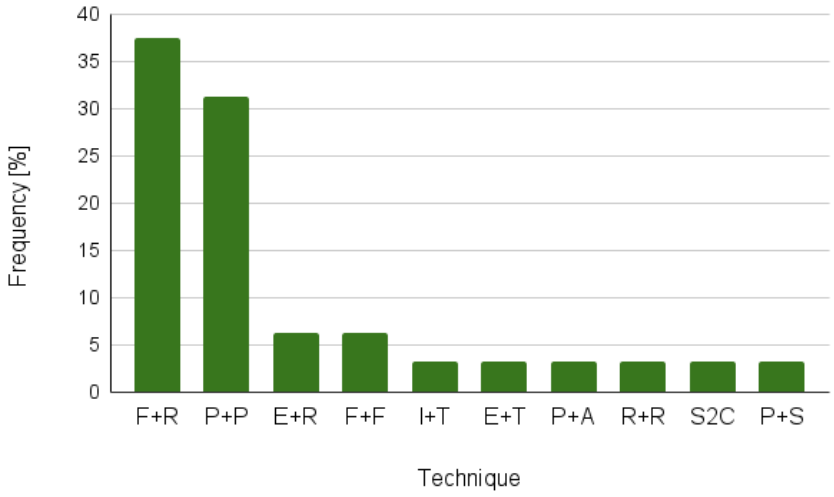


Figure 7: Techniques used in attacks

Taking in consideration the encryption algorithm or feature most attacked, in Figure 8 one can see that AES is the most used, followed by RSA.

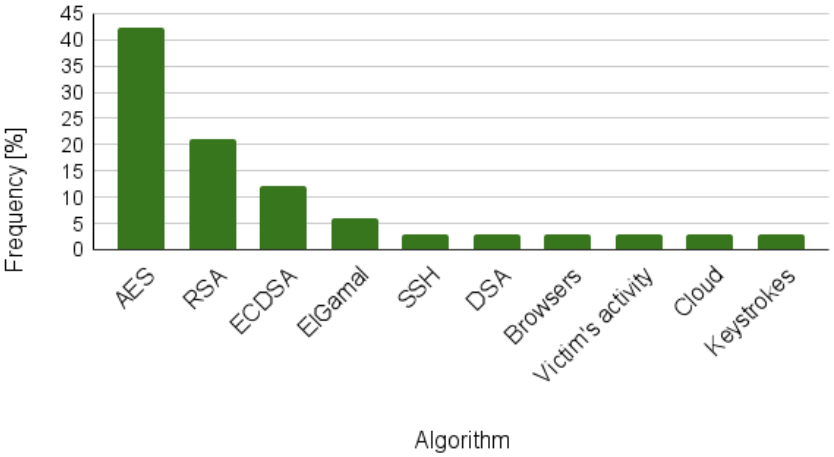


Figure 8: Attacked algorithms

In case of countermeasures for cache attacks, a large part of countermeasures consist of detection techniques, followed by actual mitigations such as logical or physical partitioning or isolation and randomization techniques, as can be seen in Figure 9.

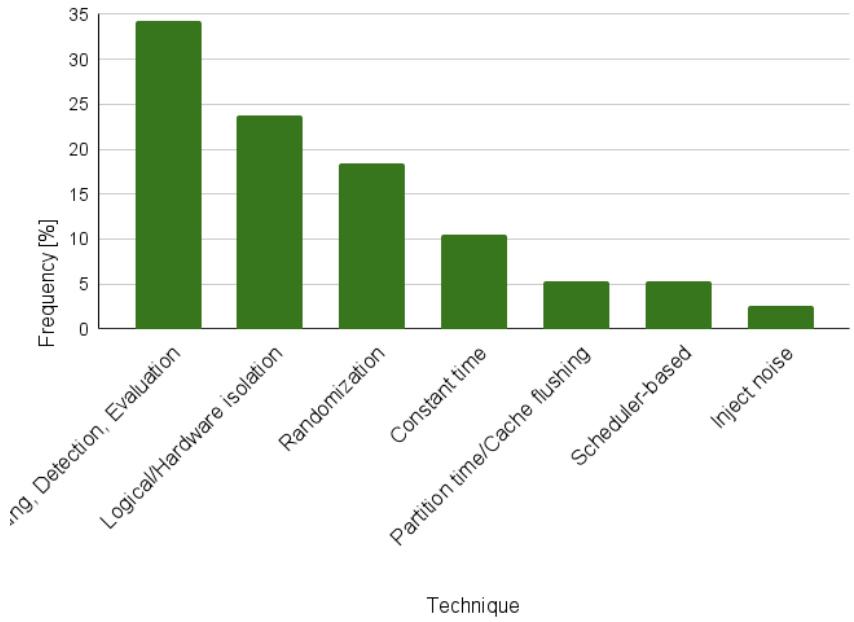


Figure 9: Cache countermeasures

The performance overhead brought about by countermeasure techniques in the case of TLB, Prefetcher, Speculation, Power, and Thermal attacks is shown in Figure 10, where with the red color the maximum performance overhead brought about by the techniques and with the green the minimum. Based on the chart, the Speculation attacks are the most difficult to detect and mitigate. However, in the Prefetcher attacks, the countermeasures managed to boost the performance of the system in addition to stopping the attack.

Based on my studies, I have observed that the first attacks that appeared were those aiming caches starting with 2014, when important attack techniques were created. In terms of countermeasures for cache attacks, many mitigation techniques were presented until 2019, however, after this year the number of detection techniques has increased, whereas the number of mitigation methods has decreased.

The techniques proposed for cache attacks created the basis for speculation attacks that started in 2018, where the Flush+Reload technique was widely used. For these attacks, many software countermeasures were added in the Intel processors.

The first attacks on TLB and prefetcher were presented only with 2017 and 2016, respectively, and compared to cache attacks, they are more complicated and their number is much smaller.

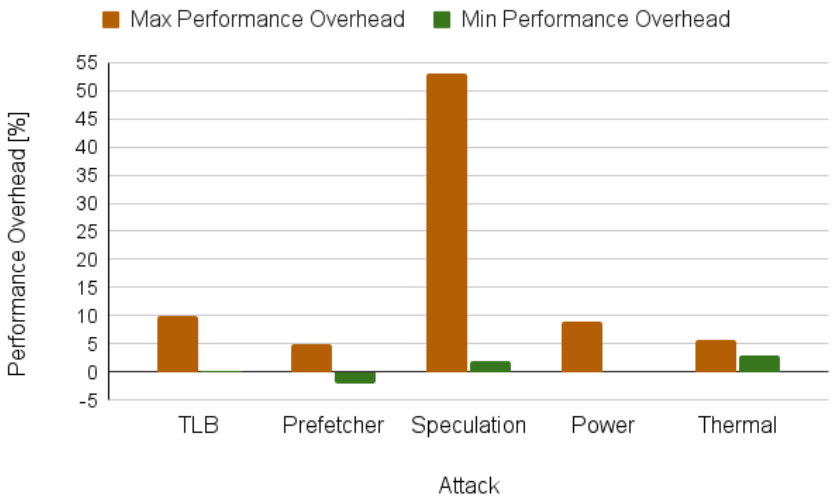


Figure 10: Performance overhead in countermeasures

The first power-based attack was published in 2016, and the next ones only in 2021, due to the low resolution of the RAPL interface; however, there is one countermeasure that can mitigate a huge number of attacks, of different types.

Thermal-based attacks and countermeasures started in 2014 and 2016, respectively; however, the majority of them are used to attack a covert channel method and mitigations are also created for covert channels. Nevertheless, techniques based on side-channels have appeared in recent years.

The key contributions in this chapter are as follows.

- An extensive survey of the most cited software-based attacks and their countermeasures in the last decade is presented.
- A classification of attacks and countermeasures is presented based on the target component and attack mechanisms.
- A comparison of attacks based on attack methodology, targeted algorithms, victim system environment, reliability, performance overhead, and mitigation techniques is presented.
- Comparison countermeasures are presented based on their effectiveness, performance overhead, and applicability to various attack scenarios.
- The emerging trends, open challenges, and future research directions are discussed.

2.12 Original contributions

As part of this research, this extensive survey has been disseminated through a peer-reviewed publication. In particular, the article ***A Decade in Software-Based Side and Covert Channel Attacks and Countermeasures: A Survey*** [175] was published in

IEEE Access Volume 13, 2025, where the most cited software-based attacks and their countermeasures were reviewed, taking into account the last decade.

One major class of security threats involves software-based attacks, divided into side-channel attacks, which exploit unintended information leakage, and covert-channel attacks, which establish unauthorized communication pathways to bypass security policies. This survey provides a comprehensive review of the most cited attacks of these types between 2014 and 2024, along with their associated countermeasures. Attacks and countermeasures are classified based on the target system component and fundamental attack mechanisms. In addition, a comparative analysis of the attacks is conducted, considering key aspects such as attack methodology, targeted algorithms, victim system environment, reliability, performance impact, and mitigation strategies. In addition, defense mechanisms are examined, assessing their effectiveness, computational overhead, and applicability in various attack scenarios. Finally, emerging trends in attack techniques and countermeasures were discussed, highlighting open challenges and future research directions to secure modern computing systems.

The analysis of the surveyed attacks reveals several clear trends regarding the targets, techniques, and countermeasures used in microarchitectural side-channel research.

From the perspective of hardware platforms, Intel CPUs dominate as the primary target across all categories of attacks. This is due to Intel's widespread market share and the openness of its architecture compared to more closed platforms. AMD and ARM processors follow at a distance, whereas Apple processors remain the least attacked, largely due to their closed design, limited public documentation, and smaller market penetration outside personal devices.

Operating systems show a similar concentration: Linux emerges as the most used OS, which reflects both its dominance in research environments and its popularity in server and cloud deployments. This makes it a natural platform for proof-of-concept attacks and large-scale exploitation attempts.

With respect to attack techniques, cache attacks remain central. The Flush+Reload (F+R) and Prime+Probe (P+P) approaches are consistently the most exploited, thanks to their relative simplicity and the depth of information they can leak. However, over time, there has been a steady diversification in attack methods, with new techniques appearing regularly. Despite this innovation, F+R and P+P continue to underpin many practical attacks, demonstrating their robustness and adaptability.

Looking at cryptographic targets, AES is the algorithm that is the most frequently attacked, followed by RSA. This reflects the widespread use of AES in modern systems and its heavy reliance on memory access patterns, making it particularly vulnerable to cache-based leakage.

Countermeasures have followed their own trajectory. Detection methods have gained prominence, especially after 2019, while the development of direct mitigation strategies such as partitioning, isolation, and randomization has slowed. This suggests a shift in focus toward monitoring and reactive defenses rather than preventive architectural redesigns. Among specific attack classes, speculative execution vulnerabilities have proven the most challenging to defend, requiring extensive software patches on Intel processors. Interestingly, in the case of prefetcher attacks, certain countermeasures not only blocked attacks but also improved system performance, which is a rare dual benefit.

Chronologically, cache attacks appeared first (around 2014), setting the stage for subsequent attack families. Their evolution directly informed the later rise of speculative execution attacks (from 2018 onward), which leveraged similar primitives such as Flush+Reload. TLB and prefetcher attacks appeared later (2016–2017) and remain fewer in number due to their greater complexity. Power-based attacks surfaced in 2016 but stalled until 2021, primarily because of limitations in measurement resolution; nevertheless, a single countermeasure in this domain has proven highly effective across multiple attack types. Thermal-based attacks and defenses emerged between 2014 and 2016, initially focusing on covert channels, but more recently shifting toward side-channel exploitation.

It was observed that these trends show a clear trajectory: early emphasis on cache attacks, followed by diversification into speculation, TLB, prefetcher, power, and thermal vectors. Over time, defenses have shifted away from broad mitigations toward detection, while attackers continue to refine foundational techniques such as Flush+Reload and Prime+Probe. The result is an ongoing arms race, with Intel platforms, Linux systems, and AES encryption standing out as the primary battlegrounds.

CHAPTER 3

3 The Security of Parallel Computing

3.1 Introduction

Digital security risks threaten our lives every day, especially when they appear in High Performance Computing (HPC), which are used in aerospace industries, healthcare and manufacturing [176]. Unfortunately, in these systems, the performance is imperative, which leads to little interest in security that could possibly decrease the performance. Therefore, the challenge is to increase security without decreasing performance.

This chapter introduces Hot-n-Cold, a novel technique that can be used to detect anomalies in the behavior of Linux commands through dynamic analysis, using the Intel CPU Digital Thermal Sensors. Hot-and-Cold is used to map the attack surface introduced by syscalls, by detecting possible anomalies that can be used by an attacker. Such anomalies are detected by a correlation between the temperature of an original Linux command and an augmented with risky syscalls command, which can be applied in real-time. The results show a high correlation of up to 0.8 between the original Linux command and the version enhanced with syscalls, with a temperature difference of approximately 0.5 ℃, results that indicate a clear detection of anomalies.

One user process can invoke all of the available system calls while running different programs. If one of those processes accesses a compromised system, it could lead to a break in the kernel space, which can lead to gaining access to unauthorized data, escalating privileges, crashing the system, leaking kernel internal information, and so on. The theft of valuable and confidential information is a consequence of all the aforementioned outcomes. Security in High Performance Computing (HPC) has been largely neglected in the name of high performance [177] [178]. However, HPC systems, being shared resources among many users, may execute workloads that process sensitive data. Hence, it is imperative that security is increased [177] [178] [179] and attacks are detected and mitigated.

The proposed Hot-n-Cold approach is the first to my knowledge in the beginning of 2025, to utilize CPU temperature to identify the attack surface created by system calls on High Performance Computers. The Intel CPU Digital Thermal Sensors are normally used for thermal management and can be accessed without privileges through the *hwmon* Linux interface. Hence, I use the *coretemp* kernel module [180] to measure the core temperature during the execution of original Linux commands and commands augmented with syscalls.

The augmented commands, access system calls that have been reported to be the most attacked in Common Vulnerabilities and Exposures (CVE) [181]. The correlation between the vector of temperatures of the original commands and the vector of temperatures of the augmented commands is very high, up to 0.83 using the Pearson correlation index. This approach can be extended and can be employed during runtime to automatically identify any abnormalities in the execution process. The extension would imply transform-

ing the correlation calculated at the final of the experiments into an automatic calculation, which is run together with reading temperature and recalculated periodically.

3.2 Methodology

3.2.1 The system

The tests were run on the High Performance Computer (miniHPC) [1] at the University of Basel, with a peak of 28.9 double precision TFLOP/s. The miniHPC includes 22 Intel Xeon and 4 Intel Xeon KNL computing nodes, one login, and one storage node. Each Intel Xeon node has two sockets, each with an Intel Xeon E5-2640 v4 CPU with 10 cores, operating at a CPU frequency of 2.4 GHz. Each CPU has 64 GB of RAM and an L3 cache of 25MB. The miniHPC system runs CentOS version 7.9. It is housed in a room where the air conditioning system is set to 23 ℃.

3.2.2 Thermal sensors

Intel CPU have Digital Temperature Sensors on each core, which are located on the CPU die. The thermal management of the system is very important in avoiding hardware damages. In case a critical temperature is detected, an immediate shutdown of the system will take place [182]. Therefore, these sensors are used for thermal management [182]. Figure 11 shows the CPU package and the organization of its components [183].

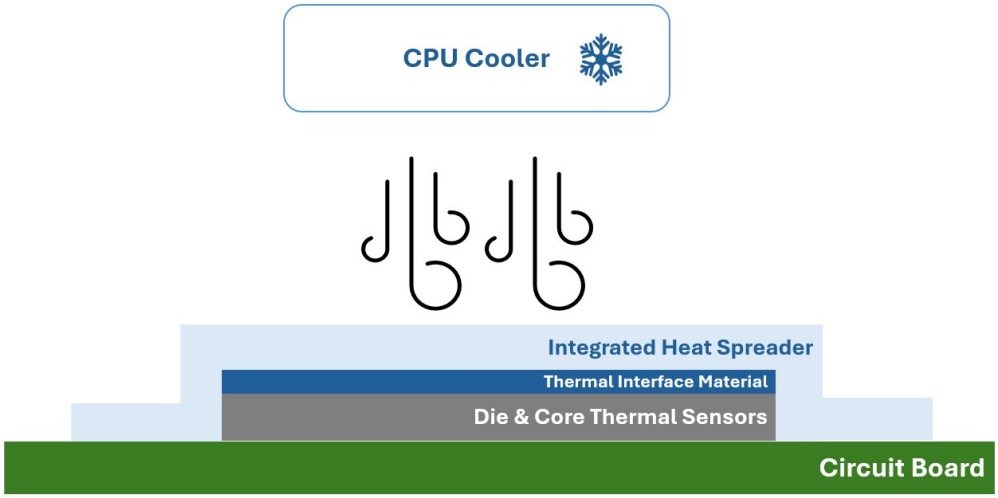


Figure 11: CPU package.

The Thermal Interface Material (TIM) is a substance that is placed between two components to improve their thermal conductivity. Thermal resistance is present at every interface that obstructs heat dissipation. Furthermore, the electronic efficiency and lifespan of the device may significantly deteriorate in case of overheating and substantial thermal restrictions at the interfaces. As a result, there has been a strong focus over the past several decades on developing different types of TIM to improve the overall performance

of thermal management and to reduce the thermal resistance at the boundaries. An Integrated Heat Spreader (IHS) is a metal surface with high conductivity, used to make contact between a heatsink and a CPU and to protect it from external risks. The CPU cooler is used to cool the two above components and the CPU die.

Intel processors provide a set of on-die digital thermal sensors which, together with an interrupt mechanism, let the OS be in charge of the thermal management. Each core has its digital sensor, which can be accessed using a model-specific register (MSR). The position of the sensors, which are located close to the hot spots on the die, has the advantage of being able to accurately read the temperature of the respective core [182]. For each core, the temperature is read from a place near the transistor junctions [184], [185], as seen in Figure 12.

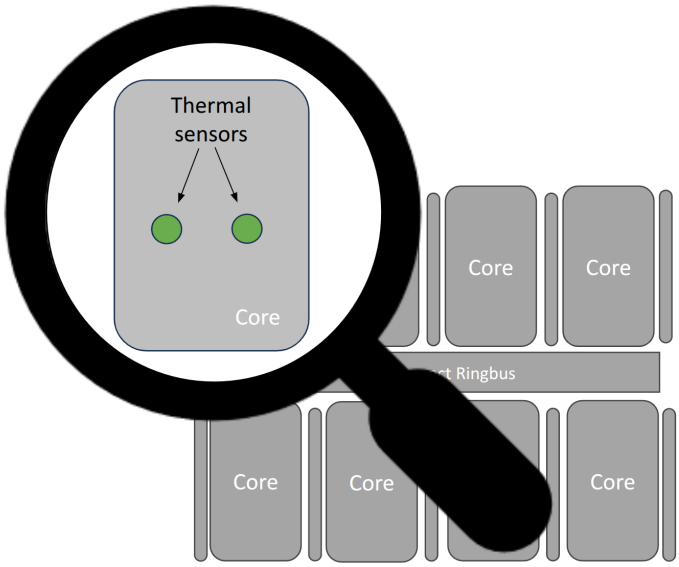


Figure 12: Layout of Intel Core i7-10700's architecture and sensors.

If a particular core exceeds the allowed temperature limit, the protection mechanism will initiate a shutdown of the CPU to prevent any harm to the silicon and allow it to cool down. The maximum temperature allowed in the IHS processor for Intel Xeon E5-2640 v4 is 76℃ [186].

3.2.3 OS system calls

For security reasons, virtual memory is divided into two spaces, the kernel space that requires specific privileges and the user space where unprivileged functions can be used [187]. During operation, a system call changes the control between the two spaces mentioned above [169]. Syscall (short for system call) is an interface between the operating system and a process, containing a mechanism that allows the program to request a service from the kernel of the operating system [188], connecting a user application with the Linux kernel.

In the Linux kernel v5.11, there are approximately 450 system calls [189], while Windows has thousands [187]. One can use a syscall in three ways: via assembly code, calling the syscall directly, or via wrapper functions in *glibc* or other libraries [167]. The first two methods switch between user and kernel spaces multiple times.

For example, to print a word, the context will be switched for every printed letter. The second method will fill a buffer and will change context only once [169]. Multiple switches between kernel and user spaces can introduce security risks.

3.2.4 Linux commands

Linux terminal commands are text-based instructions that are used to interact with the operating system through the Command Line Interface (CLI). These commands allow users to perform a wide range of tasks such as managing files and directories, installing software, configuring the system, and monitoring performance.

One of the most widely used methods to invoke system calls is the use of commonly used and essential Linux terminal commands, which are indispensable for nearly all HPC system users. Due to the vast number of available commands, a list of commonly used commands that are applicable in real-world scenarios was created. Thus, the list of Linux commands used in September 2022 on a miniHPC [1] (a small research cluster at the University of Basel) was analyzed (see Figure 13). It was found that *ls* is the most used command, executed 10000 times during that month.

From the full list of Linux commands used on miniHPC [1], the top 60 commands were selected to reduce the experimental space. To continue the study based on syscalls, it was necessary to identify the system calls generated by the commands. For each of these command the number of syscalls done was retrieved, as seen in Figure 14.

The *strace* utility was used to gain information about the system calls in each command. This utility allows to observe and trace the system calls and libraries in any program, to diagnose any problems therein. An example of its usage and output can be seen in Figure 15.

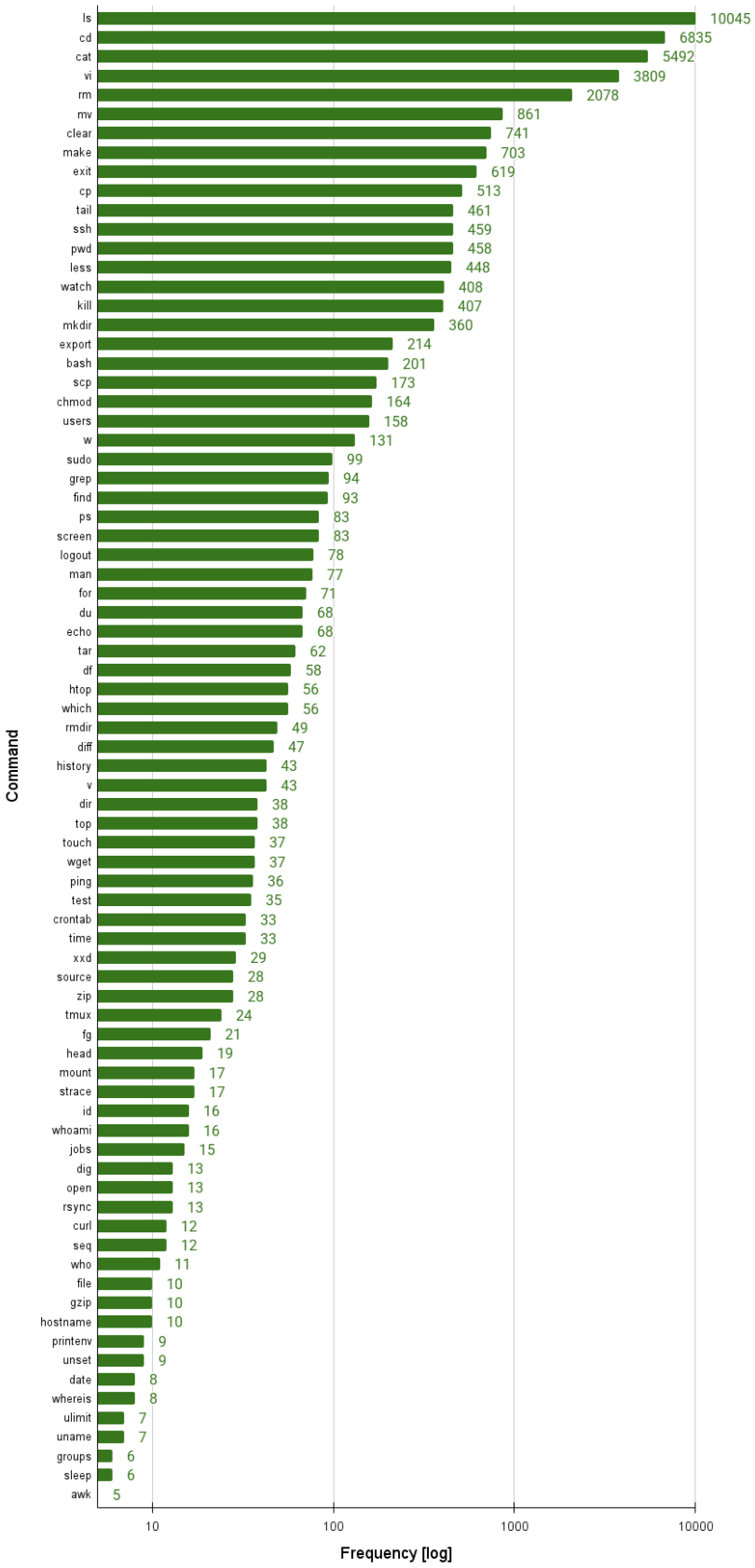


Figure 13: Frequency of Linux commands on miniHPC [1].

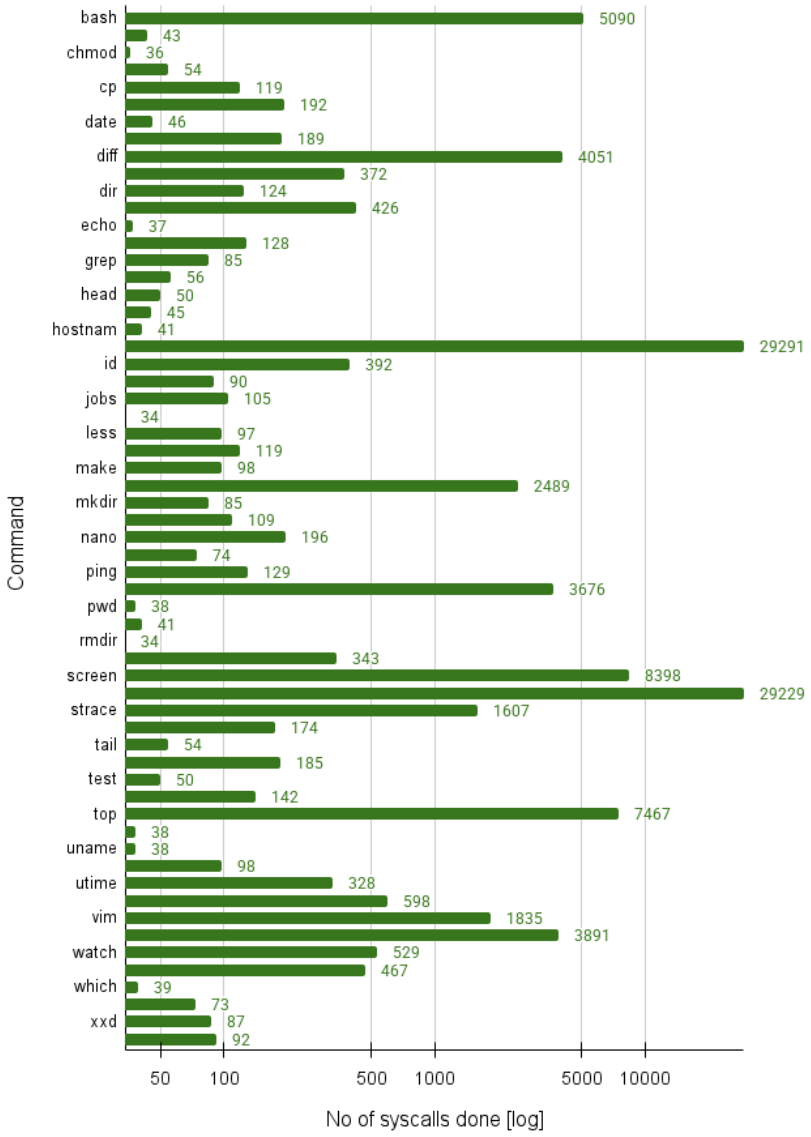


Figure 14: Number of the syscalls done by each command.

```
teo@teo-Yoga-Pro-7-14IRH8:~$ strace ls
execve("/usr/bin/ls", ["ls"], 0x7fff0a80a6c0 /* 62 vars */) = 0
brk(NULL)                               = 0x5896232a0000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ef2b8330000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/local/cuda/lib64/glibc-hwcaps/x86-64-v3/libselinux.so.1", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
newfstatat(AT_FDCWD, "/usr/local/cuda/lib64/glibc-hwcaps/x86-64-v3/", 0x7ffc02002310, 0) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/local/cuda/lib64/glibc-hwcaps/x86-64-v2/libselinux.so.1", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
newfstatat(AT_FDCWD, "/usr/local/cuda/lib64/glibc-hwcaps/x86-64-v2/", 0x7ffc02002310, 0) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/local/cuda/lib64/libselinux.so.1", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
newfstatat(AT_FDCWD, "/usr/local/cuda/lib64/", 0x7ffc02002310, 0) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "glibc-hwcaps/x86-64-v3/libselinux.so.1", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "glibc-hwcaps/x86-64-v2/libselinux.so.1", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "libselinux.so.1", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=85371, ...}) = 0
mmap(NULL, 85371, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ef2b831b000
close(3)                                = 0
```

Figure 15: Example of output of *strace* command.

Further on, the frequency of the syscalls in these 60 commands was computed to see

which one is the most used, see Figure 16.

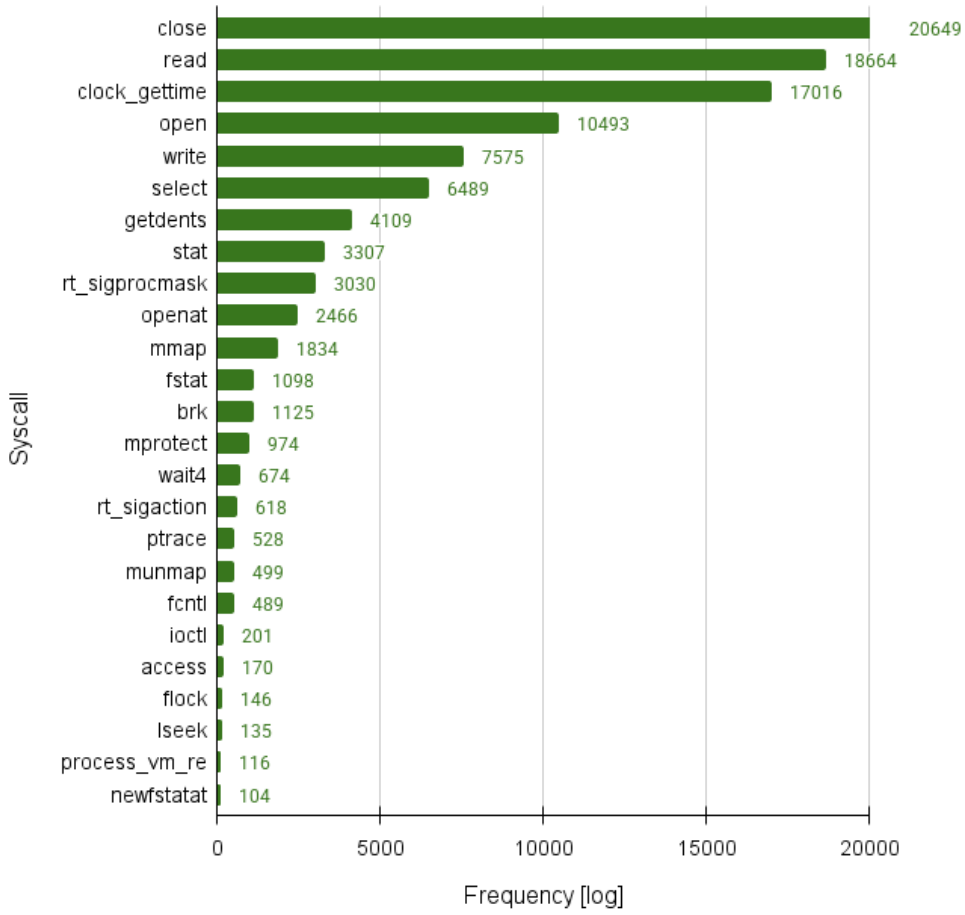


Figure 16: Frequency of the syscalls in the commands.

To simplify the analysis, the system calls parameters were discarded.

Moreover, the number of syscalls done by reading the thermal sensors was calculated. While reading the thermal sensors a number of 429 of system calls were executed, and their frequency can be seen in Figure 17.

3.2.5 Syscalls in CVEs

The Common Vulnerabilities and Exposures [181] database contains a comprehensive record of cybersecurity vulnerabilities that have been reported over the years, including a wide range of types. Applying a filter on this list, one can find all security breaches including the respective words, together with their description and impact. The selected CVEs were filtered to include words like *syscall*, *system call* and *attack*, and 101 vulnerabilities reported between 2016 and 2022 were selected, that include attacks made on, or using system calls.

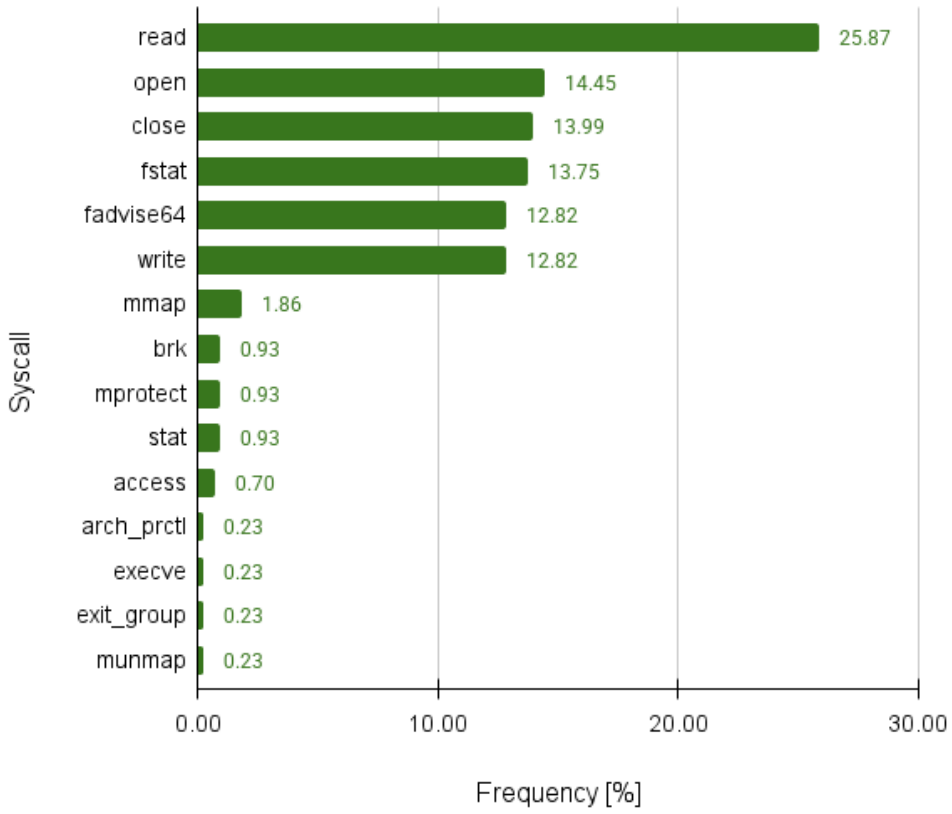


Figure 17: Frequency of the syscalls while reading the thermal sensors.

Table 13: Syscalls found in CVEs.

Syscalls in CVSs	**Frequency**
combination of syscalls	23
ioctl	11
crafted syscalls	11
write	3
setsockopt	3
connect	3
syncfs	2
request_key	2
rename	2
mmap	2
mincore	2
execve	2
add_key	2
accept	2

The syscalls used in most attacks, what types of risks they introduced and which were the most harmful were studied. Table 13 lists the most frequently used syscalls among the 101 attacks reported between 2016 and 2022. Table 14 describes the most damaging benefits for attackers that exploit these vulnerabilities. From the frequency of the syscalls that appeared in the CVEs, *ioctl* syscall is the most frequently used in attacks. Thus,

the focus remained on the *ioctl* syscall, which is also frequently used in common Linux commands.

Table 14: Benefits for attackers exploiting the vulnerabilities listed in Table 13.

Attacker's benefits	**Frequency**
Denial of service	40
Privilege escalation	21
Leak/read kernel internal info/process memory	13
Allow remote attackers to execute arbitrary code as admin	13
Crash the system	11
Protections disable / Pass security checks / Bypass KASLR protection mechanism / Bypass ASLR	7
Remote control of the device using user privilege	6
Gaining unauthorized access	3
Write in the kernel memory / Write in memory without having permission to do so	2
Kernel stack corruption	2
Heap / memory corruption	2
Gain access to a physical page	2
Copy attacker's data anywhere in the memory / Overwrite memory	2

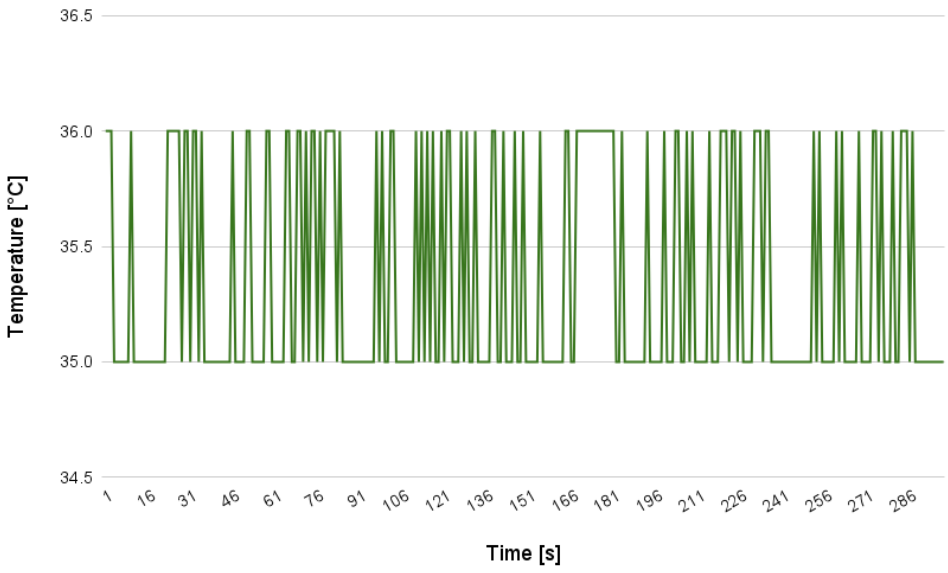


Figure 18: Temperature for *read* system call.

3.2.6 Anomaly detection using thermal sensors

The proposed Hot-n-Cold approach follows the anomaly detection definition, which implies obtaining information about the system during its most stable and usual circumstances and comparing against it with when an anomaly is injected [166]. An experiment was conducted that aimed to detect and identify syscalls using thermal traces. During the experiments I reach the conclusion that a system call cannot be extracted or located in the measured temperature, as seen in Figure 18. In the experiment, no differences were noticed in the core temperature that would distinguish syscalls. This is attributed to the low resolution and sampling capacity of the sensor. In conclusion, the general belief is that the capability of the sensors prevents the differentiation between syscalls.

3.2.7 Linux commands used in experiments

Taking into consideration what was written above, the experiments were continued using Linux commands. Several commands were eliminated from the top used list because extraction of the syscalls with *strace* was impossible, for example: *exit, export, sudo, find, screen, logout, for*, etc.

For others, I could not measure their temperature because they could not have been run repeatedly and automatically, for example: *exit, kill, find, screen, logout*, etc. In particular, the *exit* and *logout* commands stop the execution of scripts or log out, respectively, making them impossible to use in my experiments.

To reduce the overall number of Linux commands, a subset was selected for experimentation based on two criteria: the linear behavior of the CPU temperature measured during its execution and how common a command is. To establish the linear behavior of the temperature associated with a Linux command, each command was repeatedly called for 5 minutes, with a 1-second pause between calls, when the temperature of the core was read. A set of highly popular commands was further selected, even though their temperature profile was not linear. This selection results in a list of 10 commands: *chmod, cp, du, file, history, ls, mkdir, mv, tar,* and *w*, as seen in Figure 19.

For the 10 selected commands, their execution time was measured using *time* command, an example can be seen in Figure 20.

Based on Figure 21, one can observe that the execution of a command requires anywhere between 1 and 23 milliseconds.

3.2.8 Augmentation of Linux commands

To augment the original Linux commands with the *ioctl* syscall and still let them continue to function correctly, the commands that utilize file descriptors in their source code were selected. The four such chosen commands are: *chmod, cp, ls* and *mv*. The source code for the Linux commands can be found in the GNU core utilities [190]. The code of the selected commands was modified by adding three to five dummy *ioctl* calls, close to a section of code that already uses file descriptors, to serve as a parameter to the dummy *ioctl* syscall introduced by us.

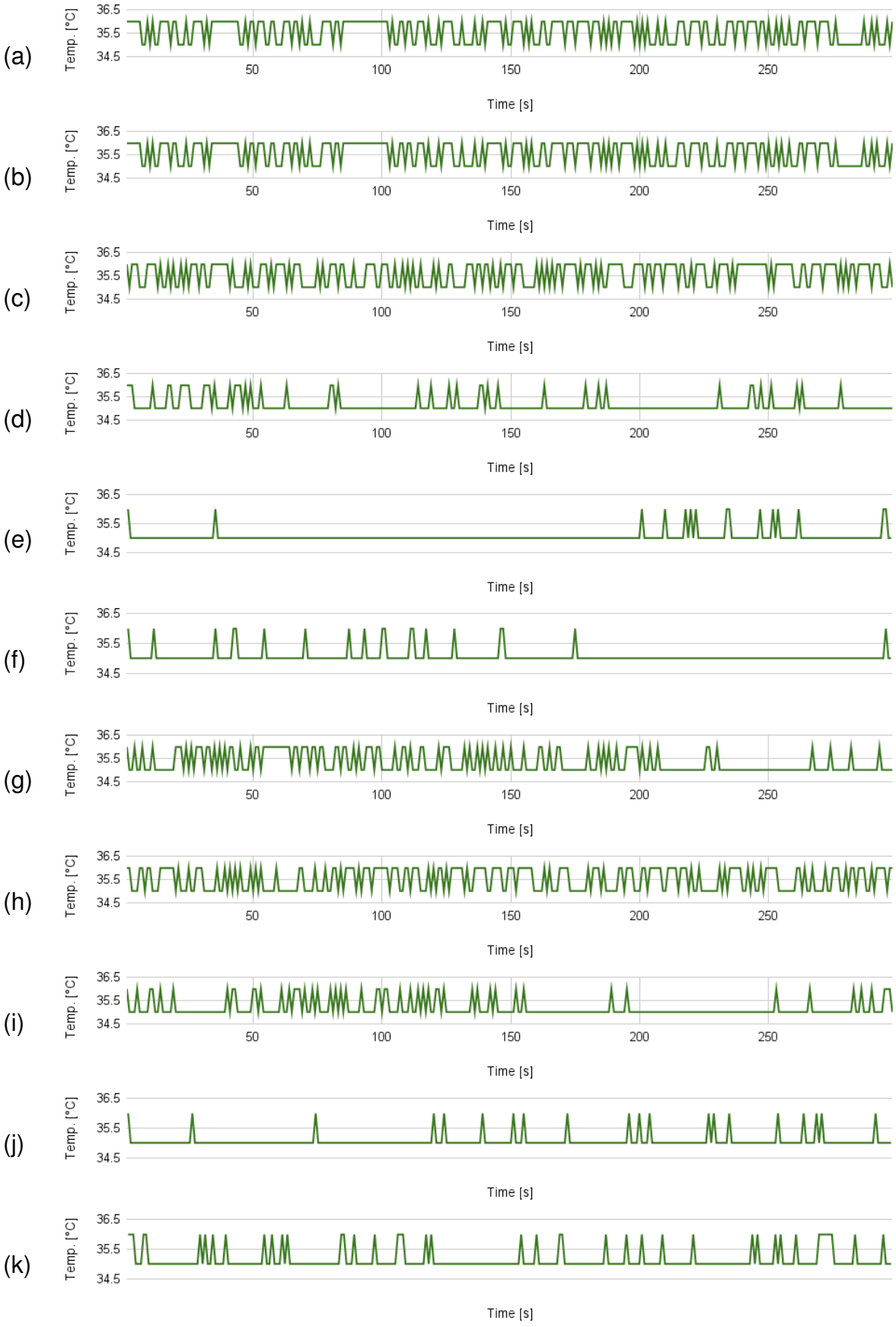


Figure 19: Temperature of the 10 Linux commands considered initially

```
teo@teo-Yoga-Pro-7-14IRH8:~$ time ls

real    0m0,012s
user    0m0,001s
sys     0m0,004s
```

Figure 20: Example of output of *time* command.

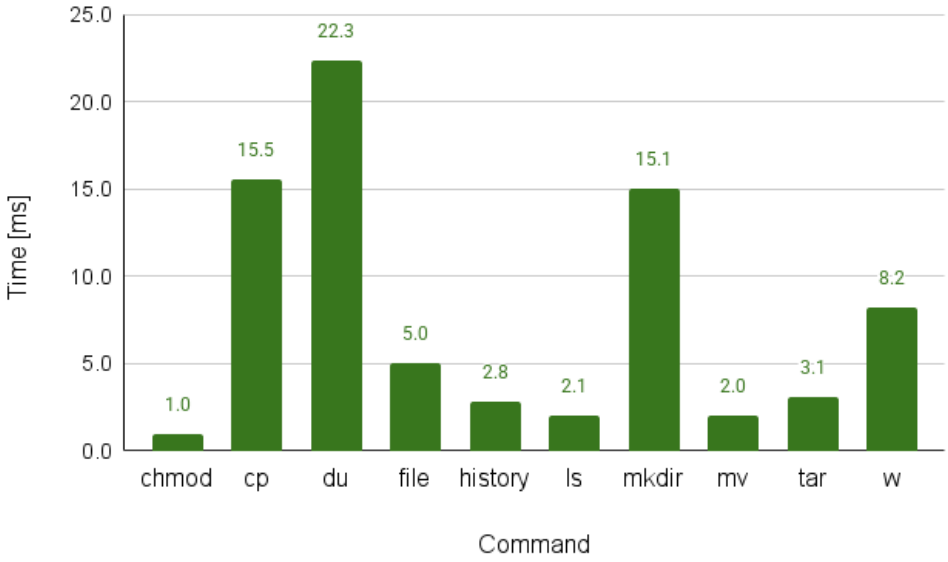


Figure 21: Execution time of the 10 Linux commands considered initially

After modifying the four selected commands, only two of the commands, *chmod* and *ls*, continued to work the same way as the original commands. Thus, for this reason, experiments with *mv* and *cp* were not carried out anymore.

3.2.9 Baseline system temperature.

Knowing the baseline temperature helps to understand what is the CPU's normal temperature range in idle state or under typical load. This is an essential thing to know when one want to know how different actions affect the CPU temperature.

The baseline of the CPU temperature was started using a script, *readTempForOneWeek*, that reads the temperature of the entire socket together with the cores for one week. The reading of the temperature sensors was performed every second, for a total of 5 minutes, with a pause of 20 minutes between experiments. In this way, CPU activity was recorded over several time intervals throughout the day, and these measurements were repeated every day of the week. The baseline temperature of the selected socket was measured in a controlled environment, while no other user process was running, except

for the services necessary for the OS to work properly. In addition, the script reading the temperature was executed from another socket of the respective two-socket node. This way, the influence of heat dissipation over the baseline temperature due to running the script was eliminated. The *taskset* command was used to select the core on which to run the temperature reading script.

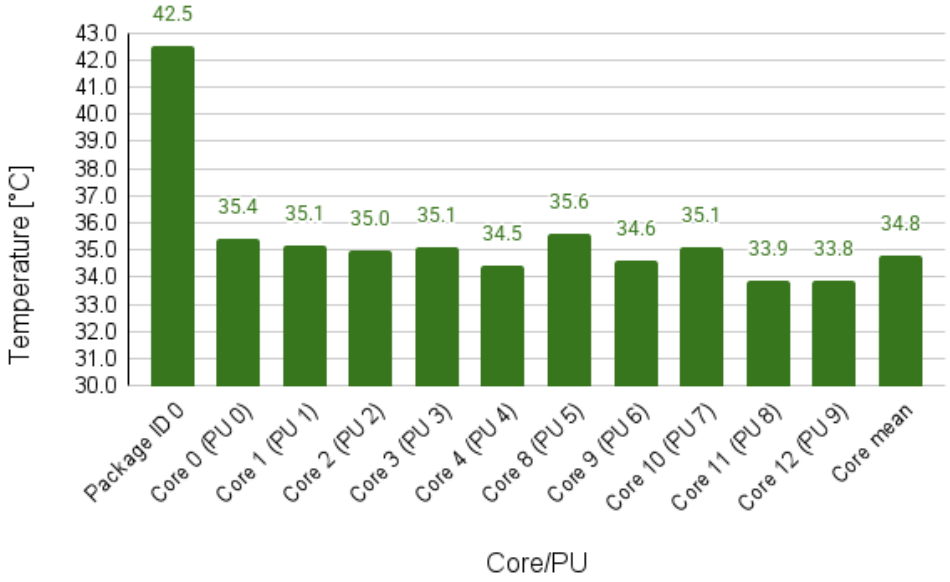


Figure 22: Baseline temperature of the system

While the baseline temperature was measures, also the daemon status was checked every second for 24 hours, in order to see if there are any major changes in the functioning of the system. The daemons were read using `systemctl | grep daemon` and their status was was compared during the measurements. In Table 15 the daemons and thir status can be seen.

Table 15: Daemon status.

Daemon name	**Status for 24h**
irqbalance.service	running
libstoragemgmt.service	running
lvm2-lvmetad.service	running
slurmd.service	running
sshd.service	running
dm-event.socket	listening
lvm2-lvmetad.socket	running
lvm2-lvmpolld.socket	listening

3.2.10 The scripts

Two types of scripts were developed and used to run the experiments, namely the *readTemp* script (see Listing 1), which reads the temperature of the whole socket and the *runCommands* script (see Listing 2), which runs the respective command.

Listing 1: readTemp script

```
#!/bin/bash
secs=10
endtime=$(( $(date +%s) + secs ))
fileout=$1
while [ $(date +%s) -lt $endtime ]
do
    cat /sys/devices/platform/coretemp.0/hwmon/hwmon1/temp* >> "
       $fileout"
    sleep 0.002
done
```

Listing 2: runCommands script

```
#!/bin/bash
ls
sleep 0.01
ls
sleep 0.01
...
```

To avoid a possible heat propagation between the cores, as described in the literature [146, 27], I chose to run the *runCommands* on the first core of the first socket, while the *readTemp* script ran on the second socket, as far away as possible from each other. Again, the *taskset* command was used to select the core to run each of these scripts, as seen in Listing 3.

Listing 3: *taskset* scripts

```
taskset -c 0 ./readTemp rawLs1.txt & taskset -c 3 ./runCommand
```

Moreover, to ensure that the measured temperatures are as unperturbed as possible by other system calls than those made by the commands in focus in this work, the scripts were constructed in such a way that they do not contain any programming expressions. Specifically, no loops were used, e.g. *for*, *while*, no conditional clauses, e.g. *if* and so on, nor expressions that may inadvertently introduce system calls. The *sleep* command is the only additional Linux command to those analyzed here, as seen in Listing 2.

3.2.11 Methodology of the experiments.

The temperature of each core and also of the package, for the majority Intel processor families, can be read using the *coretemp* [180] field of the *hwmon* interface, which can be found in the Linux file system path */sys/class/hwmon/hwmon[*]*. The *hwmon* interface offers a resolution of +/- 1 Celsius degree and a sampling rate of 2 ms [27].

For the two selected Linux commands, *chmod* and *ls*, the temperature was gathered together with the duration of a single run. The temperature was collected by running the Linux command every 10ms, 100ms and 1000ms, respectively, and reading the temperature sensors every 2ms. This is illustrated in Figure 23 in an aggregated view of the two Linux command experiments, which, however, were performed in isolation.

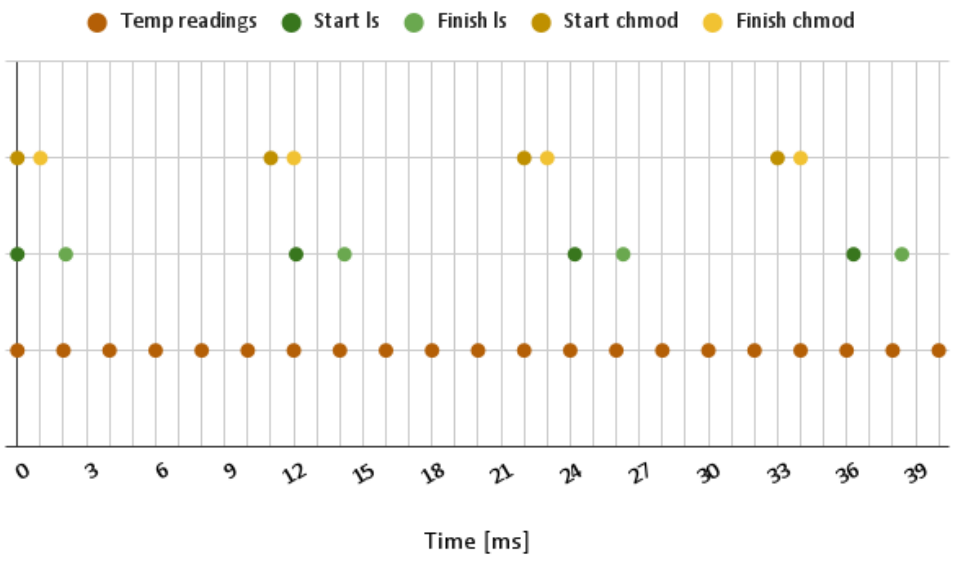


Figure 23: Methodology of temperature measurements with respect to execution lengths of the two selected Linux commands: *chmod* and *ls*.

Both experiments are presented in one figure to better illustrate the differences between the length in time of the executed Linux command. The x-axis denotes the time, in milliseconds, for the execution of the two scripts. The orange dots on the first line on the y-axis denote the points where the actual temperature reading is done, namely every 2 ms. On the second and third rows of dots on the y-axis, the starting point of the command is shown in a darker color, while the lighter color represents the endpoint of the command. The green color denotes the *ls* command, which takes 2.1 ms to execute, according to Figure 19. The yellow color denotes the *chmod* command, which takes 1 ms to execute, according to Figure 19.

The main idea behind this methodology is to observe the differences in the temperature between the original and the modified command, and to detect, using the thermal sensors, an anomaly which could indicate a possible attack.

3.3 Hot-n-Cold Results

In this subchapter, I seek to observe and find the differences in the temperature of the original and augmented syscall commands. The *ls* and *chmod* commands were used in the experiments. Each of these commands was executed 300 times with three different time gaps between calls: 10 milliseconds, 100 milliseconds, and 1000 milliseconds.

3.3.1 *ls* command.

In Figure 24, the green time series denotes the temperature of the original *ls* command instances, and yellow denotes that of the *ls* command augmented with the *ioctl* syscall.

One can see that there is a difference in temperature of 0.5-1 Celsius degree between these commands. This indicates that the augmented commands dissipate more heat than the original ones. To verify whether these two temperature profiles follow the same pattern, the Pearson correlation index was used between the vector of values of the original command and the modified one. The Pearson correlation index is 0.90 for the commands executed with time-gaps of 10ms between calls, and 0.22-0.29 for the experiments with 100ms and 1000ms time-gaps between calls, respectively. The smaller correlation index between the two runs with more time between command calls can be attributed to the CPU cooling down faster between calls.

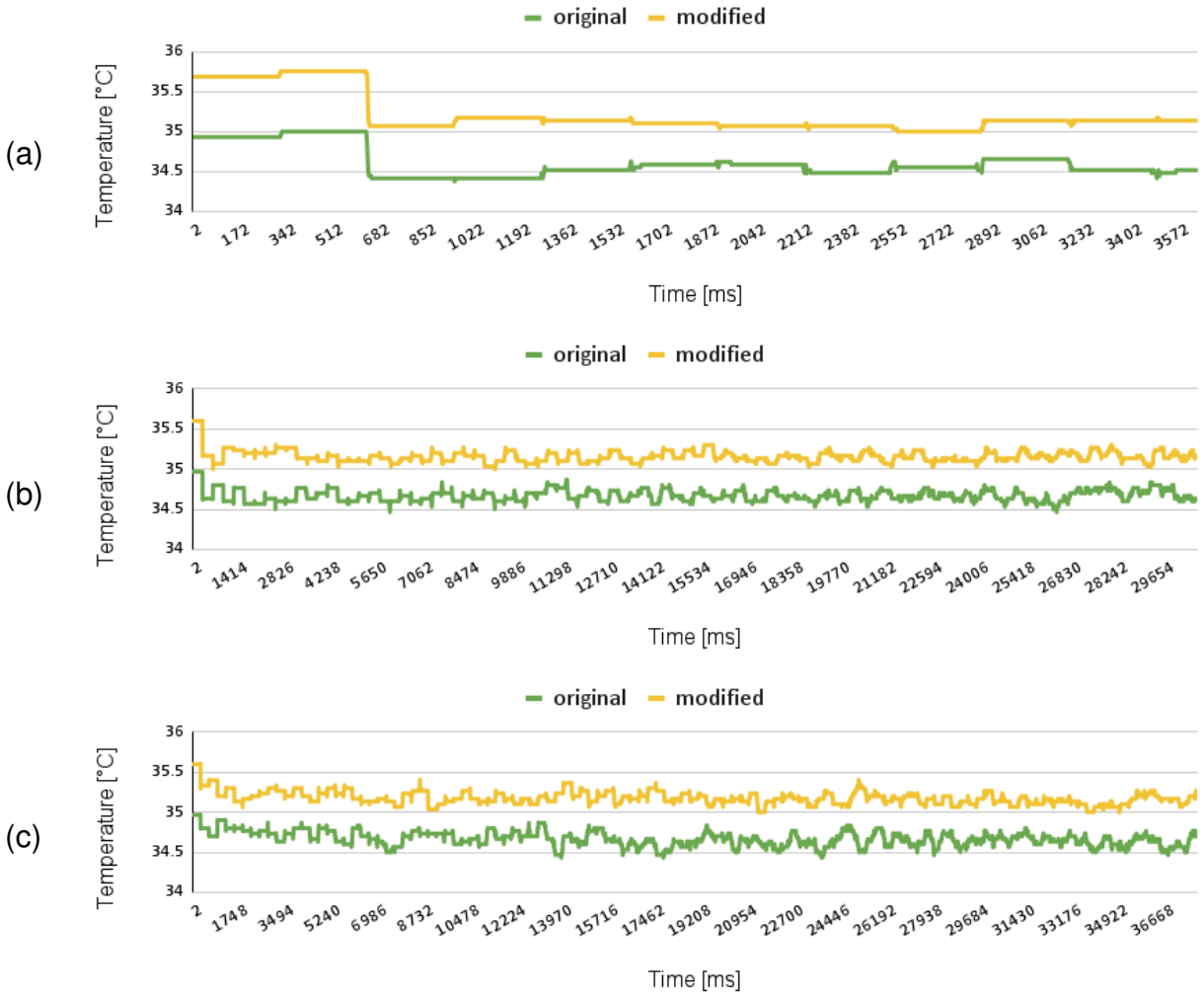


Figure 24: Temperature profile of the *ls* Linux command. Where a) contains the results for 10ms pause between calls, b) for 100ms pause and c) for 1000ms pause.

3.3.2 *chmod* command.

In Figure 25, the green time series denotes the temperature of the original *chmod* command instances, and yellow denotes that of the *chmod* command augmented with the *ioctl* syscall. Similarly to the observation from Figure 24, in this case, there is also a difference in temperature of 0.5-1 °Celsius between these commands.

Similarly to the *ls* command, there is a visible correlation between the two temperature profiles, for each time-gap experiment. Specifically, the Pearson correlation index is 0.95 for the commands executed with 10ms between calls, and 0.27 and 0.02 for those with 100ms and 1000ms time gaps, respectively, between command calls.

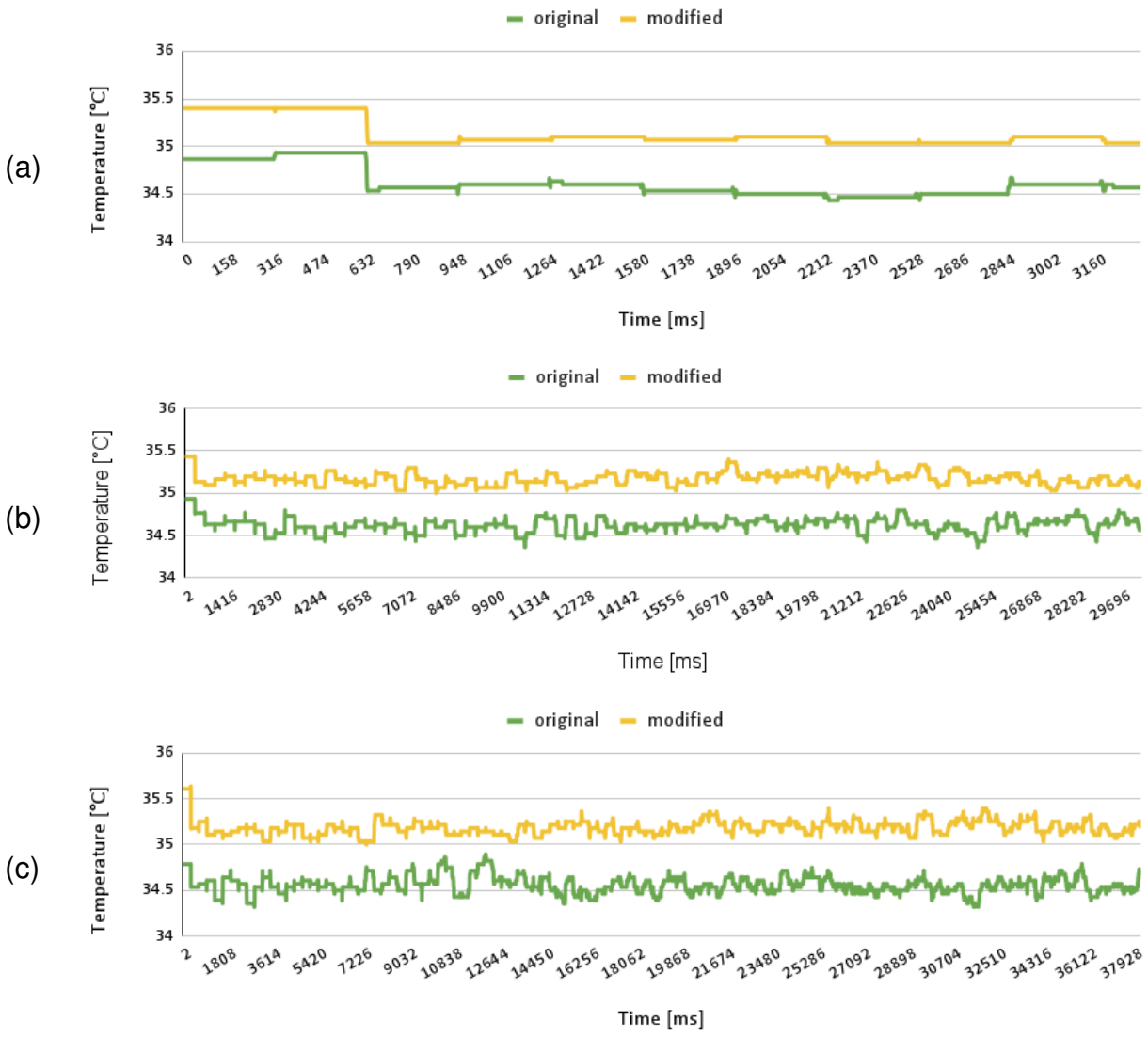


Figure 25: Temperature profile of the *chmod* Linux command. Where a) contains the results for 10ms pause between calls, b) for 100ms pause and c) for 1000ms pause.

3.3.3 Limitations of the proposed approach.

The nature and type of sensors used in this work impose restrictions on the method and insights. These temperature sensors have a resolution of 1 degree Celsius, which is too low and makes it difficult to distinguish the temperature of distinct syscalls. Therefore, I resort to using Linux commands instead of syscalls, as they have longer execution times (milliseconds) than syscalls (nanoseconds). Another constraint is the heat propagation between cores, which was circumvented in my experiments by executing the two scripts on distinct sockets of the same computing node. However, in real-life situations, the cores neighboring the core on which the temperature was measured during the execution of Linux commands also propagate heat, which may perturb the results obtainable with the Hot-n-Cold approach.

3.3.4 Propagation of heat through cores.

The propagation of heat from one core to another can influence the proposed detection method. Two cores that are physically near one another can influence each other's temperature reading. In [146], an attack was mounted based on the heat that originates in a nearby core. In this way, they were able to detect if the CPU core was processing a value of *1* or *0*. In [27], a short experiment shows how one core is influenced by the temperature of the other. Based on the above literature findings, I decided to develop two scripts, one for measuring the CPU temperature, and another one for calling the Linux commands, to run them on distinct sockets, as far away from each other as possible, to avoid any such heat propagation.

However, reproducing the heat propagation behavior by repeatedly calling Linux commands was tried. Only a very small or no difference in temperature was observed on the core close to the core that executed the command repeatedly. This is probably because reduced computing power is needed to process these commands. This leads to the conclusion that real heat propagation studies can be conducted using benchmarks, such as CPU Burn-in [191], Mersenne Prime Search[192], the MiBench Suite [147], or by implementing a stress algorithm as in [146].

3.4 Original contributions

As part of this research, the findings have been disseminated through a peer-reviewed publication. Notably, the article ***Hot-n-Cold: Mapping the Syscall Attack Surface Using Thermal Side Channels*** was published in The 22nd International Symposium on Parallel and Distributed Computing (ISPDC) (pp. 93-100), IEEE.

In this work, ***Hot-n-Cold*** **technique is introduced, a novel dynamic analysis technique designed to detect anomalies in the behavior of Linux commands by leveraging Intel CPU Digital Thermal Sensors**. To my knowledge (in the beginning of 2025), this is the first approach that utilizes CPU temperature variations as an indicator to identify the attack surface created by system calls. The method relies on correlating temperature traces recorded during the execution of original Linux commands with those of augmented versions that include high-risk system calls, as reported in Common Vulnerabilities and

Exposures (CVEs). By analyzing these correlations, Hot-n-Cold is able to identify abnormal thermal patterns that suggest the presence of potentially malicious or unsafe syscall activity.

The implementation employs the Intel *coretemp* kernel module to access the Digital Thermal Sensors through the Linux *hwmon* interface, allowing temperature measurements without requiring elevated privileges. This makes the proposed method suitable for the detection of thermal anomalies. The experimental evaluation demonstrates a strong correlation, up to 0.95 using the Pearson index, between the thermal profiles of the original commands and their augmented counterparts, with a temperature difference of approximately 0.5 °C. These findings confirm that syscall-induced behavioral deviations can be captured thermally, thus establishing a new form of security based on CPU temperature.

The key idea behind *Hot-n-Cold* is that the thermal characteristics of a processor provide an observable side channel that reflects the internal activity of the system. By continuously measuring the CPU temperature during command execution, one can detect subtle deviations in thermal behavior that may indicate the presence of injected or malicious syscalls. This approach represents a new form of behavioral profiling, using physical measurements as indirect indicators of software integrity. Hot-n-Cold is used to map the system call attack surface of Linux commands on a local HPC system, showing that temperature-based monitoring can serve as a low-cost and non-intrusive method for anomaly detection. The experiments focus on two commonly used Linux commands, `ls` and `chmod`, and compare their thermal profiles in normal operation against versions augmented with syscalls extracted from the known CVEs. The results reveal a strong positive correlation of up to 0.95 between the thermal traces of original and modified commands, with a heat difference of up to 1 °C, demonstrating that CPU temperature can be used to characterize and distinguish legitimate executions from potentially compromised ones.

CHAPTER 4

4 Contributions in Computing Systems Cyber-Security

4.1 BeatTheHeat Method

A user process can use any of the existing system calls in an operation system while executing various programs. If any of the executed programs interacts with a jeopardized syscall, it could breach the kernel space, potentially resulting in unauthorized data access, privilege escalation, system crashes, the leakage of internal kernel information, and other issues. These consequences can ultimately lead to the unauthorized acquisition of valuable and sensitive data.

Moreover, there is a correlation between the temperature of a CPU and the applied current and voltage, which means that the differences in temperature can be detected for identical sets of instructions but different operands [27]. The above statement leads to the main idea of this study that monitoring the temperature of a CPU while an original application and one that contains additional compromised code are running must bring differences in temperature, but not in the pattern of the execution.

As stated in the above chapter, Hot-and-Cold is a technique that detects anomalies using the *ls* and *chmod* Linux commands, which shows a correlation up to 0.95 between an original and an augmented command, the second having a higher temperature due to the additional code. However, this technique was used in a controlled environment with zero noise, which in fact did not lead to a strong conclusion about the application of the technique in real-life scenarios.

Therefore, this chapter introduces *BeatTheHeat* technique, which builds upon the Hot-n-Cold anomaly detection technique, with the objective of making it work in more realistic computational environments. This is achieved by monitoring the CPU temperature to detect irregularities in the Linux command behavior. The innovative aspect of this approach is the inclusion of five types of noise during the operation of the attack detector. These noise types encompass user-specific activities, such as moving files, performing extended mathematical computations, playing songs, and browsing the web. This added complexity simulates real-world conditions more accurately, boosting the robustness and practicality of the anomaly detection method.

BeatTheHeat has the objective of identifying an attack created using crafted system calls that are added in the original code, in scenarios similar to real life. The hypothesis of the work is that an augmented command, containing additional code, will dissipate more heat; however, the execution visual results of the original and the augmented command will be very similar. Temperature monitoring is performed again via Intel digital sensors, which are integrated into Intel processors to monitor the CPU temperature. Similarly, the augmentation is performed based on information from the Common Vulnerability and Exposures (CVE) database [193]. To process and compare the two temperature vectors of the original and augmented with code commands, I used the method of correlation

thermal analysis, which is a variation of the well-known correlation power analysis (CPA) technique, together with the Pearson correlation.

4.1.1 Experimental methodology

Experimental Setup The experiments are performed on three systems situated in a room with controlled temperature, set to 20 Celsius degrees. The systems are Dell and HP desktops and contain Intel CPUs, the first one has an Intel Core i7-10700F 2.90 GHz CPU with 8 physical cores (which will be further addressed as *D-150a*), the second one has an Intel Core i5-4590 3.30 GHz with 4 physical cores (which will be further addressed as *D-151a*) and the third one an Intel Xeon W3550 3.07 GHz CPU with 4 physical cores (which will be further addressed as *D-152a*). All of them have the same exact OS and installed programs, achieved using Clonezilla [194], which helped to clone the first setup and transfer it to the other two desktops. The OS used is Ubuntu 20.04.1 together with a kernel version of 5.15.0. The naming conventions for the systems come from the last number in the their IP used in their remote connection, and the letter *D* comes from *desktop*.

The Augmentation The experiments were performed using *ls* and *chmod* Linux commands. The idea behind this choice is that these commands are widely used by any user of Linux distributions, even if they use desktop systems or servers. All other factors are presented in Chapter 3.2.7.

For the original commands, the built-in commands were used and for the augmented ones, the code sources of the original commands were downloaded from the GitHub repository [195] in February 2023.

To augment the original commands, I used the *ioctl* system call, which is one of the most commonly used in attacks involving system calls. The attacks performed using this syscall led to privilege escalations, system crashes, or gaining unauthorized accesses, as presented in Chapter 3.2.5.

The augmentation was performed by calling the *ioctl* C function near the usage of a file descriptor in the original code, similar to in Chapter 3.2.8. However, it was performed slightly differently by minimizing the number of calls of the syscall. Moreover, I tried to avoid using similar data in the code to bypass cache references, since I concluded after some experiments that the OS optimizations for desktops are harder than the ones in HPC systems. The augmentation rate in terms of lines of code (LOC) is 0.10% addition for *ls* command, and 2% addition in the code for *chmod* command.

The Noise The aim was to bring the technique from a controlled environment, where one node of the HPC system was used solely for the experiments, to a more real-life scenario, similarly to the one where a normal user utilizes a computer. The noise is constructed in a way that simulates the behavior of a user who can listen to music, move files from one folder to another, surf the browser, or write on the keyboard.

The *playSong* script starts the *ffplay* player [196] and plays the Baby Shark Dance song, which is the most viewed video on YouTube, as seen in the Listing 4.

Listing 4: *playSong* script

```
#!/bin/bash
a=0
while [ $a -lt 55 ]
do
     ffplay -v 0 -nodisp -autoexit song.mp3
     a=‘expr $a + 1‘
done
```

The *moveFiles* script moves a file from one folder to another in a repeated way, as seen in Listing 5.

Listing 5: *moveFile* script

```
#!/bin/bash
secs=7300
endtime=$(( $(date +%s) + secs ))
fileout=$1
while [ $(date +%s) -lt $endtime ]
do
    mv /home/files-to-be-moved.zip /home/results/files-to-be-moved.
       zip
    mv /home/results/files-to-be-moved.zip /home/files-to-be-moved.
       zip
done
```

The *startBrowser* script is simulating surfing on the web, opening a list of websites in the Firefox browser, as seen in Listing 6.

Listing 6: *startBrowser* script

```
#!/bin/bash
secs=7300
endtime=$(( $(date +%s) + secs ))
fileout=$1
while [ $(date +%s) -lt $endtime ]
do
firefox www.google.com &
sleep 1s
firefox www.maps.google.com &
sleep 1s
firefox https://stackoverflow.com/ &
sleep 1s
firefox https://ubuntu.com/ &
sleep 1s
firefox https://www.geeksforgeeks.org/ &
sleep 1s
firefox https://ro.wikipedia.org/wiki/Pagina_principal%C4%83 &
sleep 1s
firefox https://www.ieee.org/ &
```

```
sleep 1s
pkill firefox
done
```

The *simulateKeystrokes* script is simulating the writing of a text with a pause of 100 ms between each key press, as seen in Listing 7.

Listing 7: *simulateKeystrokes* script

```
#!/bin/bash
secs=7300
endtime=$(( $(date +%s) + secs ))
while [ $(date +%s) -lt $endtime ]
do
    xdotool type --delay 100 "Simulating keystrokes in a shell
       script can be achieved using the xdotool utility in Linux..."
sleep 0.5
done
```

The last script *mathComputations* runs different algorithms like division by 6, the sum of series, perfect square, the sum of two polynomials, binary to hexadecimal conversion, and the kth prime factor, the factorial of a given number and Fibonacci series, as seen in Listing 8.

Listing 8: *mathComputations* script

```
#!/bin/bash
secs=7300
endtime=$(( $(date +%s) + secs ))
while [ $(date +%s) -lt $endtime ]
do
     ./mathComputations
done
```

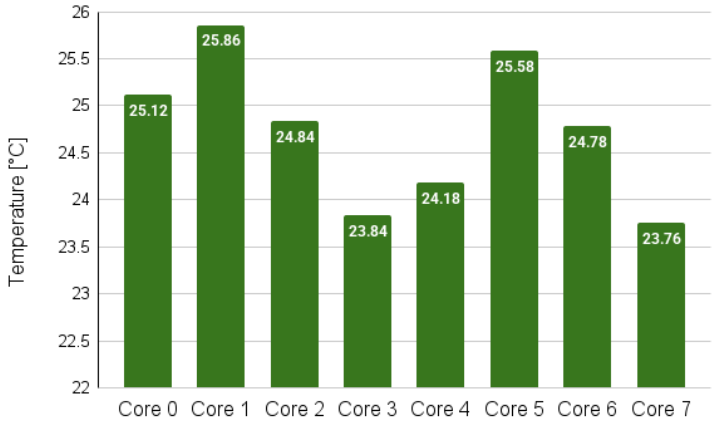
Methodology of the Experiments A trace of the experiment is a temperature measurement that contains 600 calls of the Linux command with a sleeping time of 10 ms between each call. I decided to use 10 ms of sleep for the experiments because it had the best results from the Pearson correlation point of view in the previous chapter. Each trace was run 50 times and an average temperature vector was calculated for the final results. I have increased the number of traces run compared to the previous work, from 30 to 50, to improve the accuracy of the experiments, and doubled the number of commands called in each script. I did not manage to further increase the number of samples because it would have required too much time to run.

As already mentioned, I used three different desktop systems (i.e., *D-150a*, *D-151a*, and *D-152a*) for the experiments, all placed in a controlled temperature environment. Unlike other studies (i.e. [27]), I did not fix the frequency of the desktops, nor did I disable the Intel Turbo Boost, which makes the experiments more similar to a real-life scenario.

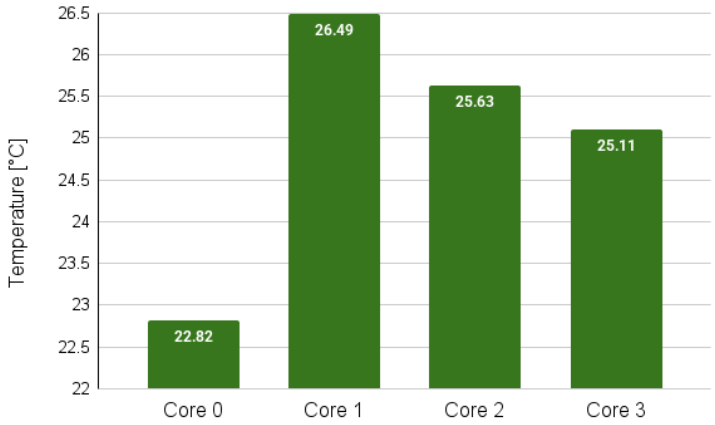
First, I started the experiments by measuring the baseline temperature of the systems (see Figure 26). The baseline was achieved by reading the temperature of all cores for

24 h, in rounds of 5 min reading once every 20 ms, with a 15 min pause. This part is important because one can see the temperature for each core in idle state. Additionally, I checked all the daemons and processes that are running while the computer is in an idle state for another 24 h, to verify their stability in running and make sure they will not influence the experiments.

(a)



(b)



(c)

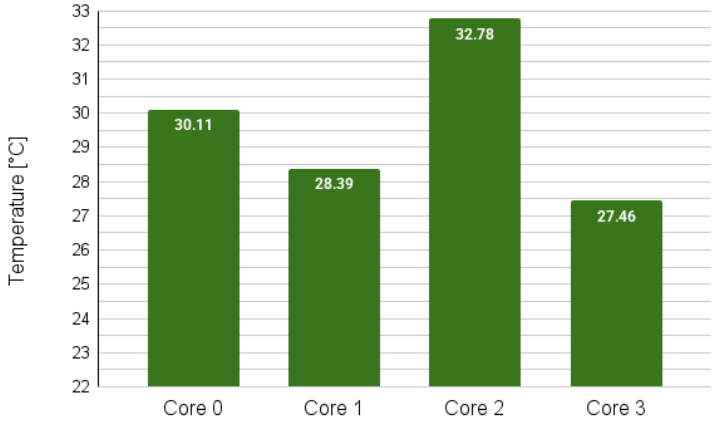


Figure 26: Baseline temperatures of the three systems. Where a) is the baseline temperature for system *D-150a*, b) is for *D-151a* and c) for *D-152a*.

Based on the baseline temperature of the systems and the architecture of the CPUs, I selected the cores on which the experiments are to be run. For each system, I selected two cores with a lower temperature and physical location on the die to be as far away as possible from each other, so that the two running scripts do not influence each other. In Figure 27, the methodology of the experiments is presented, where it can be seen that, on the rightmost core, the temperature of the entire CPU is collected with the *readTemp* script and, on the leftmost core, the *runCommands* script runs. Running the scripts on a specific core was performed via *taskset* command, which let one select the CPU affinity, more exactly, on which core the command is needed to be run [197].

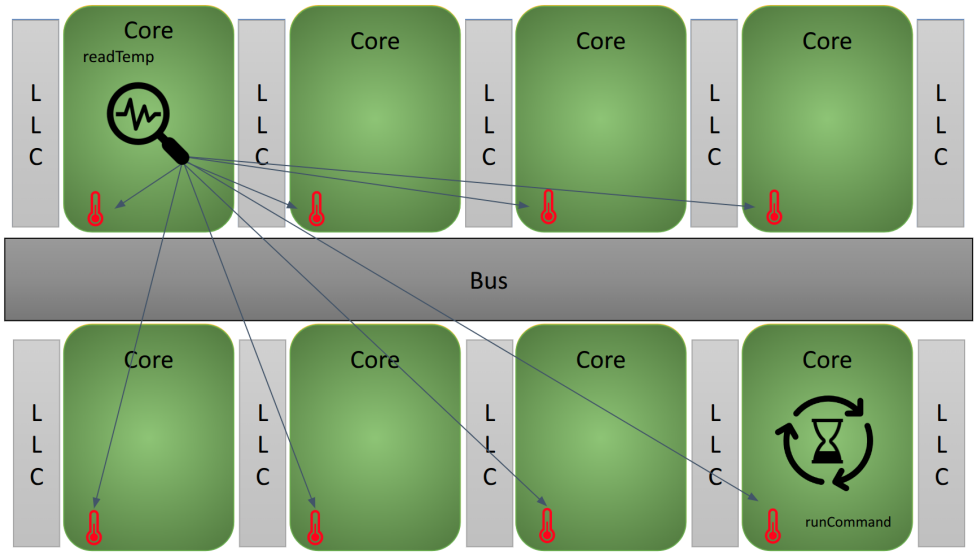


Figure 27: Methodology of the experiments.

Second, I have run the same scenario as in the above chapter, to reproduce the results. As I already mentioned, the *ls* and *chmod* commands were used, which are both original and augmented versions.

Next, after the results were reproduced, I continued to run the scripts mentioned above; however, this time, I started to introduce the noise. The experiments that included the noise were run using two different scenarios. The **first scenario** (using CPU affinity) includes running the noise script on the same core as the core where the commands are run, which leads to higher noise on that specific core. The *second scenario* (without CPU affinity) includes running the noise script without selecting a CPU affinity, and letting the OS choose where to run; in this way, I can have a different type of background noise.

4.1.2 Results of the *BeatTheHeat* method

This part of the work aims to detect anomalies in the operation of a CPU based on the core temperatures. This is achieved by observing differences in the temperature of two Linux

commands in their original and augmented states and similarities in their behavior. In addition, graphs with the results will be presented, containing a comparison of the vectors (traces) of temperatures for each command in both states. In the following graphs, the green line represents the array of values for the original command and the orange line represents the vector of values for the augmented command, modified with *ioctl* calls. All figures contain the average temperature of the command vector near the series naming in brackets (e.g., *(avg: 28.4)*). To compare the similarity in a pattern of the two vectors of temperature, I used the Pearson correlation index. A high correlation is considered if the Pearson correlation index is greater than 0.7 [198]. If the two commands have a high correlation, and there is a difference in temperature between them, one can conclude that there was an anomaly in the respective command which was detected.

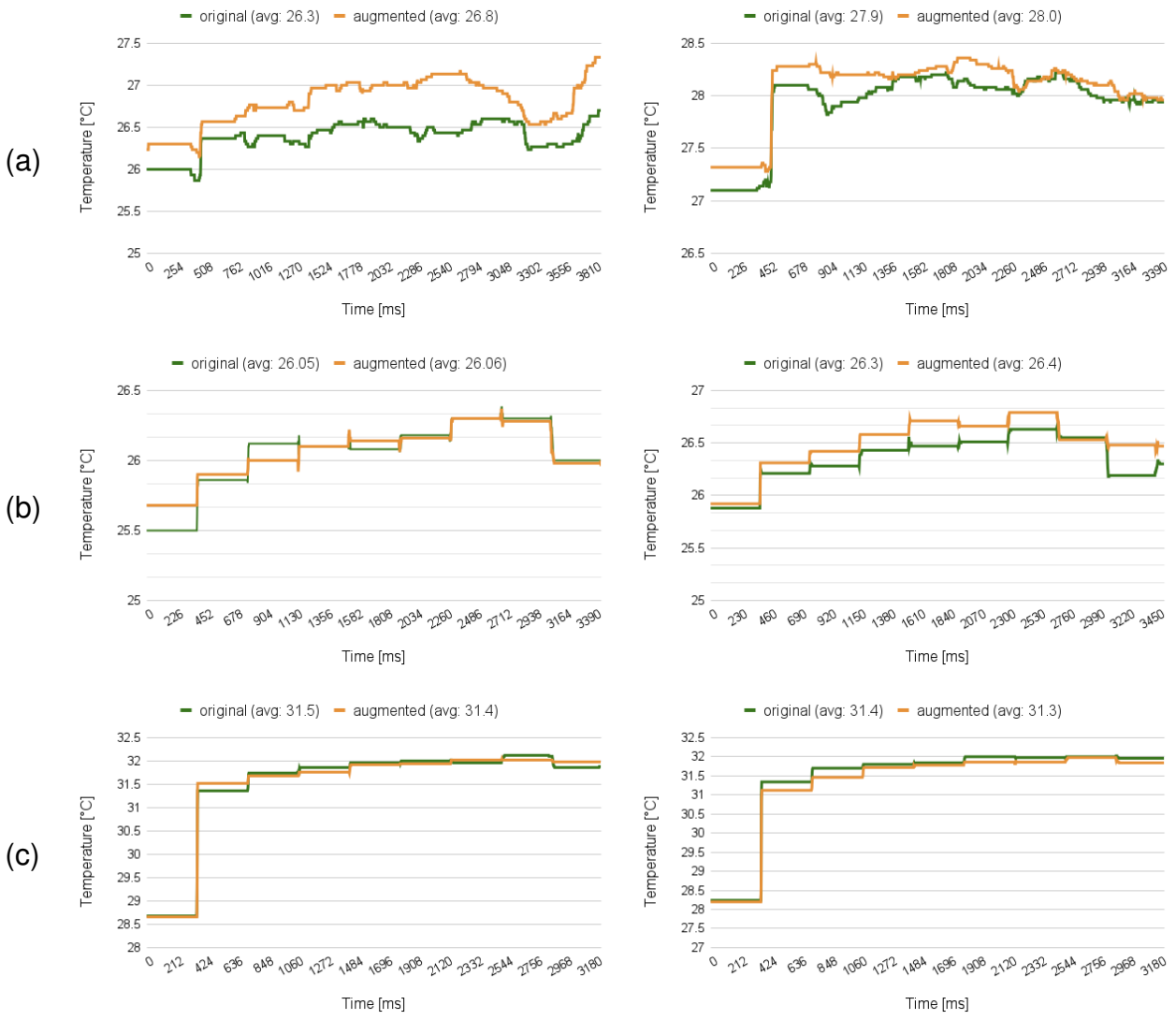


Figure 28: Results for reproducing the original experiment, a) contains the results for *ls* and *chmod* commands for system *D-150a*, b) contains results for *D-151a* and c) for *D-152a*.

Reproducing the results from *Hot-n-Cold* technique First, the aim was to reproduce the results from the original paper and to check whether the selected cores are a good option. In Figure 28, there are six graphs, one for each command (i.e., *ls* and *chmod*) that runs on all systems (i.e., *D-150a*, *D-151a*, and *D-152a*).

For *D-150a*, the results are the most accurate and a high correlation can be seen between the vector values with the Pearson index at 0.85 for *ls* and 0.96 for *chmod*, respectively. Moreover, a difference can be seen in the average temperature between the original and the augmented command of almost half a degree Celsius. The higher heat is dissipated by the augmented command, leading us to the conclusion that the reproduction of the experiments was a successful one and an anomaly was detected in the behavior of the command.

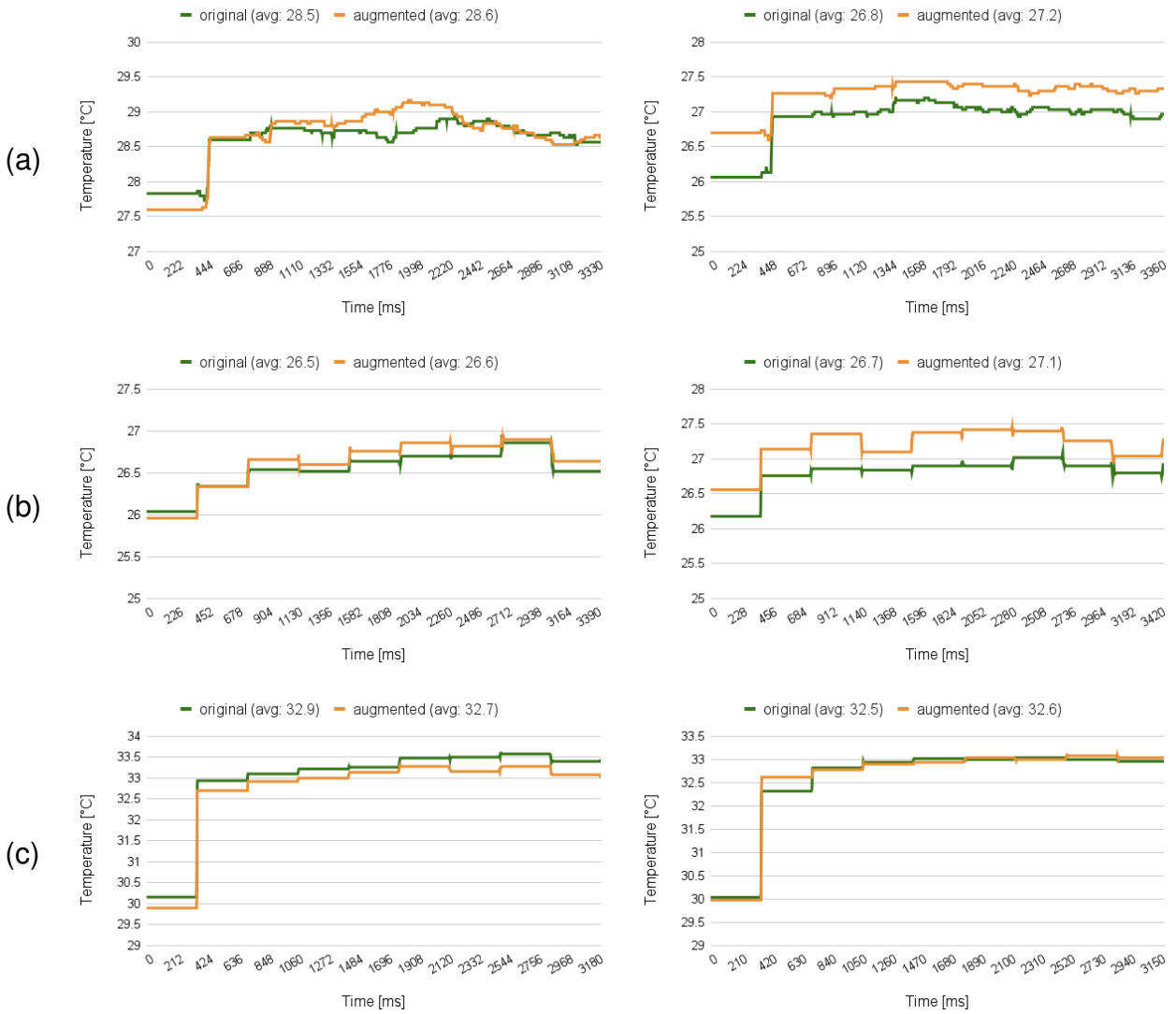


Figure 29: Results with playing song as a noise, selecting CPU affinity, a) contains the results for *ls* and *chmod* commands for system *D-150a*, b) contains results for *D-151a* and c) for *D-152a*.

For the other two systems, *D-151a* and *D-152a*, the results in terms of Pearson correlation are over 0.93. However, they are not as conclusive in terms of difference in the average temperature, where I can only see 0.1 ℃ (see Figure 28 b) and c)). I attribute these results to older CPUs, which have thermal sensors with lower accuracy, and a small number of physical cores which can influence each other through heat dissipation.

Results Using CPU Affinity As I already mentioned, when I introduced the noise, I doubled the experiments. First, the noise was applied on the same core as the command, whose results are presented in this subsection. Second, noise was used without setting the CPU affinity, whose results are presented in the next subsection. By selecting the CPU affinity, there will be more noise on the respective core.

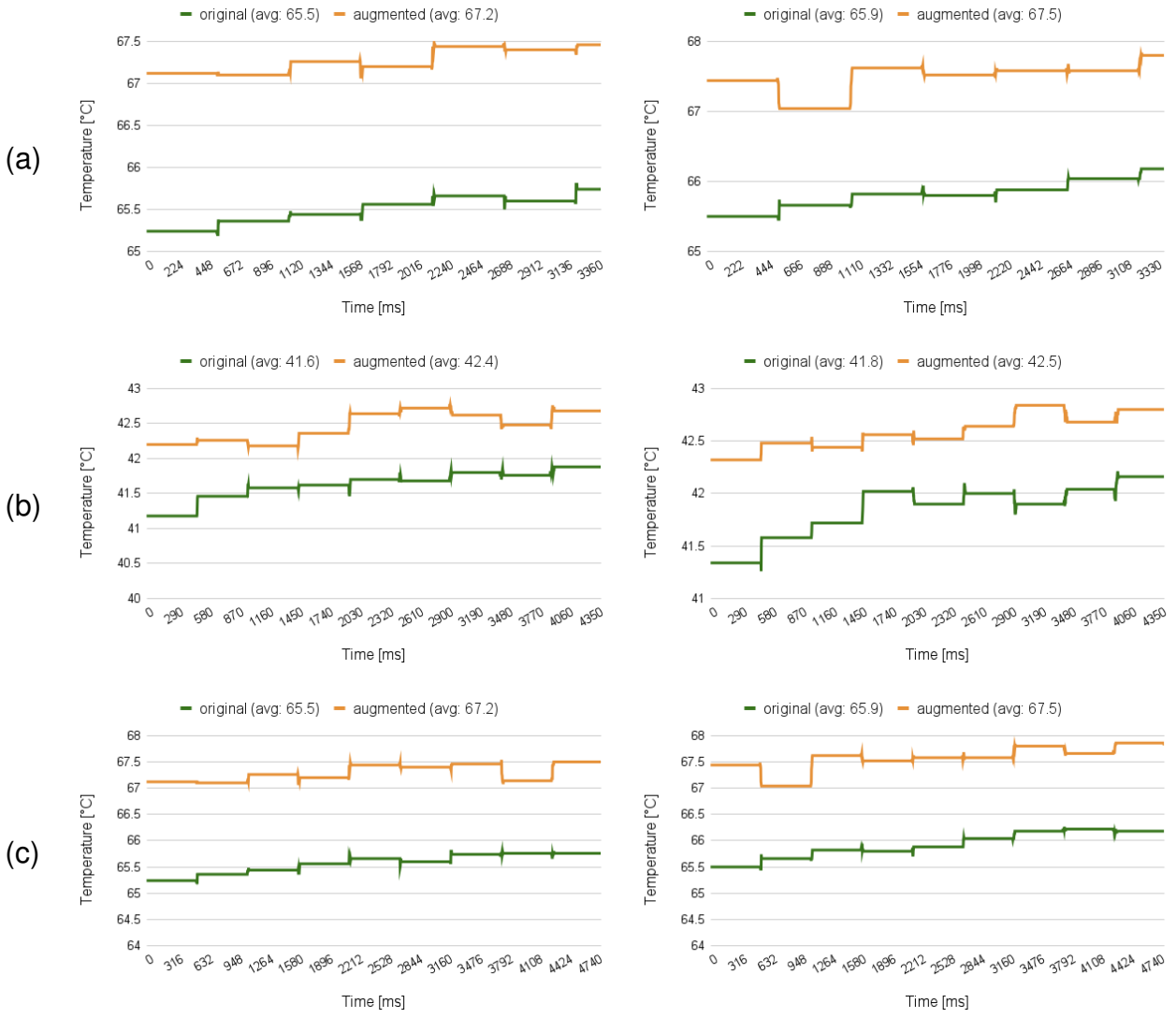


Figure 30: Results with moving a big file as a noise, selecting CPU affinity, a) contains the results for *ls* and *chmod* commands for system *D-150a*, b) contains results for *D-151a* and c) for *D-152a*.

Noise: Playing Song The experiments in which playing the *mp3* was used as noise lead to high correlations (see Figure 29 a)) of 0.94 for the *ls* command and 0.98 for the *chmod* command, respectively. I can see a difference in the average temperature between the original and augmented commands up to 0.4 ℃, as a result of which the augmented command spreads more heat. The high correlation between the pattern of the commands and the augmented command that had a higher temperature led to the conclusion that an anomaly was found. However, for the *chmod* command, I see more accurate overall results.

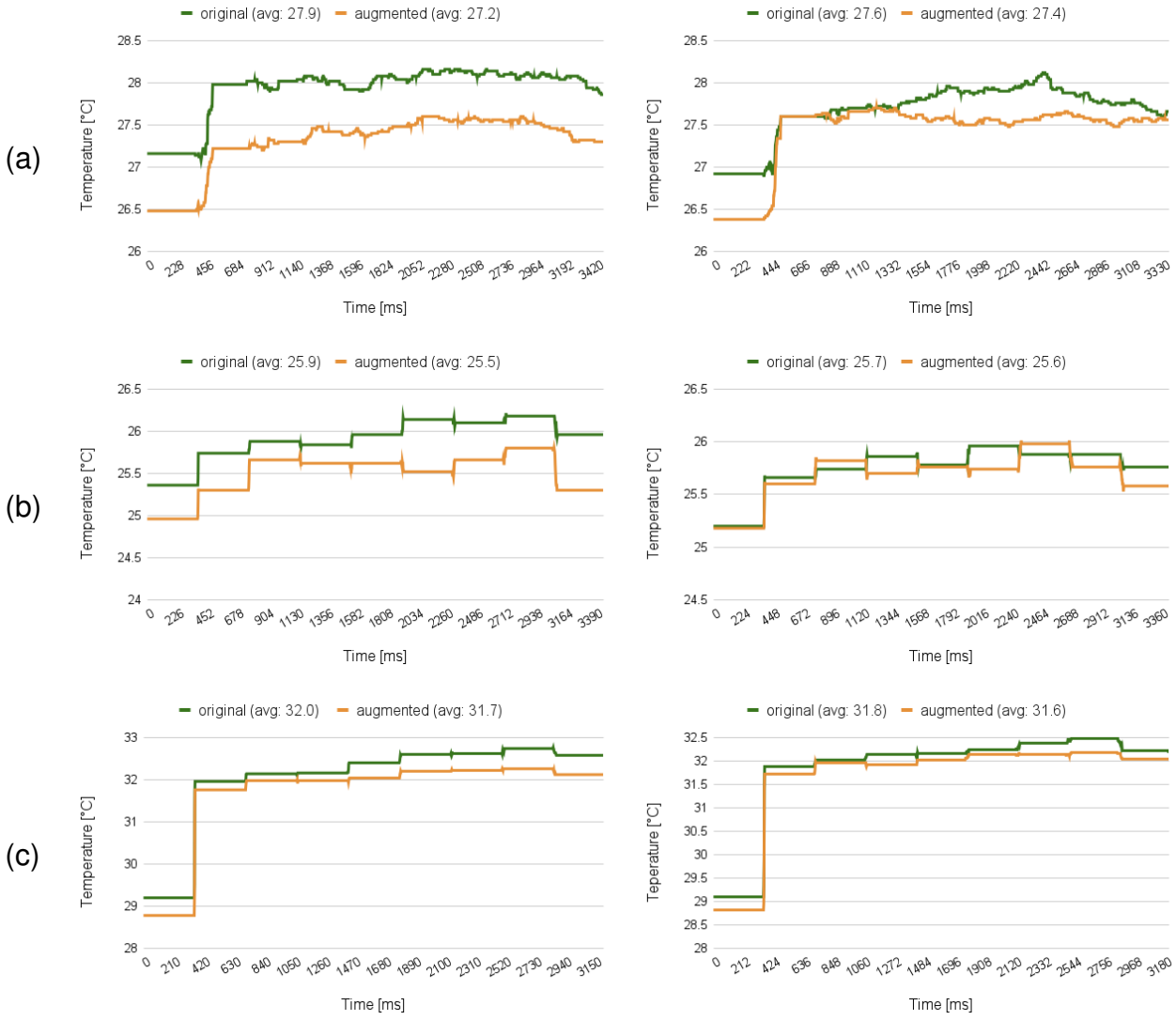


Figure 31: Results with moving a small file as a noise, selecting CPU affinity, a) contains the results for *ls* and *chmod* commands for system *D-150a*, b) contains results for *D-151a* and c) for *D-152a*.

For *D-151a* and *D-152a*, I can see a lower accuracy of the experiments, but still one can see good correlations, over 0.93 and a somewhat difference in the temperature, lower

than half of a Celsius degree. However, the results for *D-152a* show no significant results, as can be seen in Figures 29 b) and c).

Noise: Moving Files For moving files as noise, I performed two experiments, one with a large file of 18.8 GB in size (results in Figure 30) and one with a small file of 3.3 MB in size (results in Figure 31).

Figure 32: Results with surfing the browser as a noise, selecting CPU affinity, a) contains the results for *ls* and *chmod* commands for system *D-150a*, b) contains results for *D-151a* and c) for *D-152a*.

As seen in Figure 30 a), the correlation between the two temperature vectors is 0.66 for *ls* and 0.74 for *chmod*, with a difference in the average temperature of almost 2 °C. Even if the difference in temperature is visible, considering the lower correlation values, one cannot conclude whether an anomaly is taking place or not.

For moving the big file as noise (see Figure 30 b) and c)), I also see good results on the other two desktops, with correlation indexes between 0.67 and 0.74 and a difference in the average temperature between 1 and 2 ℃.

However, the results for moving the small file are not necessarily the ones expected (see Figure 31 a)), with a good correlation of 0.77 and 0.79; however, the average temperature of the original command is the same or higher than the one of the augmented one, and I attribute these results to the high probability of caching the small file.

For moving the small file as noise (see Figure 31 b) and c)), I also see not so good results on the other two desktops, with correlation indexes between 0.82 and 0.99 and a small difference in the average temperature.

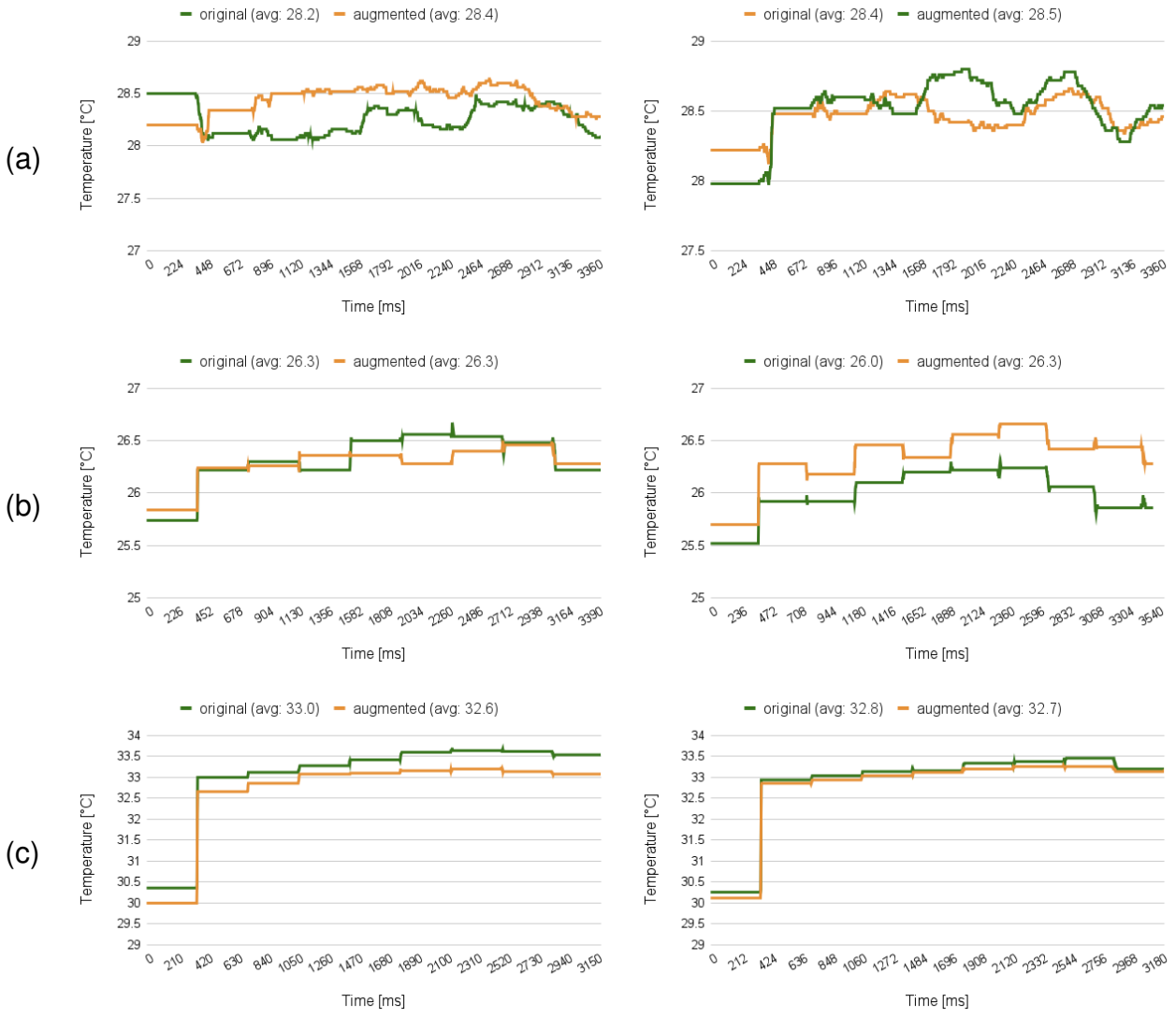


Figure 33: Results with keystrokes as a noise, selecting CPU affinity, a) contains the results for *ls* and *chmod* commands for system *D-150a*, b) contains results for *D-151a* and c) for *D-152a*.

Noise: Browser Surfing In the browser surfing script, I repeatedly open different sites as Google, Google Maps, StackOverflow, Ubuntu, Geeks for Geeks, Wikipedia, or IEEE, using the Firefox browser. The results can be seen in Figure 32 a), where there is a high correlation of 0.85 for the *ls* command and a good correlation of 0.75 for the *chmod* command. For both commands, the average temperature for the augmented command is higher with 0.3 ℃. Therefore, taking into account the high correlation of both patterns and the difference in temperature for the augmented commands, it can be concluded that an anomaly is occurring.

Figure 34: Results with mathematical computations as a noise, selecting CPU affinity, a) contains the results for *ls* and *chmod* commands for system *D-150a*, b) contains results for *D-151a* and c) for *D-152a*.

In the case of surfing the Internet, there are better results for *chmod*, with a correlation of 0.71 and 0.98 and a difference between the average temperatures of up to 1 ℃. For

the *ls* command, the pattern correlation is greater than 0.80 with a smaller difference between the average of the original and augmented command of 0.2 ℃, as can be seen in Figure 32 b) and c).

Noise: Keystrokes The simulation of keystrokes was performed using a text of 123 words. The correlation is between 0.45 and 0.72 for the *ls* and *chmod* command, respectively, as seen in Figure 33 a). Moreover, the average temperature between the original and augmented commands is 0.2 ℃ for the first one and -0.1 ℃ for the second one. These results lead to the conclusion that for the *ls* command, the anomaly can be detected; however, for the *chmod* command it cannot.

For the keystrokes, I again see better results for *chmod* command, with a correlation of 0.88 and a difference in the average temperature of 0.3 ℃, on *D-151a*, as seen in Figure 33 b) and c).

Noise: Mathematical Computations Similar results to those in the keystroke experiment are seen in the mathematical computations in Figure 34 a). Since the mathematical computations as well as the keystrokes were performed in a repeated way, it is possible that the optimization mechanisms, as caching, were applied. Still, the Pearson correlation is a quite high one between 0.77 and 0.79, which leads to the conclusion that the pattern of the command remain very similar in different scenarios, even though the average temperature is higher for the original command. Going forwards, similarly as in keystrokes, these results will not lead to an anomaly detection.

In the case of mathematical computations (see Figure 34 b) and c)), as for keystrokes, better results are obtained for the *chmod* command, where the pattern correlation is higher than 0.88 and the difference in the average temperature up to 0.4 Celsius degrees.

Results without Using CPU Affinity In this subsection, I will present the results while not selecting CPU affinity for the noise, instead letting the OS decide where to run.

Noise: Playing Song For experiments in which the noise consists of playing the *mp3* file while the CPU affinity was not set, it can be seen in Figure 35 a) that the temperature for the augmented command is lower than the original, and the results were reproduced in almost all experiments in which I did not select a specific core for the noise. This happens most probably because the OS used one of the optimization methods. However, the Pearson correlation index has good values, 0.7 and 0.8, respectively, with an average difference between temperatures of 0.3 ℃.

For playing music without selecting which core to run on, I see that the original temperature has higher values than the augmented one, with up to 0.2 ℃, and still the correlation has good values as in Figure 35 b) and c).

Noise: Moving Files In the experiments where the noise was moving a file, without selecting the affinity of the CPU, I used the small-sized file. In terms of correlation, I have a high correlation of 0.88 and 0.93, respectively, which shows a high similarity in the execution pattern, which can also be visually observed in Figure 36 a). Again, the

difference in temperature is 0.2 ℃ for both experiments, where the original command has a higher temperature.

Figure 35: Results with playing song as a noise, without selecting CPU affinity, a) contains the results for *ls* and *chmod* commands for system *D-150a*, b) contains results for *D-151a* and c) for *D-152a*.

Figure 36: Results with moving the small file as a noise, without selecting CPU affinity, a) contains the results for *ls* and *chmod* commands for system *D-150a*, b) contains results for *D-151a* and c) for *D-152a*.

In the case of moving a small file (see Figure 36 b) and c)), I can see good results for the *ls* command, with a correlation index between 0.88 and 0.99 and up to 0.6 degrees Celsius for the difference between the average temperatures.

Noise: Browser Surfing In the experiments where I used surfing the browser as a noise (see Figure 37 a)), I again saw that most of experiments have the original command with a higher temperature than the augmented one. However, Pearson's correlation indices have good values of up to 0.77.

Figure 37: Results with surfing the browser as a noise, without selecting CPU affinity, a) contains the results for *ls* and *chmod* commands for system *D-150a*, b) contains results for *D-151a* and c) for *D-152a*.

The browsing noise without selecting the CPU affinity indicates a higher temperature in the original commands, which leads us to the assumption that there is interference from the cache, as shown in Figure 37 b) and c).

Noise: Keystrokes While using the keystrokes as a noise, I saw better results for the *chmod* command (see Figure 38 a)), with a correlation index of 0.82 and a difference of 1 °C between the original and augmented commands average temperature, something that cannot be seen in case of the *ls* command, where the correlation is 0.58 and the average temperature is the same. In conclusion, the results for the *chmod* command can be used for anomaly detection.

Figure 38: Results with keystrokes as a noise, without selecting CPU affinity, a) contains the results for *ls* and *chmod* commands for system *D-150a*, b) contains results for *D-151a* and c) for *D-152a*.

Better results for key press noise are obtained in the case of *chmod*, which can be seen in Figure 38 b) and c), where the Pearson correlation is 0.94 and the difference between the original and the augmented command is 0.7 ℃.

Noise: Mathematical Computations In the case of noise made up of mathematical computations, just like in the case of keystrokes, better results can be seen for the *chmod* command in Figure 39 a). For *chmod*, the correlation index shows a value of 0.68, while the difference between the average temperature is half a degree Celsius. Although the correlation for the *ls* command is high, almost 0.80, the temperature of the original command is visibly higher.

Figure 39: Results with mathematical computations as a noise, without selecting CPU affinity, a) contains the results for *ls* and *chmod* commands for system *D-150a*, b) contains results for *D-151a* and c) for *D-152a*.

Good results are also found for the mathematical computations on *D-151a* for both commands, with a correlation between 0.89 and 0.94 and a difference in temperature between 0.3 and 0.5 Celsius degrees, as in Figure 39 b) and c).

4.2 WhenThingsHeatUp Method

This chapter addresses the limitations of conventional security mechanisms and further tests the approach that leverages Intel's Digital Thermal Sensors (DTSs) to detect anomalies at the physical layer. I demonstrate that thermal signatures can be effectively exploited for security purposes, even in the presence of environmental noise.

As stated in the previous chapters, *Hot-and-Cold* is a technique that detects anomalies

in Linux commands. However, this technique was used in a controlled environment, with zero noise or user noise, which in fact could not lead to a strong conclusion about the application of the technique in real-life scenarios.

Therefore, this chapter introduces *WhenThingsHeatUp* technique, which builds upon the *Hot-n-Cold* and *Beat-the-Heat* anomaly detection techniques, with the objective of testing it in even more realistic computational environments. This is achieved by the inclusion of nosier, real-world, server environments with varying system loads. This added complexity simulates real-world conditions even more accurately, increasing the robustness and practicality of the anomaly detection method.

Although execution timing patterns remain similar, the augmented version consistently runs hotter, signaling the presence of the injected code. It is a lightweight anomaly detection method, but it was demonstrated only in controlled and quiet environments. The findings validate the ability of the technique to detect anomalies, even in noisy environments. Its robustness under various operational conditions underlines the practical viability of thermal-based anomaly detection in deployed systems. Although still in its early stages, the technique offers a foundation that can be extended and adapted for deployment in practical and industrial contexts. It must be extended to support the continuous monitoring of arbitrary processes and incorporate adaptive learning mechanisms. Integrating thermal data with other low-level system metrics can further enhance detection accuracy and robustness, paving the way for deployment in industrial, embedded, and security-sensitive contexts.

4.2.1 Experimental methodology

Experimental Setup. The experiments were carried out on three systems located in a temperature-controlled room at 20 degrees Celsius. The three desktops will be further called *D-150b*, *D-152b*, and *D-153b*. They contain different architectures of Intel CPUs: *D-150b* has an i7-10700F processor [199], Comet Lake-S architecture on 14nm, and 8 physical cores; *D-152b* has an i7-13700 processor [200], Raptor Lake-S architecture on 10nm, and 16 physical cores (8 performance cores and 8 efficient cores); *D-153b* has an i9-11900K processor [201], Rocket Lake-S architecture on 14nm, and 8 physical cores. All of them operate with Ubuntu 24.04.1 LTS.

Usually, in the experiments, notebooks were avoided because of their packed hardware and many ways of optimization to stay cool and save power. Notebooks exhibit variable frequency and power consumption depending on workload and operating conditions, leading to corresponding fluctuations in the amount of heat dissipated. Furthermore, power consumption strategies differ between power states, such as when running on a battery, charging, plugged in, or fully charged, with each state implementing different power optimization mechanisms to balance performance and energy efficiency. The above leads to unreliable test reproducibility.

In addition, I avoided AMD processors, as their temperature cannot be read from each core; only a general temperature of the CPU can be established. Having only a single temperature sensor on the die limits the ability to detect localized thermal variations, meaning only significant or extreme temperature differences across the die are observable.

The technique is well-suited for server and desktop processors which provide services

and can be targets of attacks. These typically operate at fixed or stable frequencies and under continuous workloads, providing a more consistent thermal and power profile for analysis.

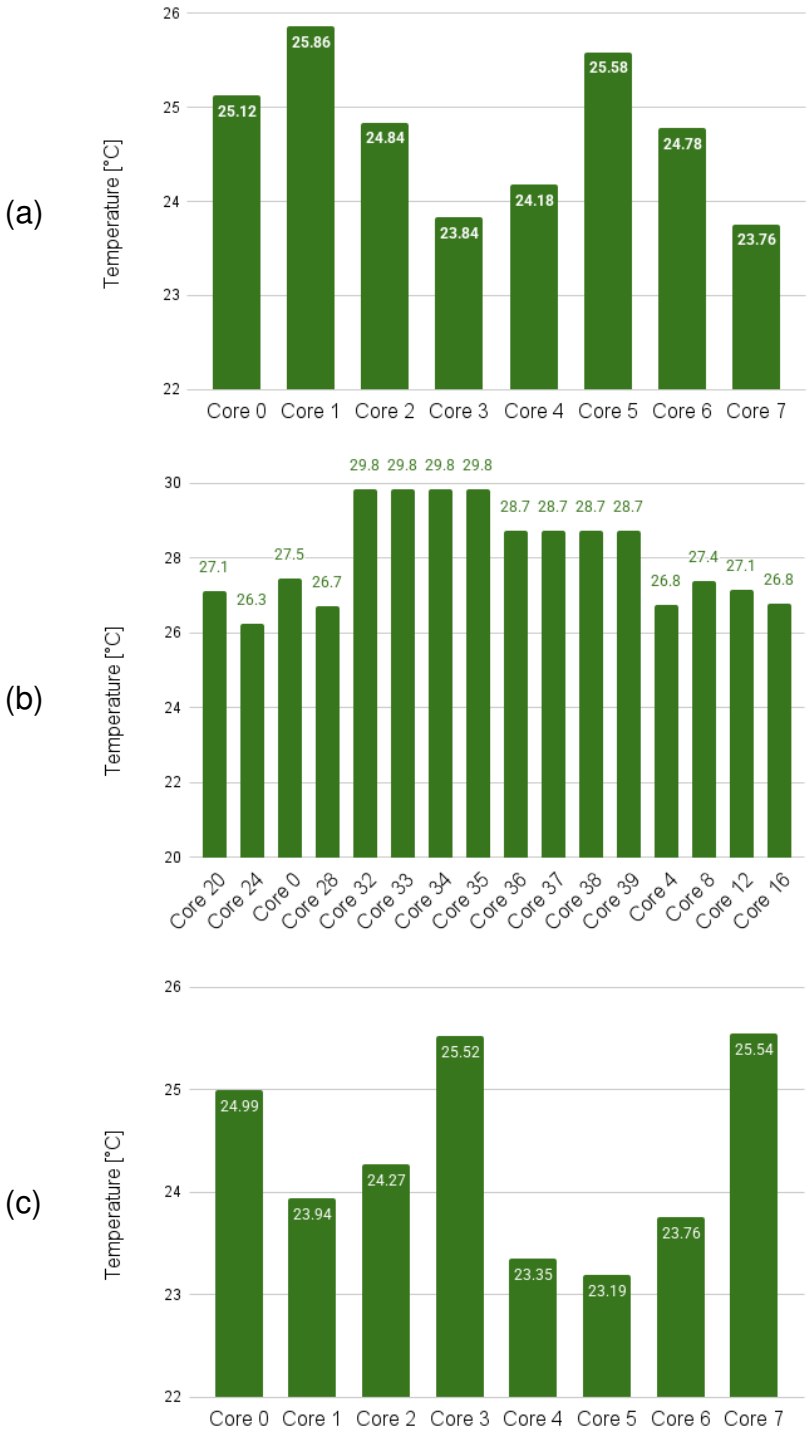


Figure 40: Baseline temperatures of the systems, a) is the baseline for system *D-150b*, b) is for *D-152b* and c) is for *D-153b*.

Baseline Temperature. For each system, the baseline temperature of each core was measured; see Figure 40. This baseline was determined by continuously recording the temperature of all CPU cores over a 24-hour period. Measurements were taken in 5-minute intervals, with readings sampled every 20 milliseconds, followed by a 15-minute pause between intervals. This step is critical to identifying the idle temperature profile of each core under controlled conditions, as stated in Chapter 3.2.9.

In parallel, a comprehensive monitoring of all active daemons and background processes was conducted during a separate 24-hour idle period. This analysis was performed to ensure system stability and to confirm that no background activity would interfere with or bias the thermal measurements collected during the main experimental phase.

Figure 41: Stress on each core of *D-153b*.

Core Selection. For each system, the stress application Stress-ng [202] was used to determine the physical layout of the cores. For each core, the stress application was *taskset* on every core at a time, with a CPU usage of 0.90, while the temperature was recorded. In the created chart, it can be seen which core influences the temperature of which core during the stress experiment. That is, when stress was applied on Core 0 (dark blue), it can be seen in the first part of Figure 41 that the temperature increased to approximately 43 °Celsius. In the same area, the core that has a higher temperature than normal is Core 3 (yellow); Core 7 is not taken into account since it is the one on which the *readTemperature* script was run. This leads to the conclusion that Core 0 (dark blue) is physically located near Core 3 (yellow), and heat dissipates between them. Using this logic, data from charts were extracted as this and created mathematical permutation problems to determine the real physical arrangement of the cores on the die. The possibilities in Figure 42 resulted from knowing that for *D-153b*, the 8 cores are placed 4 cores in two rows [203]. Taking into consideration the lowest temperatures in the baseline charts, together with the layout solutions, I decided to run the *readTemperature* script on Core 0 and *runCommand* on Core 5. Similar experiments were performed for the other two systems.

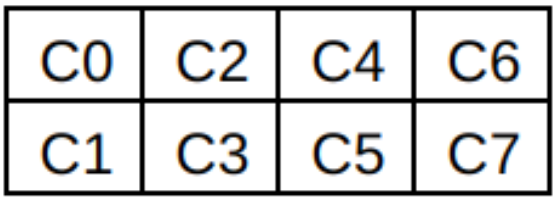


Figure 42: Core layout for D-153b.

The Noise. To simulate various real-world operational conditions, the experimental environment was subjected to various forms of server-like noise intended to simulate typical background activity and load found in multi-user systems.

The Apache [204] server was used to host a simple web page, *png*, *jpg*, and *bmp* images of 1 KB, 480 KB, and 1 MB in size.

The noise was created through the execution of automated client-side scripts developed in both Shell and Python. The Python-based scripts utilize the *requests* library to initiate repeated requests, either to retrieve static image files or to load web pages from the designated server. Specifically, the script responsible for web page access initiates HTTP requests at intervals of 2 milliseconds (see Listing 9), while the image retrieval script issues requests at a frequency of one request per second (see Listing 10). Each script was run by one client at a time.

Listing 9: Request HTML script

```
import requests
server_ip = "xxx.xxx.xxx.150"
url = f"http://{server_ip}"
try:
    response = requests.get(url)
    print("Response from server:")
    print(response.text)
except Exception as e:
    print("Error connecting to server:", e)
```

Listing 10: Download image script

```
import requests
import time
server_ip = "xxx.xxx.xxx.150"
image_url = f"http://{server_ip}/image-1kb.png"
duration = 260000
interval = 1
start_time = time.time()
while time.time() - start_time < duration:
    response = requests.get(image_url)
    if response.status_code == 200:
        with open(f"downloaded_image2.png", "wb") as file:
            file.write(response.content)
```

```
13          print("Image downloaded successfully!-150")
14      else:
15          print(f"Failed to download image. Status code: {response.
                status_code}")
16      time.sleep(interval)
```

To increase the system load and introduce greater thermal and computational noise, a combination of these scripts was used concurrently. That is, five client instances simultaneously accessed the web page and downloaded images. When I refer to the five clients' noise, this means simultaneously requesting the web page and downloading a 1KB image, two 480KB images, and two 1MB images.

For more intensive experimental conditions, the above setup was scaled up to include thirty concurrent client processes, thereby simulating a significantly more demanding environment. That is, the five-client noise was started six times simultaneously.

Evaluation. For the evaluation, I considered that the level of noise introduced in the initial study was insufficient to rigorously assess the robustness and reliability of the proposed technique under more realistic operating conditions. To address this limitation, additional server-like noise was introduced, simulating typical web server workloads. Specifically, noise was generated through server activities by responding to client requests, such as repeatedly requesting a web page, requests to download images of varying sizes, and executing combinations of these tasks using 5 or 30 concurrent clients.

This type of activity leads to a dynamic and non-uniform computational load on the CPU, thereby altering its thermal behavior through fluctuating core utilization, increased cache activity, and more frequent context switches. By incorporating such noise, the goal was to evaluate whether the proposed anomaly detection method remains distinguishable and detectable while the background thermal fluctuations.

4.2.2 Results of the *WhenThingsHeatUp* technique

In this study, the objective was to evaluate the Hot-n-Cold technique in a noisy environment.

First, the original method was reproduced on the three systems. The results can be seen in Figure 43, where green is the original command and orange is the augmented one. It can be seen that the original command has a lower temperature than the augmented one, meaning that the addition in the code consumes more power and dissipates more heat. The Pearson correlation index, over 0.9, shows that the original and the augmented commands have very similar patterns.

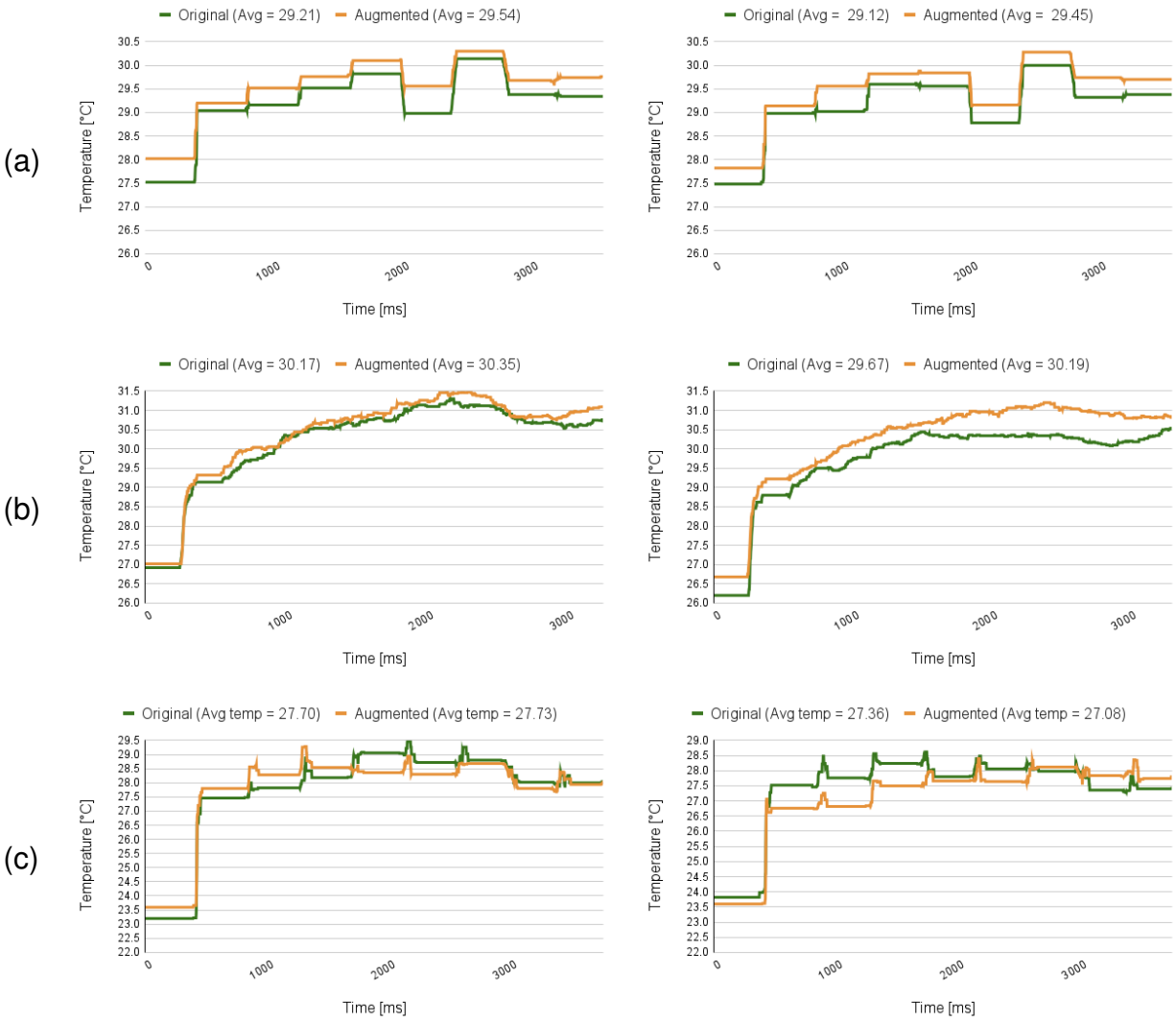


Figure 43: Reproduce original results for *ls* and *chmod* commands, a) contains the results for *D-150b*, b) for *D-152b* and c) for *D-153b*.

Next, server-like noise was introduced. Figure 44 shows the results for when 1KB of data was requested from the server every second. From the graphs, it can be seen that the correlation index is more than 0.9, and the augmented command dissipates more heat. This means that the theory of detecting additional code in a command stands still.

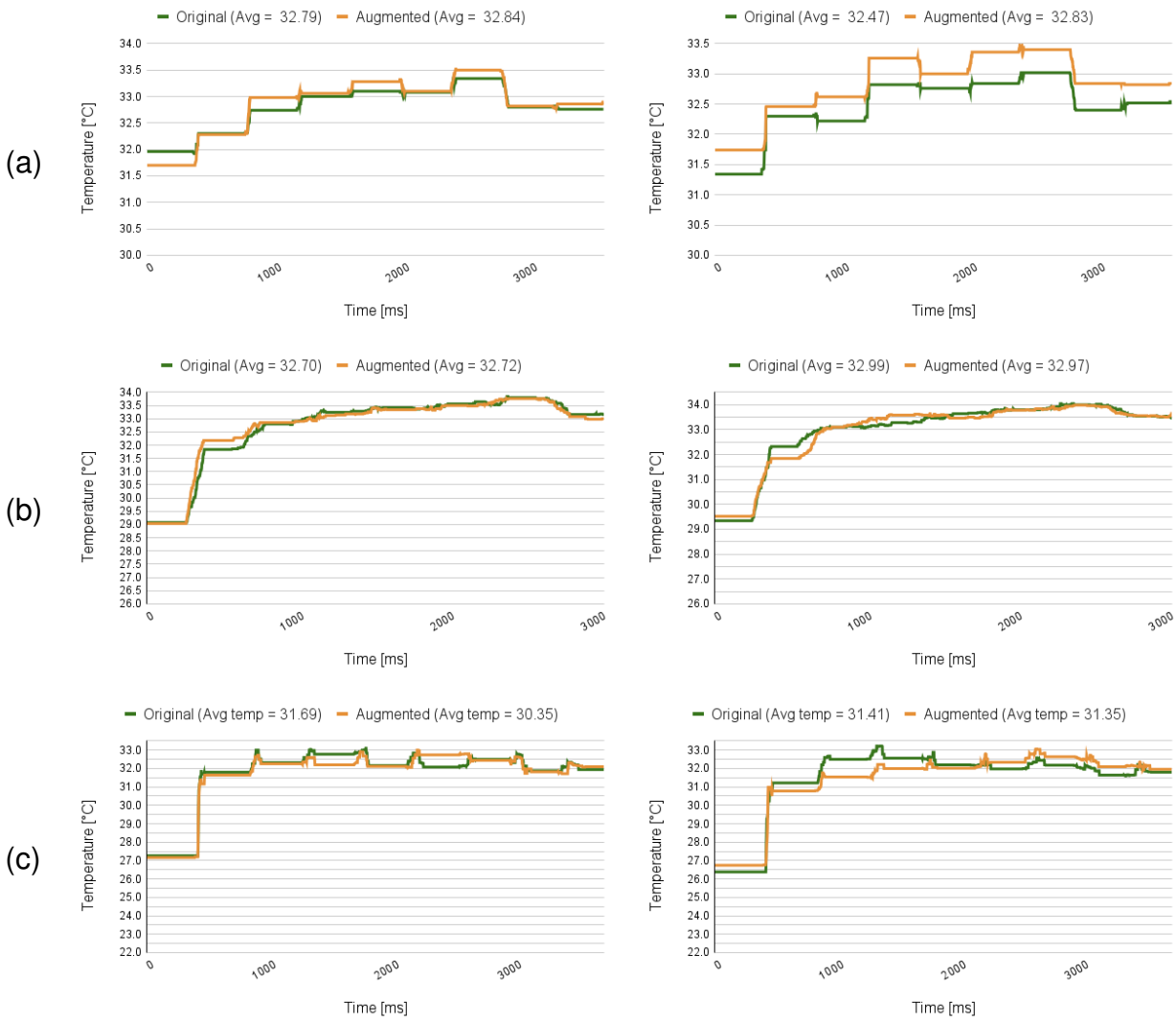


Figure 44: Results with download 1KB data as a noise for *ls* and *chmod* commands, a) contains the results for *D-150b*, b) for *D-152b* and c) for *D-153b*.

Forward, the size of the requested data was then increased to 480KB. As seen in Figure 45, the augmented command dissipates more heat with a correlation index over 0.9.

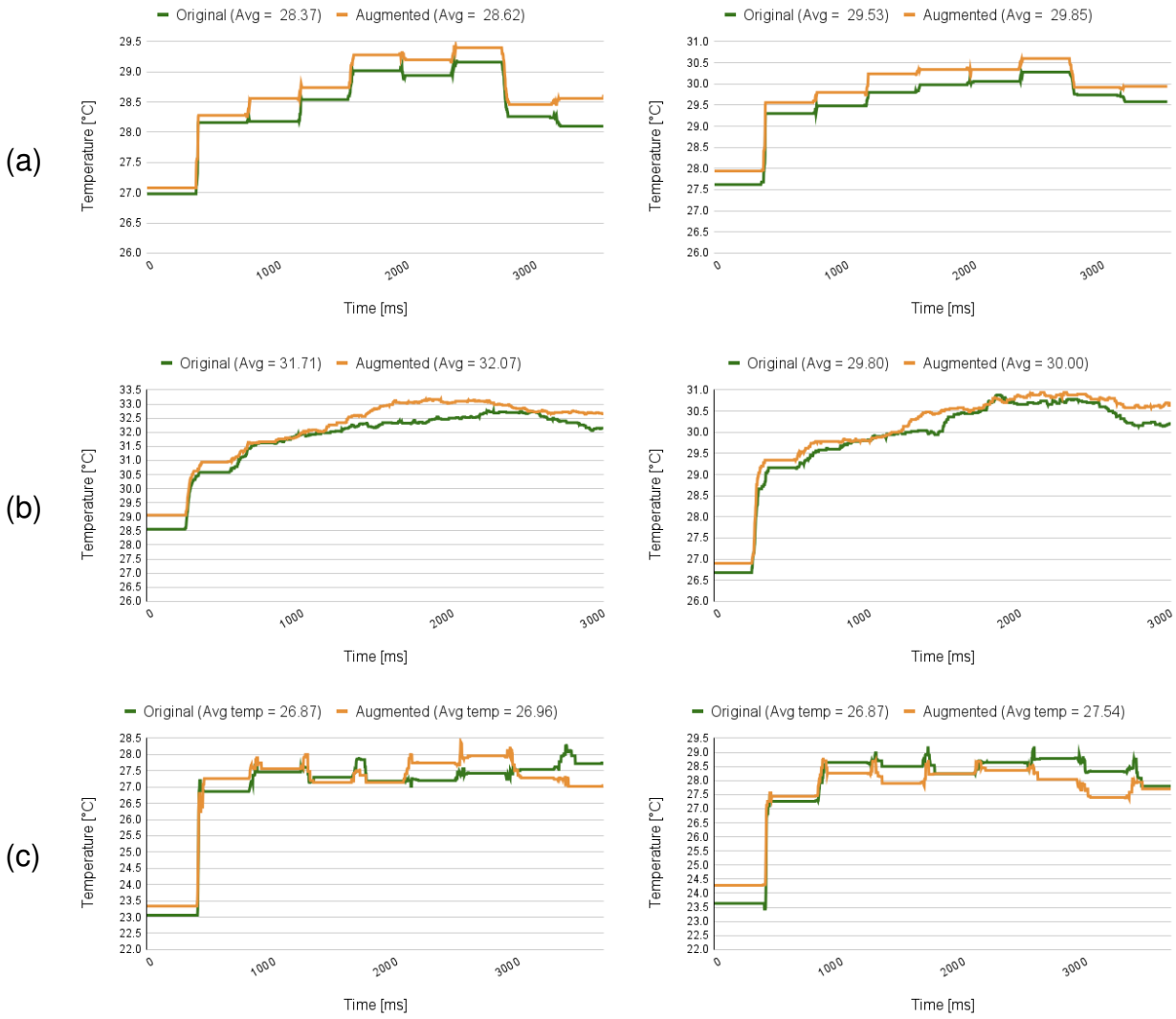


Figure 45: Results with download 480KB data as a noise for *ls* and *chmod* commands, a) contains the results for *D-150b*, b) for *D-152b* and c) for *D-153b*.

Next, the noise was again increased to downloads of 1MB data each second. The results are in Figure 46, where again, the augmented command dissipates more heat and the Pearson correlation is more than 0.9.

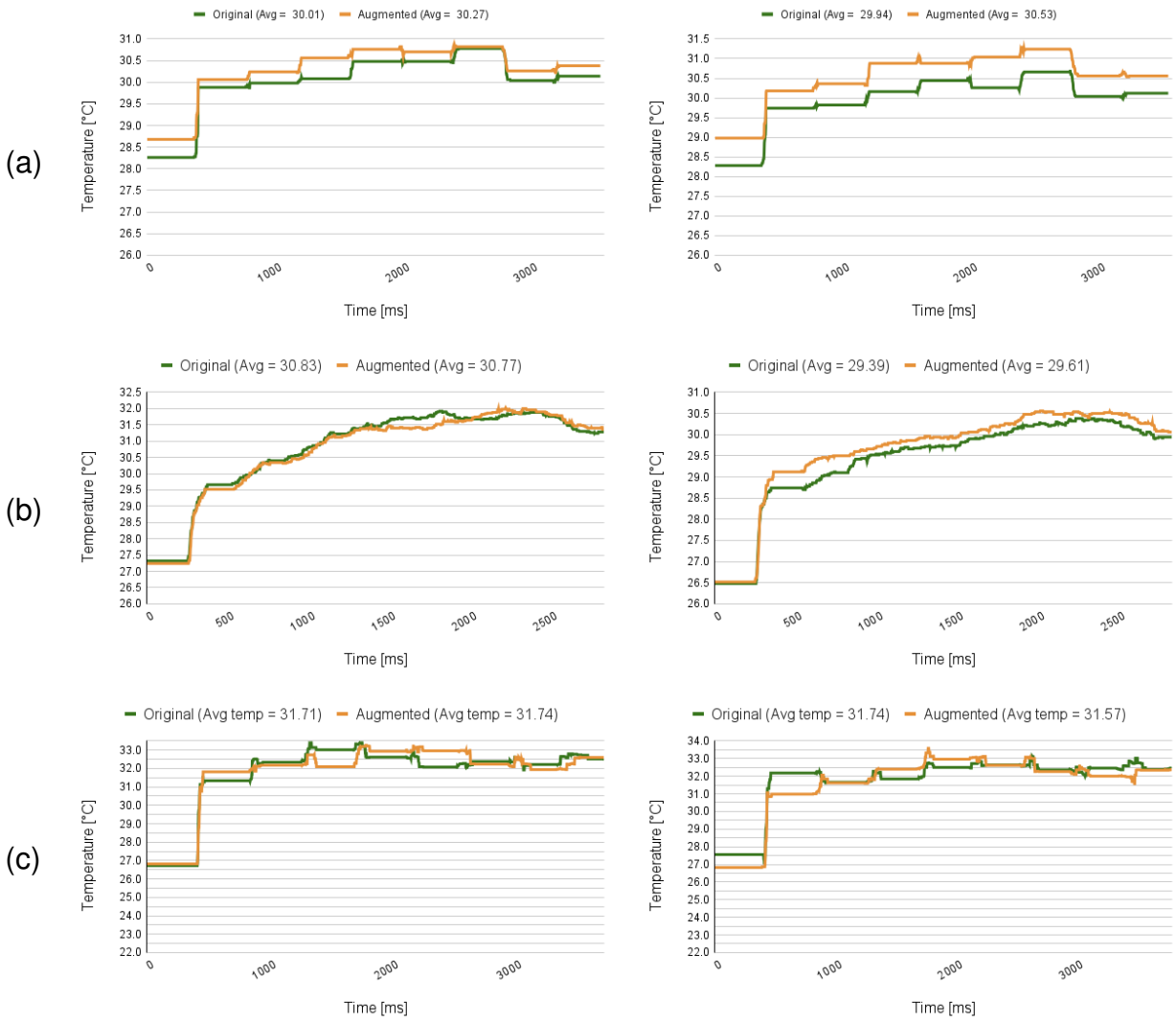


Figure 46: Results with download 1MB data as a noise for *ls* and *chmod* commands, a) contains the results for *D-150b*, b) for *D-152b* and c) for *D-153b*.

For the type of noise with five-clients, the results can be seen in Figure 47, where the rules of above are respected, and the original command can be distinguished from the augmented one. Although the results for D-153b are noisier than those for the others, as seen in Figure 47, the Pearson correlation index remains above 0.9.

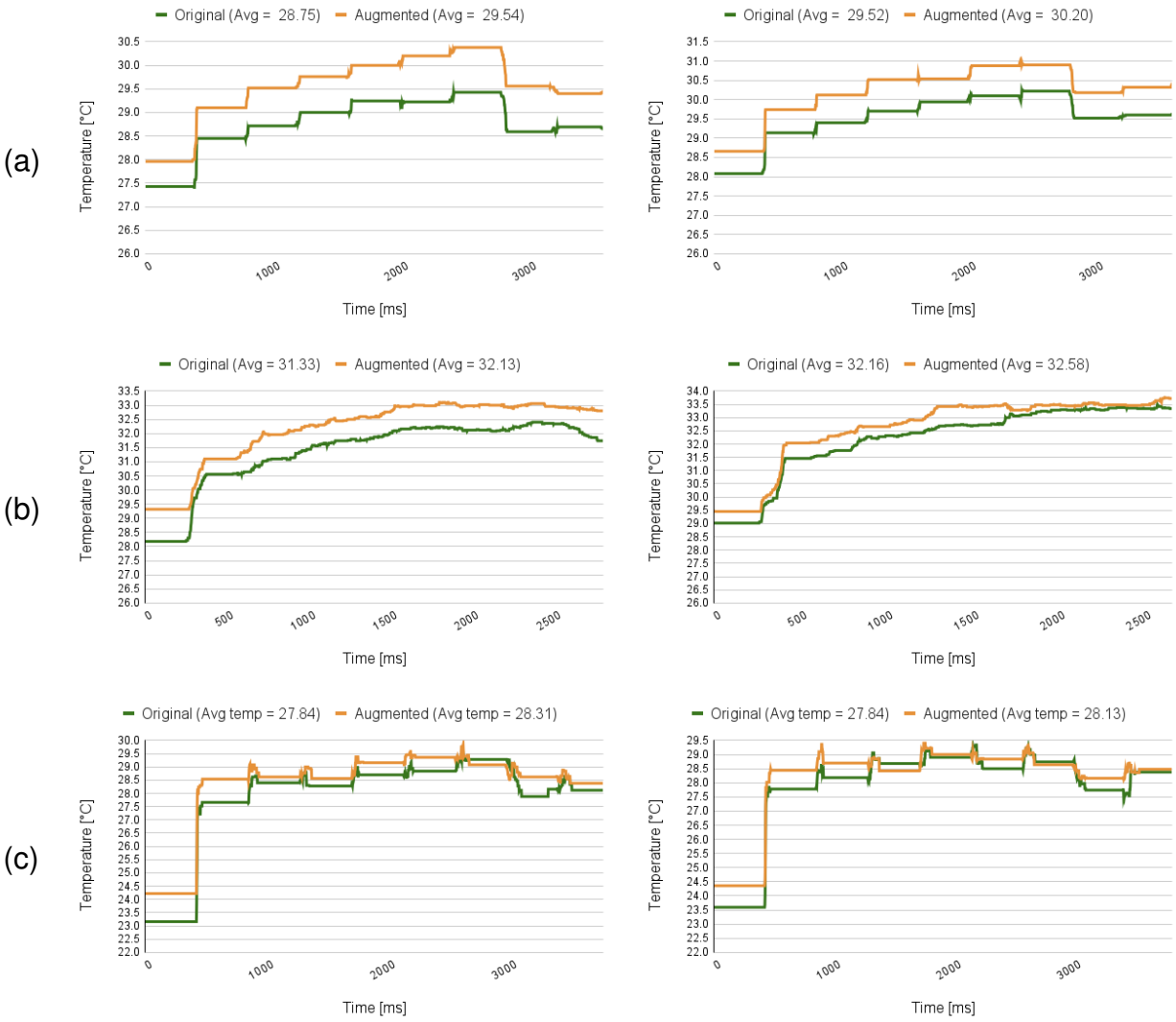


Figure 47: Results with 5-clients noise for *ls* and *chmod* commands, a) contains the results for *D-150b*, b) for *D-152b* and c) for *D-153b*.

4.3 Fingerprinting CPU activities

This subchapter improves the understanding of thermal side channels by establishing a methodology to capture and analyze thermal footprints associated with specific system events, such as cryptographic process execution and unauthorized system logins. I detect and create the thermal fingerprint of cryptographic processes in the CryptoTrooper [205] ransomware attack and thermal footprints for system logins and also detect unauthorized login attempts.

The proposed methods offer several important insights into the application of CPU temperature monitoring to system security. I demonstrate that thermal data, traditionally used solely for hardware protection and performance optimization, can serve as a reliable side channel to identify subtle changes in CPU activity that correspond to malicious behavior, even under challenging environmental conditions.

Through these insights, this study not only confirms the feasibility of thermal anomaly detection but also motivates further exploration into exploiting thermal sensor data for more sophisticated security mechanisms, highlighting the potential of CPU thermal sensors as a low-overhead hardware-assisted layer of defense that augments conventional cybersecurity approaches.

4.3.1 Fingerprinting CryptoTrooper

CryptoTrooper [205] is an open-source ransomware attack for Linux, developed for educational and research purposes, particularly for reverse engineering and cryptographic analysis. It uses a white-box implementation of the AES-128-CBC algorithm to encrypt user and system files, including data stored in common locations such as */root*, */home*, and other directories associated with Apache, MySQL, etc. Each file is encrypted with a randomly generated AES key and initialization vector, which are themselves encrypted using the white-box cipher, producing the auxiliary files necessary for decryption.

To investigate the thermal behavior of cryptographic operations, I executed three commonly used encryption algorithms and monitored their thermal activity throughout execution.

For this experiment, I downloaded the CryptoTrooper Git project [205] and executed the encryption process on the following locations */usr/share*, */var/lib*, */var/www*, */root*, */home-/media*, on all three systems. As encryption algorithms, I used AES-128, AES-256, and ChaCha20, from the OpenSSL library [206].

For each system, 25 traces of encryption have been collected for each algorithm. The experiments were carried out on one core of each processor, first, without noise, and then with noise from 30 clients (described in the subsection *The Noise*). Furthermore, the level of background noise was tripled for the AES-128 encryption algorithm to evaluate whether the increase in interference would affect the thermal footprint associated with the cryptographic operation.

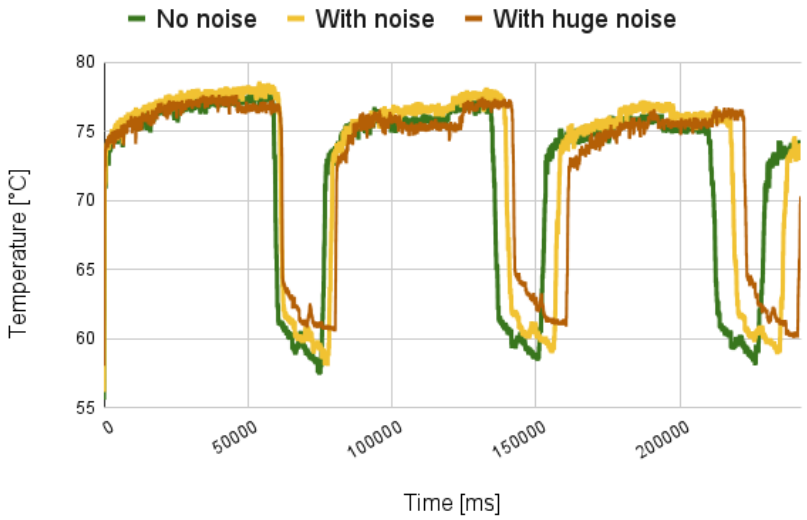
The thermal data collected allowed us to construct thermal footprints for encryption algorithms. These footprints reflect the variation in temperature over time during the encryption process and highlight the unique thermal characteristics of cryptographic algorithms.

Results for fingerprinting cryptographic processes. For each system, I run the cryptographic algorithm in the following scenarios: without noise, with five-client noise, and thirty-client noise, as described in Chapter 4.2.1 paragraph describing the noise. Figure 48 shows the thermal footprint of AES128 on the three systems, where green is the execution without noise, yellow is the execution with 5-client noise, and orange is the execution with 30-client noise (named ”huge noise” in the figure).

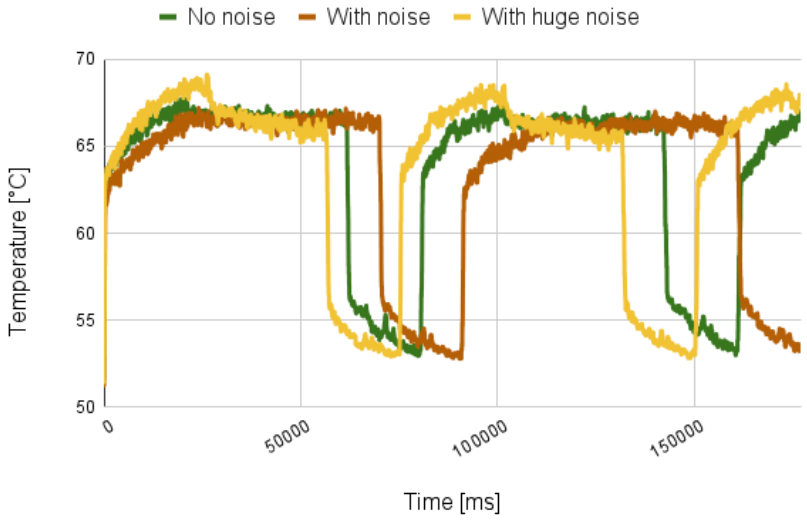
Figure 49 shows the thermal footprint of AES256 on the three systems, where green is the execution without noise, yellow is the execution with 5-client noise (named ”with noise” in the figure).

Figure 50 shows the thermal footprint of ChaCha20 on the three systems, where green is the execution without noise, yellow is the execution with 5-client noise (named ”with noise” in the figure).

(a)



(b)



(c)

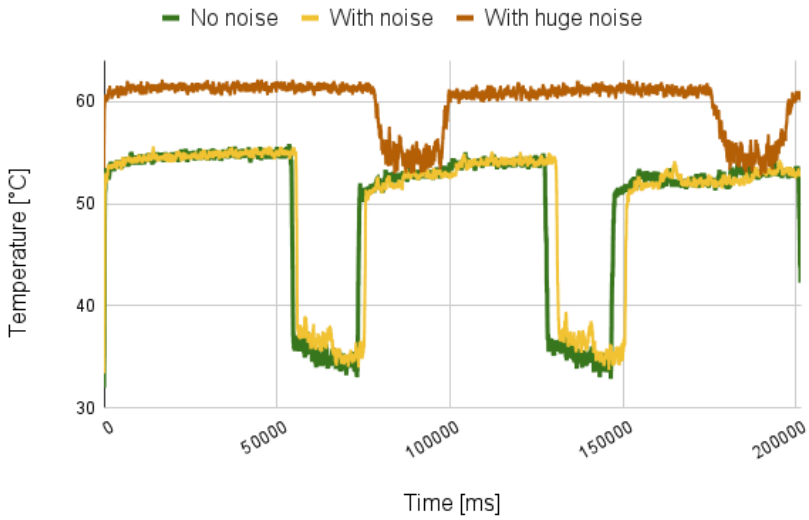
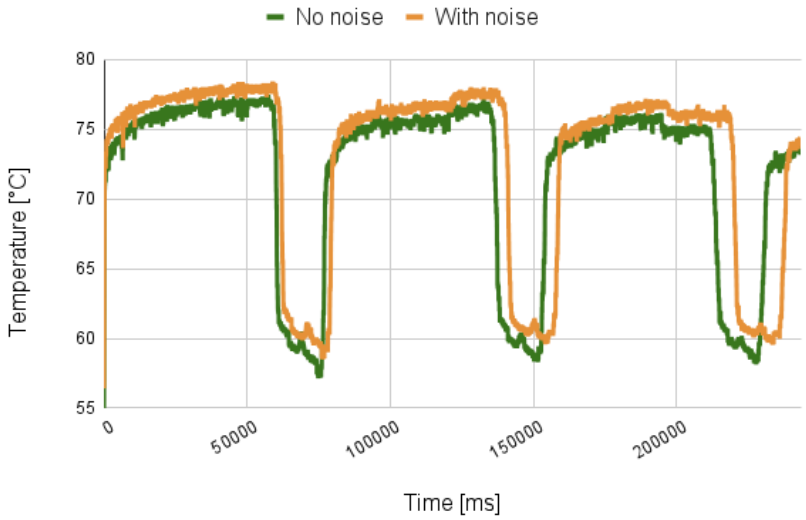
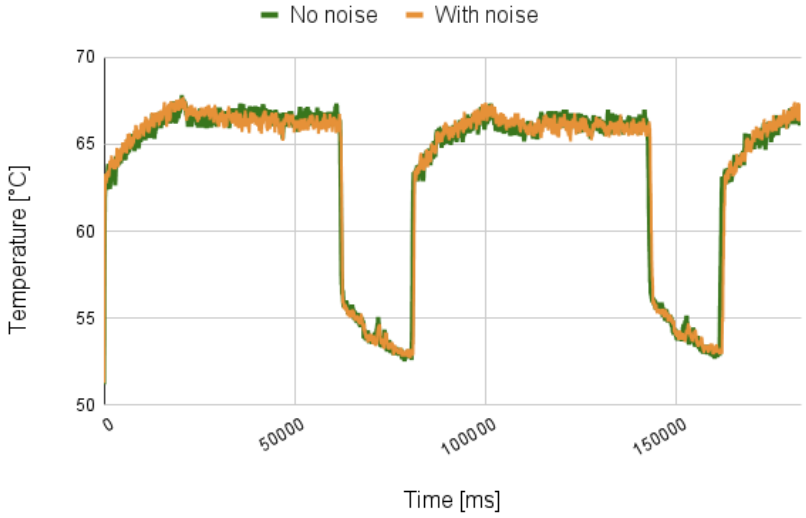


Figure 48: AES128 on the three systems, a) contains the results for *D-150b*, b) for *D-152b* and c) for *D-153b*.

(a)



(b)



(c)

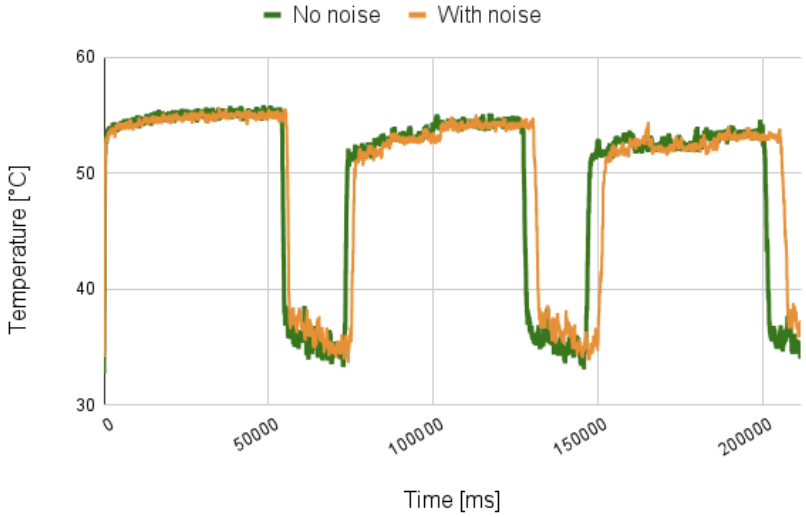
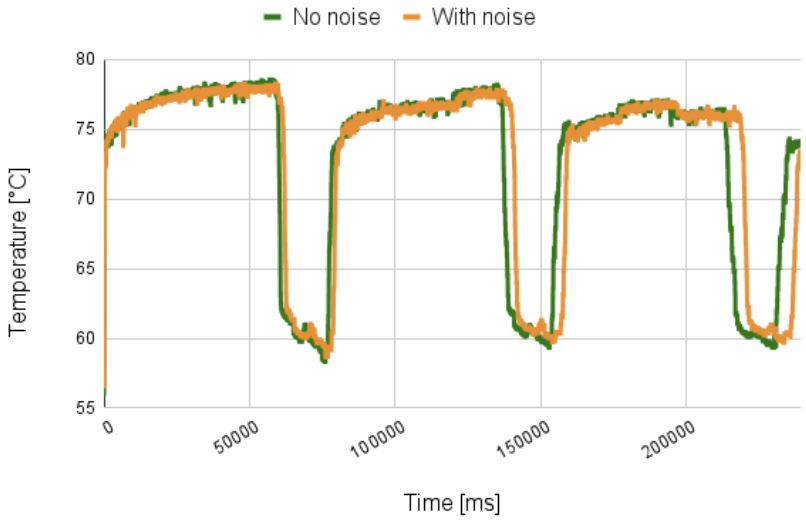
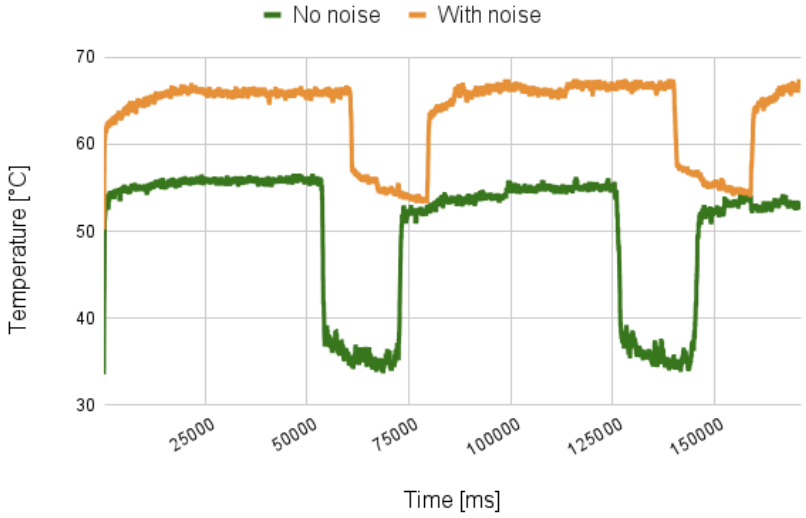


Figure 49: AES256 on the three systems, a) contains the results for *D-150b*, b) for *D-152b* and c) for *D-153b*.

(a)



(b)



(c)

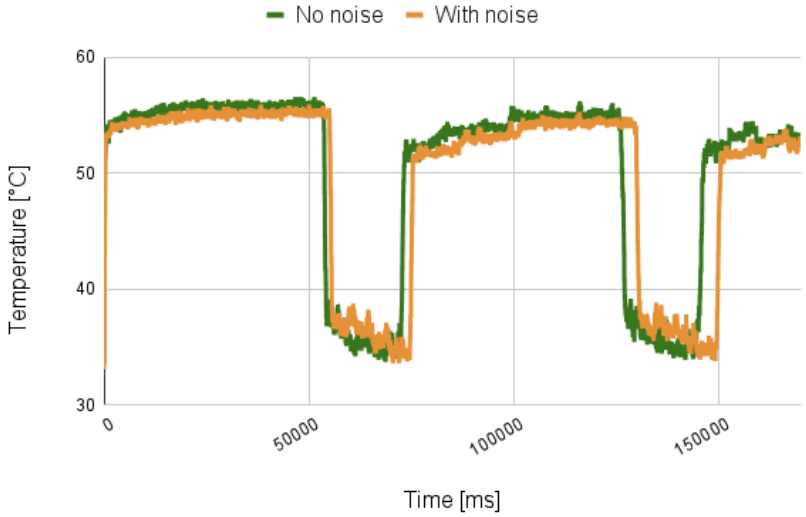


Figure 50: ChaCha20 on the three systems, a) contains the results for *D-150b*, b) for *D-152b* and c) for *D-153b*.

4.3.2 Fingerprinting Authentication Processes

The SSH [207] (Secure Shell) login process is a cryptographically secure protocol designed for remote authentication and command execution over untrusted networks. Upon initiation, the client establishes a connection with the SSH server, which then presents its host key for verification to ensure server authenticity and mitigate attacks. Following successful host verification, a secure key exchange is conducted using algorithms such as Diffie–Hellman or ECDH to derive a shared session key, which is subsequently used to encrypt all communication. The user is then authenticated. Once authenticated, a secure session is established, allowing interactive shell access, command execution, and additional services such as file transfer and port forwarding [208].

The logins were created using Shell scripts. For each system, 30 users were created, named *user1*, *user2*, ..., *user30*, and 30 different 10-character passwords were generated using random.org [209]. The experiments were carried out using various login methods. First, a login was performed for each of the 30 users, with 5 s between them. This process involved launching a new terminal instance for each attempt and logging onto the server via *ssh* using the *sshpass* utility, as seen in Listing 11.

Listing 11: *login* with terminal command

```
gnome-terminal -- bash -c ‘‘sshpass -p ’$password’ ssh -o
   StrictHostKeyChecking=no ${username}@${server_ip}; exec bash’’
```

Second, 30 login attempts were executed for the same user account, with a 5-second pause between attempts. Unlike in the previous case, these logins were performed without initiating a new terminal instance for each connection, as in Listing 12.

Listing 12: *login* without terminal command

```
sshpass -p ‘‘$PASSWORD’’ ssh -o StrictHostKeyChecking=no ‘‘
   $USER@$HOST’’
```

Finally, user switching was performed across the 30 user accounts using the *su* command to transition between them, as in Listing 13.

Listing 13: *su* command

```
spawn su - $username -c ‘‘whoami’’
```

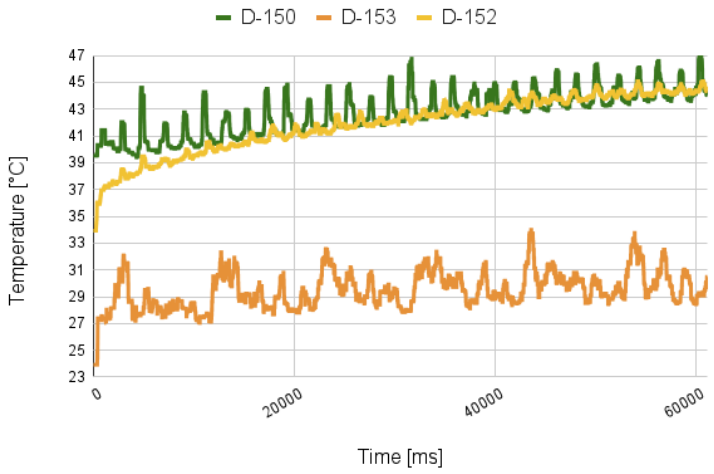
In addition to standard authentication, the accuracy of user passwords was alternated to simulate failed login attempts. Furthermore, server-like background activity was introduced to assess whether such noise would influence the thermal footprint generated during the login process.

The following scenarios were taken into account:

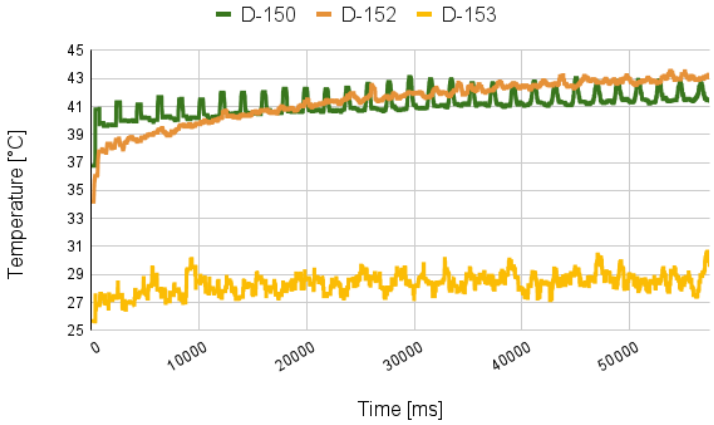
- With noise: one and two clients downloading the 1KB image;
- With noise: from 5 clients (request the web page, download a 1KB image, two 480KB images, and two 1MB images);
- With noise: from 30 clients (6×5-client noise);

- Alternate passwords without noise: one correct and one wrong;
- Alternate passwords without noise: two correct and two wrong.

(a)



(b)



(c)

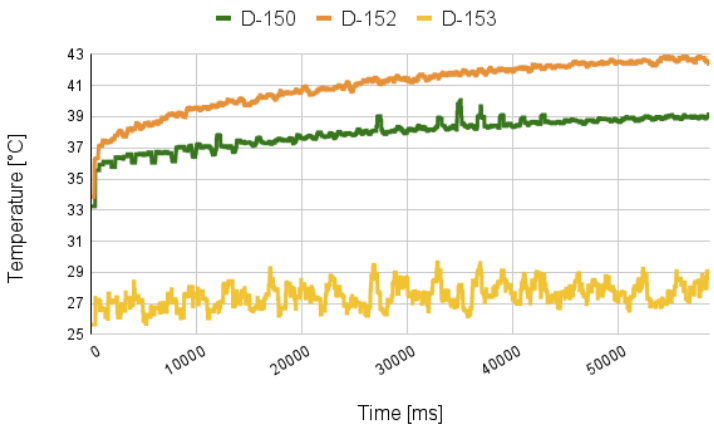
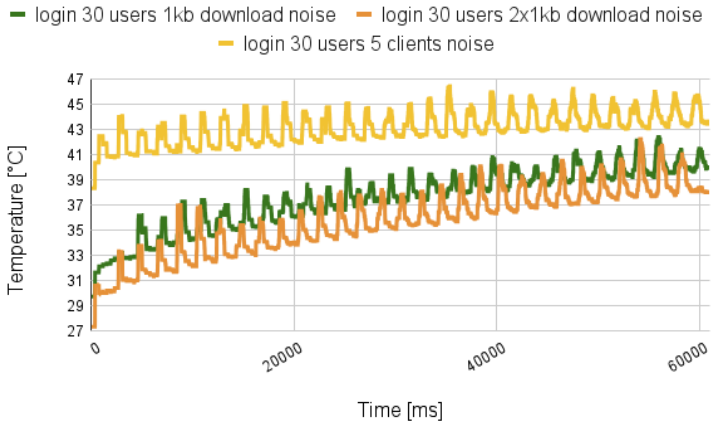


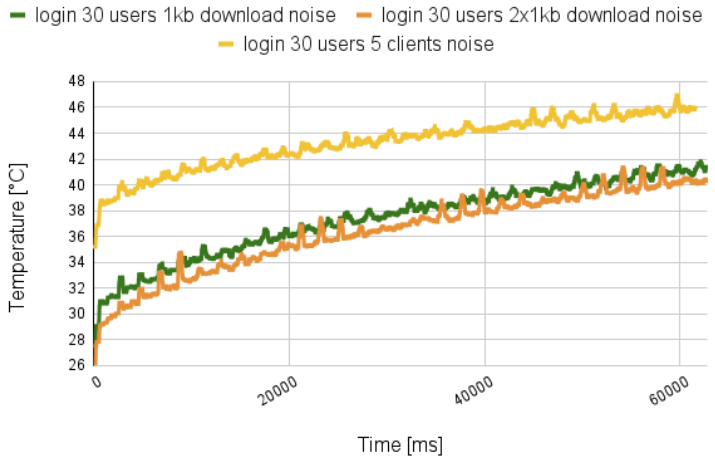
Figure 51: Authentication for all three systems (*D-150b*, *D-152b* and *D-153b*), a) contains the results for login with 30 users, b) contains the results for same user authentication and c) for change user.

Results for fingerprinting authentication processes. To investigate the thermal behavior of the authentication processes, I executed three types of authentication and monitored their thermal activity throughout execution.

(a)



(b)



(c)

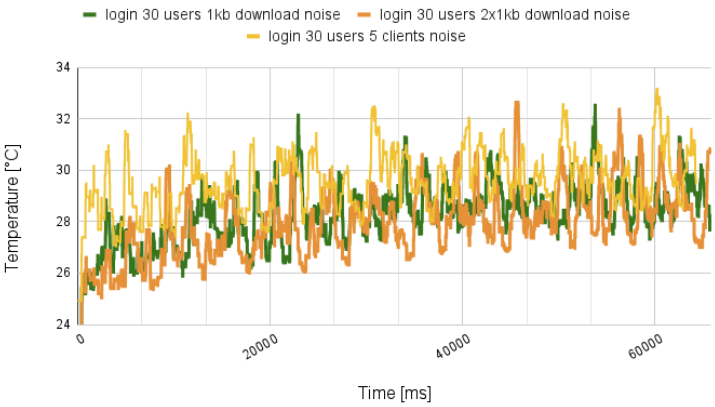
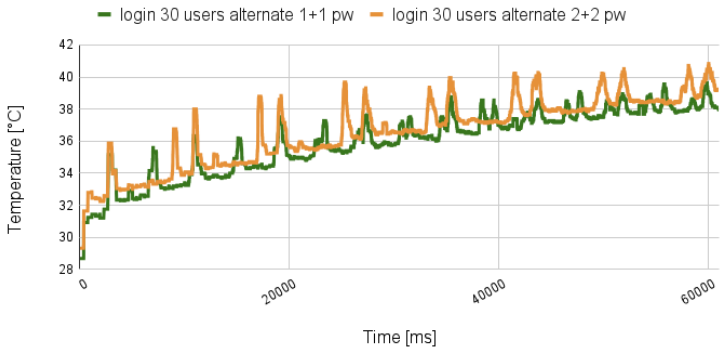


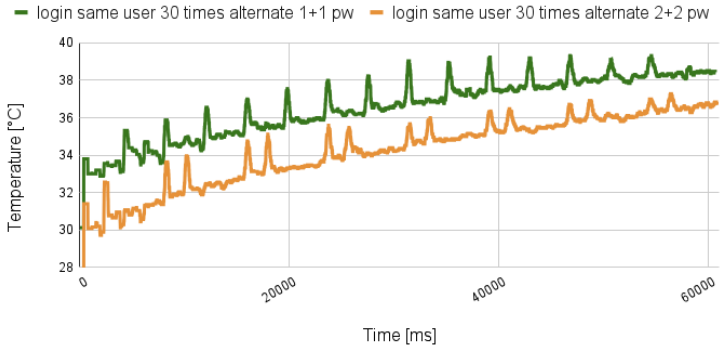
Figure 52: Authentication with noise for both systems (*D-150b*, *D-152b* and *D-153*), a) contains the results for login with 30 users, b) contains the results for same user authentication and c) for change user.

The thermal data collected allowed us to construct thermal footprints of login processes, but not necessarily for the switch-user ones. In Figure 51, the three processes can be seen. For the two login processes (in pink, the logins of 30 users, and in blue, the logins of one user), each login attempt is perfectly seen on the chart as a spike in the temperature. The switch user process can also be seen (in purple); however, they are not as clear as the login ones. Similar results can be seen for D-152. However, I cannot tell the same for system D-153.

(a)



(b)



(c)

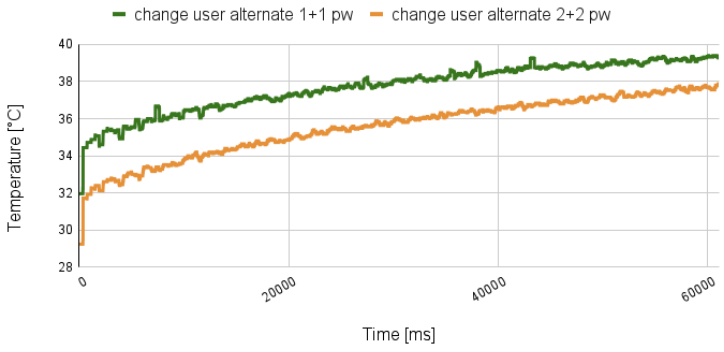


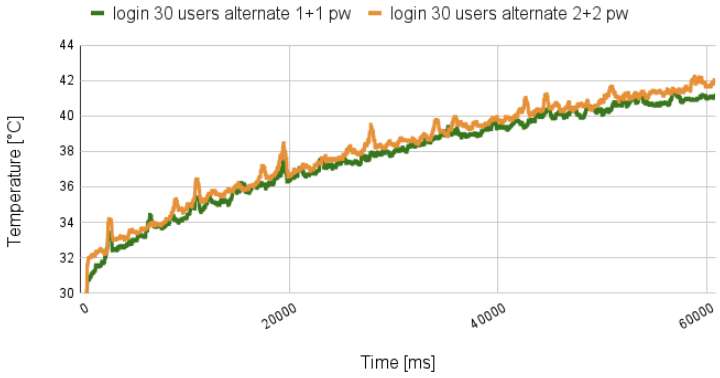
Figure 53: Authentication with alternate passwords on system *D-150b*, a) contains the results for login with 30 users, b) contains the results for same user authentication and c) for change user.

For the login of 30 users with a terminal, I have repeated the experiments using three types of noise. The results can be seen in Figure 52, where each login is seen as a spike in

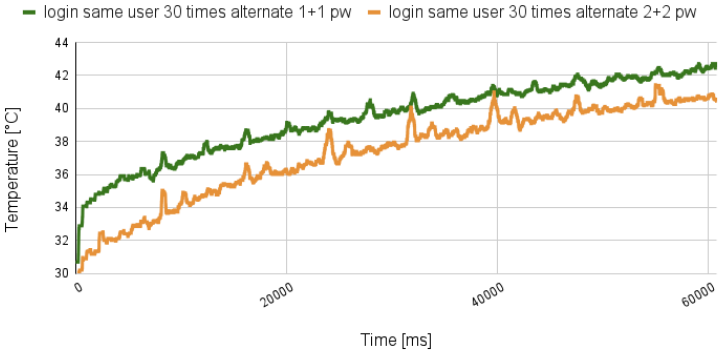
temperature, from which one can tell that the authentication process can be fingerprinted even in noisy environments. However, we cannot observe the same for system *D-153*.

For all three commands, an experiment that alternates passwords has been created. First, a correct password comes after an incorrect password, with green in Figure 53, and second, two correct passwords come after two incorrect passwords, with orange. It can be seen in the figures that there is a spike in temperature in case of authentication with a correct password and a flat line in case the password is wrong.

(a)



(b)



(c)

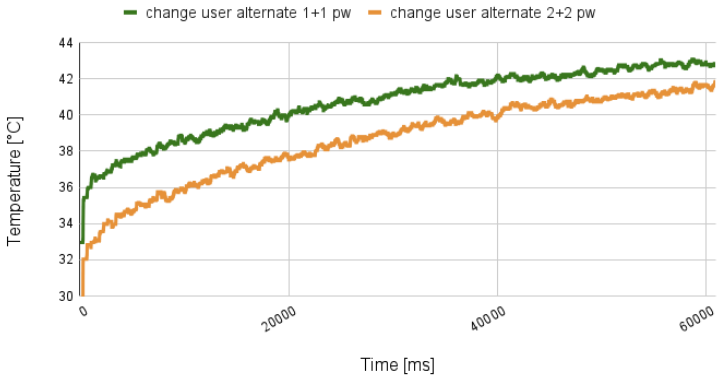


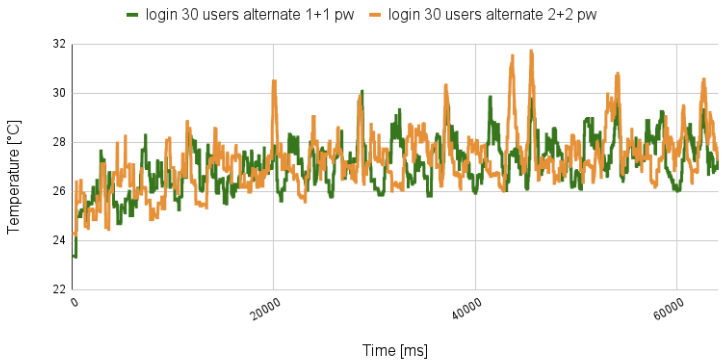
Figure 54: Authentication with alternate passwords on system *D-152b*, a) contains the results for login with 30 users, b) contains the results for same user authentication and c) for change user.

Similar was done for system *D-152*. First, a correct password comes after an incorrect

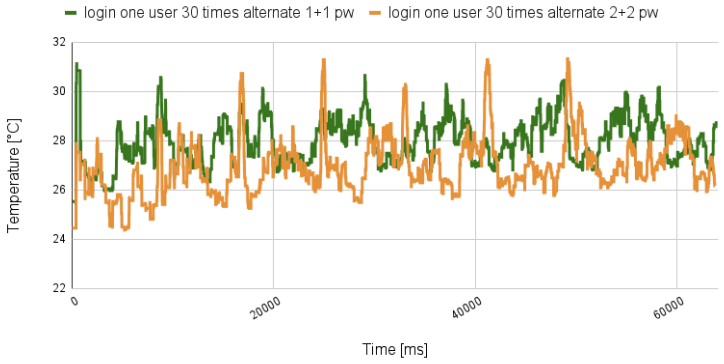
password, with green in Figure 54, and second, two correct passwords come after two incorrect passwords, with orange. It can be seen in the figures that there is a spike in temperature in case of authentication with a correct password and a flat line in case the password is wrong. However, on this system, the results are not as visible as in the above one.

Similar was done for system *D-153* in Figure 55. However, on this system, the results are not as visible as in the above ones.

(a)



(b)



(c)

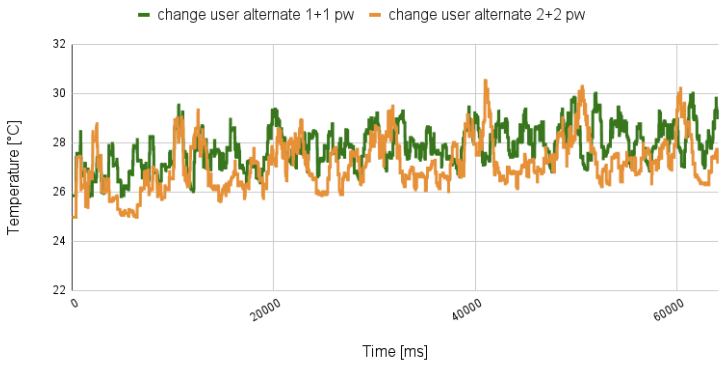


Figure 55: Authentication with alternate passwords on system *D-153b*, a) contains the results for login with 30 users, b) contains the results for same user authentication and c) for change user.

4.4 Discussion

The differences in the implementation and testing phases of the Hot-n-Cold technique can be seen in Table 16.

Table 16: Comparison between previous implementations.

Features Techniques	Hot-n-Cold [164]	Beat-the-Heat [165]	When Things Heat Up (current article)
Methodology	Command was executed 300 times with three different time gaps between calls: 10 milliseconds, 100 milliseconds, and 1000 milliseconds.	Command was executed 600 times with one time gap between calls: 10 milliseconds.	Command was executed 600 times with one time gap between calls: 10 milliseconds.
Noise	No noise	User-like noise: moving files, performing extended math computations, playing songs, browsing the web.	Server-like noise: Download web page and images of 1KB, 480KB and 1MB in size, requested by one or multiple clients.
Number of collected traces	30	50	50
Processor types	Intel Xeon E5-2640 v4	Intel Core i7-10700F, i5-4590, Intel Xeon W3550	Intel Core i7-10700F, i7-13700, i9-11900K (Minimal results: Notebook Intel Core i7-13700H, Desktop AMD Ryzen 7 5700G)
Results	0.90-0.95 Pearson correlation index, 0.5-1 ℃ differences	0.66-0.98 Pearson correlation index, 0.2-2 ℃ differences	0.93-0.99 Pearson correlation index, 0.03-0.8 ℃ differences

4.4.1 Limitations of the Proposed Approach

A CPU temperature sensor with a low resolution of 1℃ limits the precision of thermal monitoring, making it difficult to distinguish the temperature of individual commands; from here comes the methodology of running multiple commands. However, this is not the case for login commands, which can be individually identified. Therefore, I had to create multiple traces of the experiments and use the average of those measurements to increase the accuracy of the temperature readings. However, I did not need tens of thousands of traces as in other studies such as [27]. Moreover, the technique is in the early stages of

development, and it needs to be automated in order to be applied in real-life scenarios.

4.4.2 Differences in Cooling Systems

The *D-150b* and *D-152b* desktop models use air-based (fan) cooling systems, in contrast to *D-153b*, which uses a liquid-based cooling method. The main differences between the two cooling models can be seen in Table 17.

Table 17: Differences between the two cooling systems.

Factor	Fan (i7)	Liquid (i9)
Cooling latency	Higher	Lower
Heat spread	Localized	Diffused
Cooling speed	Slower to react	Faster and dissipates heat more evenly
Temperature spikes	Allow sharp curves in temperature	Flattens temperature spikes
Background tasks	Fewer	More
Temperature signal	Clear	Noisy

From the table above, the combination of the fan cooling system with i7 processors leads to more precise thermal signatures in the traces, while the i9 processors are more powerful and need a more reliable cooling system. Therefore, the cooling based on liquid brings more noise in the thermal traces and the heat is diffused, making sharp curves in temperature disappear. However, I believe that if the number of traces used in the experiments is increased, the thermal footprints can be extracted from the i9 as well.

4.4.3 Hot-n-Cold Evaluation

Looking at the above results, it can be concluded that the Hot-n-Cold technique works in noisy environments and on several types of processors as well.

Furthermore, two options were used for the *ls* command, namely *-l* and *-R*, to check if the hypothesis of the technique remains true in more cases. As can be seen in Figures 56 and 57, the augmented commands dissipate more heat, and their patterns remain similar, with a Pearson correlation index between 0.89 and 0.99.

To further expand the investigation, the experiments of *Hot-n-Cold* technique were also run on a Notebook with an Intel Core i7-13700H and on a desktop with AMD Ryzen 7 5700G. Usually, in experiments, the use of notebooks was avoided because of their noisy and unstable results. Nevertheless, some short experiments were performed, as seen in Figure 58, where the Pearson correlation index is greater than 0.66 and the augmented command dissipates more heat. Large differences in temperature were also observed in cases where the laptop was on battery, charging, or fully charged, leading to inconclusive results.

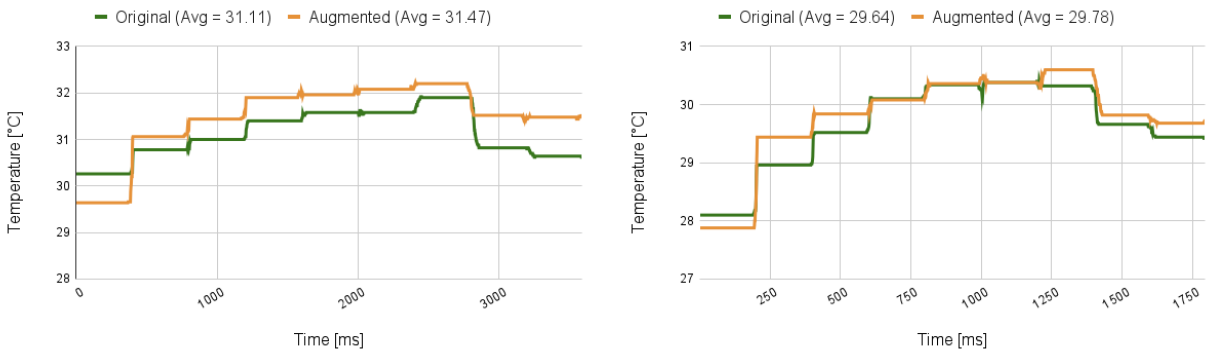


Figure 56: *ls* with options on *D-150b*

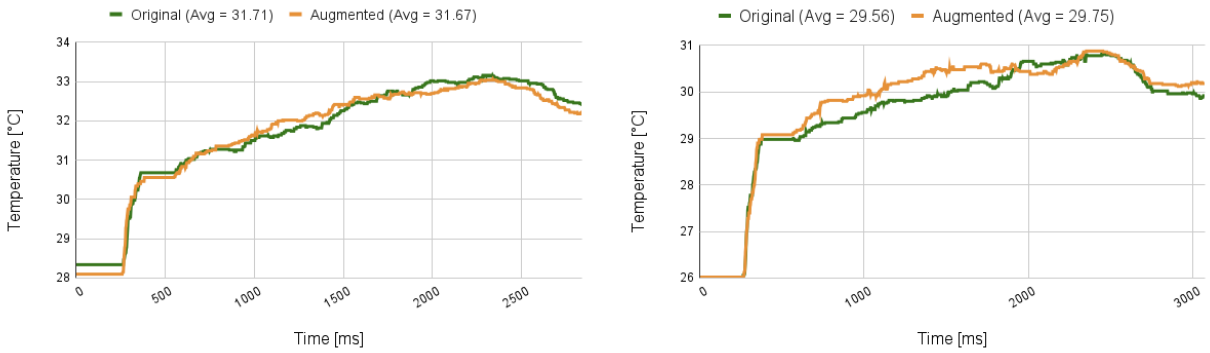


Figure 57: *ls* with options on *D-152b*

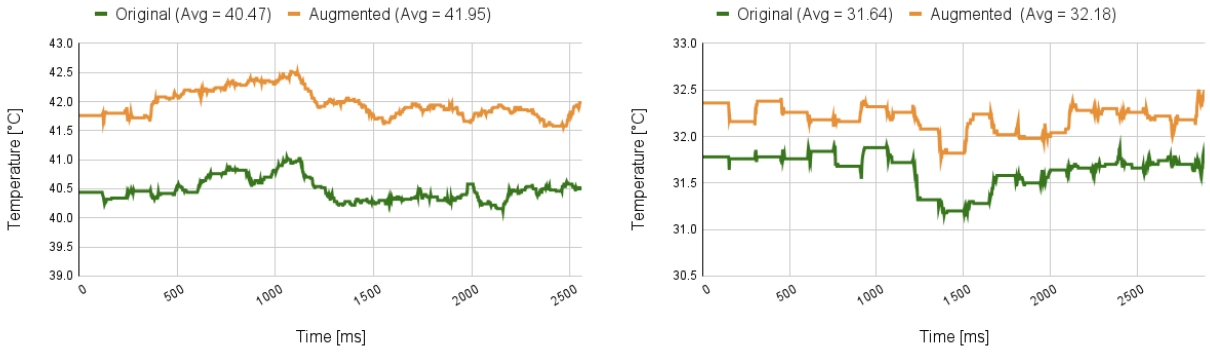


Figure 58: Results for *ls* and *chmod* on the Laptop.

If AMD is taken into consideration, for these processors, the temperature from each core cannot be read; only a general temperature of the CPU can be read. This is how the experiments were conducted. As seen in Figure 59, the results are very noisy, and fine-grained thermal traces cannot be extracted, again leading to inconclusive results.

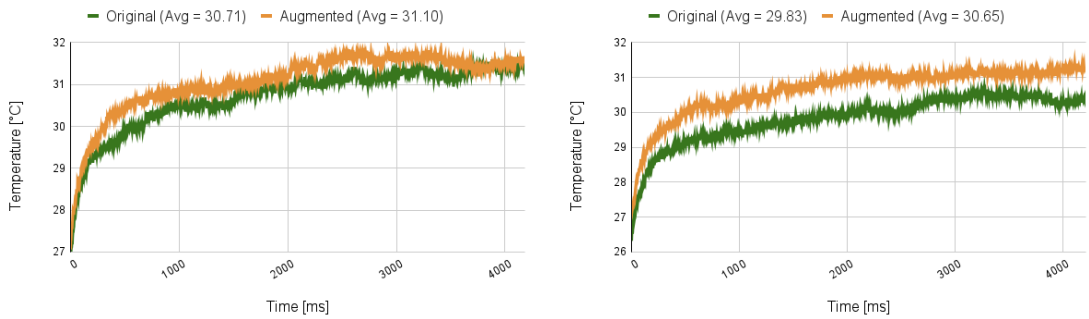


Figure 59: Results for *ls* and *chmod* on the AMD desktop.

(a)

(b)

(c)

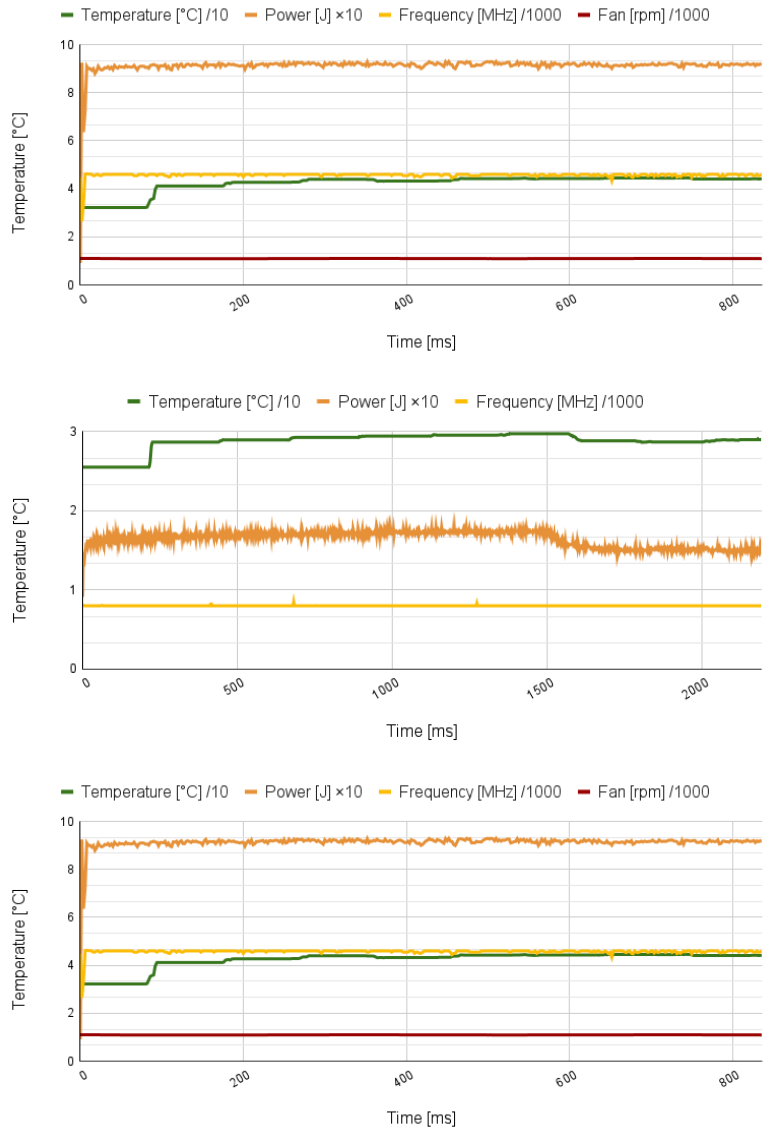


Figure 60: Power, frequency, and fan measured along temperature, a) contains the results for system *D-150b*, b) for *D-152b* and c) for *D-153b*.

Moreover, I saw that on *D-153b*, there are some visible spikes in the temperature, as seen in Figure 43. To identify their origin, a small experiment was performed in which the power, frequency and fan (only on *D-150b*) were read together with the temperature, while the *ls* command scripts were running. The results can be seen in Figure **??**, where pink is the frequency, blue is the temperature, and purple is the power. It can be seen that, during the execution, all of them remain stable, with no noticeable increase or decrease. This leads to the conclusion that the spikes are created by using multiple traces for the final result.

In addition, to measure the technique in more quantitative detection metrics, the detection rate (True Positive Rate - TPR) was calculated using Formula 1, and the miss rate (False Negative Rate - FNR) using Formula 2, where True Positives (TPs) were took as the number of runs where injection was detected and False Negatives (FNs) as the number of runs where injection was missed. The results can be found in Figure 61, with a detection rate of 83%.

$$TPR = TP/(TP + FN) \tag{1}$$

$$FNR = FN/(TP + FN) \tag{2}$$

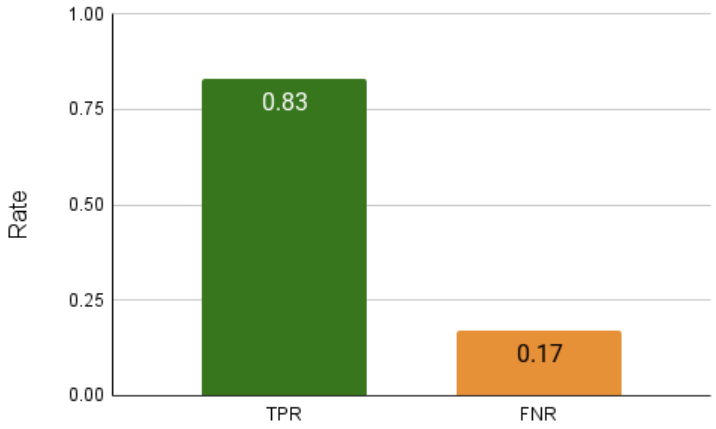


Figure 61: Detection rate for Hot-n-Cold.

4.4.4 Fingerprinting CryptoTrooper

The analysis revealed similarities in the thermal profiles over time. These results demonstrate that encryption processes leave thermal traces and that cryptographic algorithms produce similar thermal signatures even in noisy environments. This information helps us detect when a cryptographic process is running; therefore, it is possible to detect CryptoTrooper attacks using thermal traces.

4.4.5 Fingerprinting Authentication Processes

To discuss the differences between the different authentication commands, we present some comparison tables next. Table 18 presents a summary of the difference between the two *ssh* commands, first with a terminal and second without a terminal, for the authentication process.

Table 18: Differences between the two *ssh* authentication commands.

Feature	***ssh* with terminal**	***ssh* without terminal**
Opens a new terminal window?	Yes	No
Launches full user shell (e.g., bash)?	Yes	Only minimal shell setup
Allocates a full terminal?	Yes	No (unless forced manually)
Can accept multiple commands?	Yes	No
Stays connected?	Yes, until you exit	No, disconnects automatically
Resource usage	Higher	Lower

In Table 19 presents a summary of the difference between the *ssh* and *su* commands for the authentication process.

Table 19: Differences between the *ssh* and *su* authentication commands.

Feature	***ssh***	***su***
Encrypts connection?	Yes	No
Sends password over network?	Yes, encrypted	No
Exposed to network attacks?	Yes, if misconfigured	No, only local risks
User environment?	Full remote	Local user
CPU Usage	Higher, due to encryption/decryption	Very low, just basic hash computation
Memory usage	More, holds encryption keys, session buffers, etc.	Minimal, only password in memory temporarily
Time delay	Higher, due to encryption + authentication handshake	Fast, only local password check

From the tables presented above, it can be seen that the *switch user* authentication process consumes very few resources; therefore, in thermal traces, the login attempts cannot be seen as in *shh* cases where resource usage is higher.

In addition to the differences in cooling presented above, another reason why authentication could not have been fingerprinted on the *D-153b* is a distinction in configuration. To set the login processes to run on a specific core, the PAM (Pluggable Authentication Module) and Slice CPU Affinity settings were used. On the i7 systems, PAM was used to pin the login session to a specific core before any execution starts. However, on the i9, I was unable to use PAM; therefore, a login slice was created, which sets the CPU affinity only after a part of the login session was initialized. This means that a part of the login session was run somewhere else, minimizing the temperature dissipation on the monitored core.

4.5 A Closer Look at How CPUs Work

4.5.1 Cooling down of a core on the *miniHPC* system.

An experiment was conducted to see how fast a CPU core cools. In particular, the *read* function was called repeatedly, which uses the *read* syscall. Pauses of different time lengths were injected between function calls, to observe how much time the respective core needs to cool down. In Figure 62 one can see that introducing a pause of 0.85s in between function calls, the temperature decreases by one degree Celsius. Furthermore, by introducing a pause of 0.9s in between function calls, the temperature decreases by 2 degrees Celsius. Nevertheless, introducing a pause of 0.8s leads to a core maintaining its original temperature and not cooling down at all.

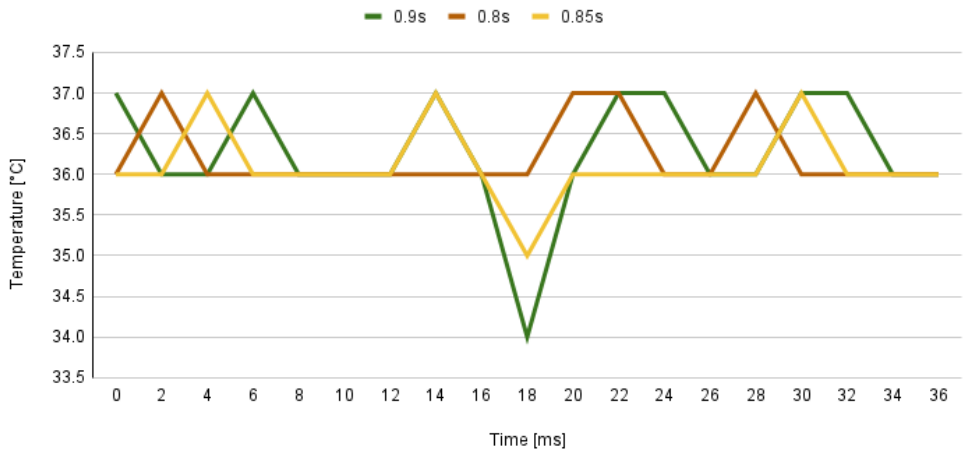


Figure 62: Cooling down of a CPU core when running the *read* Linux command with various time-gaps.

4.5.2 Rising and cooling down of a core on a desktop system.

To further test the capabilities of the CPU sensors, another experiment was conducted on the increasing and decreasing temperatures of a core on the *D-150a* system. More specifically, a CPU stressing program was used, Stress-ng [210], which increases the CPU temperature to a high one to the CPU load of 70%. The temperature was read every 500 ms to see how fast it increases and decreases after the program is finished.

In Figure 63, on the horizontal axes, the readings are performed once every 500 ms. Thus, in Figure 63 a) temperature starts at a 24 ℃ and in half a second the temperature rise to 34 ℃, which means 10 degrees Celsius in 500 ms. Furthermore, in 1.5 s, the temperature increases by a total of 18 ℃. In the case of cooling down in Figure 63 b), temperature started at 40 ℃, and in 1 s, the temperature drops with only 2 ℃. However, after another second, the temperature drops with 11 ℃, which leads to the conclusion that the cooling down is slower than the heating up.

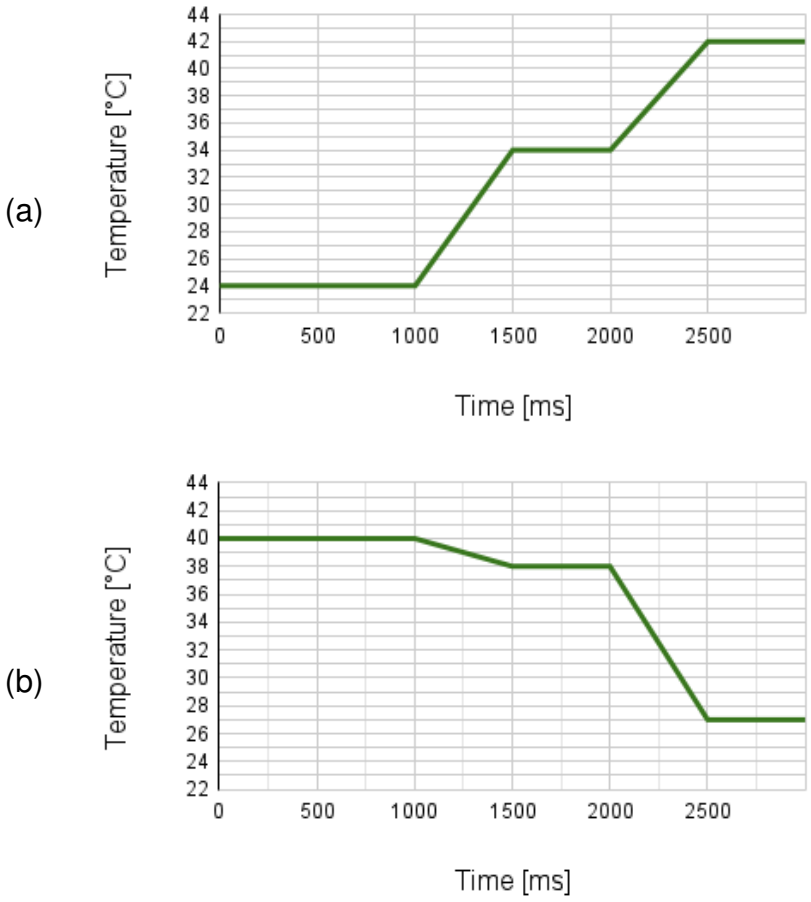


Figure 63: Rising a) and dropping b) temperature of a core.

4.5.3 On which core the OS runs

Furthermore, I wanted to know on which core the operation system runs. However, I did not find any information in the literature; therefore, I created an experiment made of repetitions of restarts and checking the temperature of the idle system to see whether there are any differences in the temperature, leading to a conclusion for the above question. I restarted the system 15 times and created the baseline temperature of the cores after each restart on system *D-150a*. After the analysis of the temperatures, the results can be seen in Figure 64, and it can be concluded that Cores 1 and 5 always have the highest temperature, whilst Cores 0, 2, 4, and 6 have medium temperatures, and Cores 3 and 7

always have the lowest temperatures. Consequently, I can conclude that the OS will start every time on the same cores.

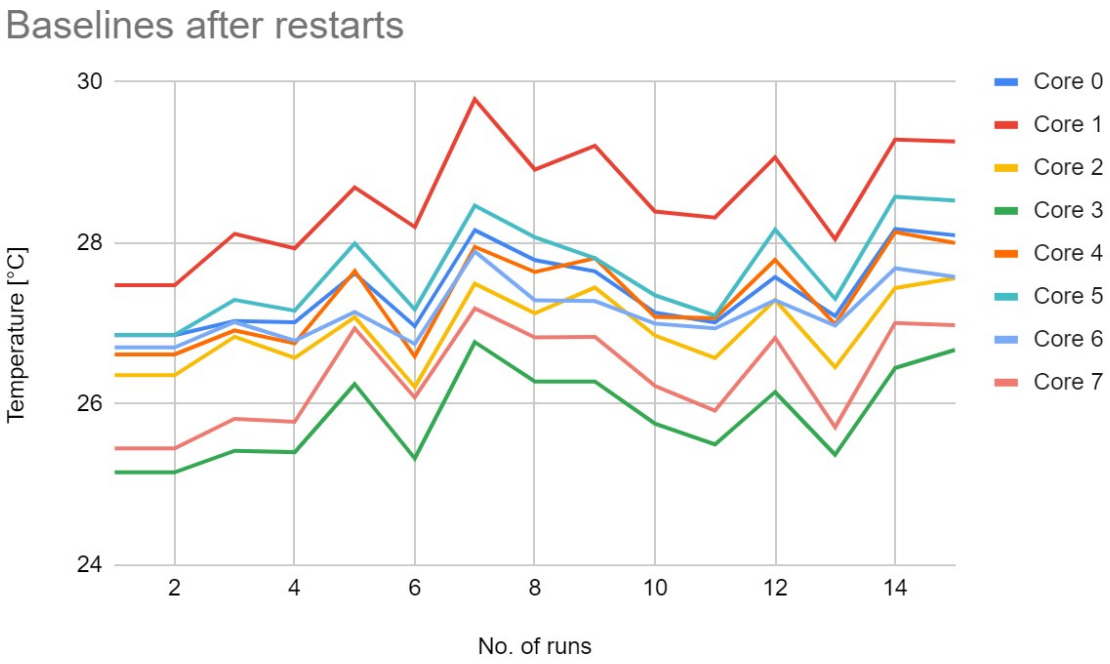


Figure 64: Baselines after restarts of the system.

4.5.4 Heat propagation

Taking into consideration the heat propagation and the cores on which the scripts are running, these two together can affect the performance of the anomaly detection. Thereby, an experiment was conducted to further see how the heat propagates through cores and analyze the physical locations of the cores. The stressing CPU program *Stress-ng* [210] was used on Core 7 of the *D-150a* system to increase the CPU load to 70%. The temperature was read with a script running on Core 0, placing the two scripts as physically far as possible on the CPU. In Figure 65, the difference between the baseline temperature of the respective core and the temperature measured can be seen, for each core, while in Core 7, the stress program is running. Therefore, it can be observed that all cores are affected by the high heat (of approximately 50 ℃ on the peak) of Core 7.

To dig further into the heat propagation topic, we will analyzed it further. On Core 7, the temperature increases with approximately 12 ℃, and the next temperature range is between 5.1 and 5.7 ℃ and includes Cores 0, 4, 5, and 6, and the last one is between 4.2 and 4.7 ℃, which includes Cores 1, 2, and 3. Thereby, the most affected cores are Cores 0, 4, 5, 6, from which I will exclude Core 0, because the script for collecting the temperature is raising its temperature for sure. Thus, I remain with Cores 4, 5, and 6 as the most affected cores by the stress program running on Core 7.

Taking into consideration what was stated above, I can assume that the layout of the cores on the CPU die is as seen in Figure 66, where Cores 4, 5, and 6 are the hottest because these are physically placed near Core 7.

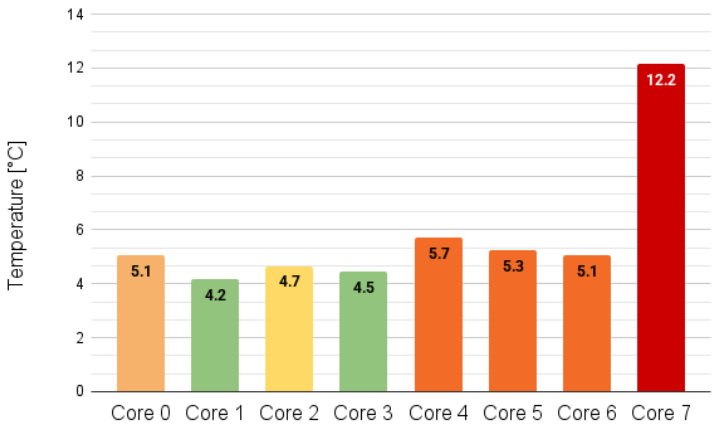


Figure 65: Difference between baseline and run on each core.

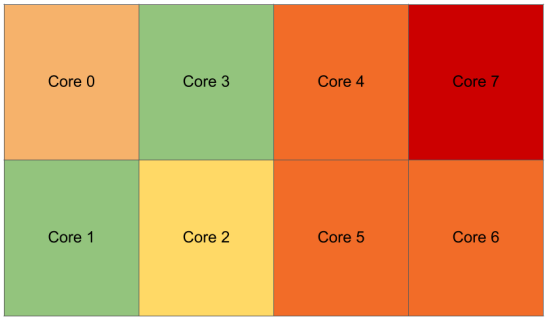


Figure 66: Possible core layout, based on heat propagation experiment.

4.5.5 Covert Channel Attack

To further expand the research in the direction of heat propagation, a small Covert Channel Attack was created. The idea of a covert channel attack is based on bypassing current security measures to transmit sensitive data without breaking them. In more detail, a person (called *A*) has access to secret data in the interior of an entity; however, due to security measures, it cannot transmit the data in the exterior to another person (called *B*). To avoid the security measures, *A* can use a covert channel to transmit the data to the exterior, which will look like a normal operation from inside, and *B* will listen from the exterior and will try to extract the secret values. Furthermore, I decided that the secret data is the text "*hello world*" which will be transmitted by *A* from Core 7 (the interior) to *B* (the exterior) which has access to a nearby core, but not to Core 7. The transmission is performed by sequences of executing the (value of 1) and sleep (value of 0) on Core 7. The secret text was transformed in binary values so that it can be sent using the rule presented above, when the core executes (increasing the CPU load to 90%) for 10 s, it sends a value of 1, and when it is idle for 10 s, it sends a 0. The listening part will be performed by *B*, who is monitoring the temperature of the nearby core, reading the temperature of its core every second. As can be seen in Figure 67, the heat is propagated

to the nearby core. Both cores are presented in this figure so that the propagation can be seen better. However, *B* only has access to the data in green, which represent the propagated data. Further thermal analysis must be applied to these data to extract the secret text. A technique called *binarization* is applied, using a filter that divides data into values of 0 and 1. After this, I have the binary values, and the final values will be extracted by searching for patterns that will help us distinguish the time for transmitting one bit. The results of this experiment are that I was able to reconstruct 71 of 88 bits of the secret text, which means 80.6% of the original text sent from *A* to *B* was recovered.

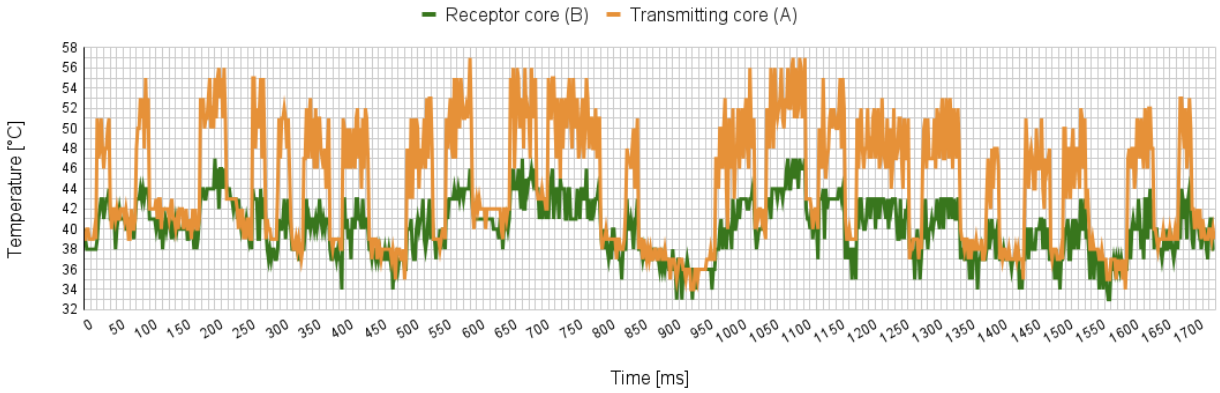


Figure 67: Covert channel attack.

4.5.6 Anomaly detection using Machine Learning

To increase the effectiveness of the Hot-n-Cold technique, I further implement a Machine Learning (ML) algorithm to help to decide whether an anomaly is encountered or not. The chosen algorithm is K-Nearest Neighbors (KNN), which helps to classify whether a thermal trace comes from an authentic command or an augmented one. Therefore, the two classes used in the algorithm are called *original* and *augmented*, and both contain, as features, the vectors of temperatures collected in the experiments in Chapter 3.

The algorithm was used with two sets of data. The first one contains the *ls* temperature vectors, and the other one contains the same data for the *chmod* command, both having 70 thermal traces for the training part and 24 and 25 traces for the testing part. The datasets are composed of the traces of both commands with a moving file as a noise created by concatenating the traces for the original and modified commands. The set for testing was made by repeatedly counting the fourth trace in the original dataset and extracting it for a test, the other traces remaining for the training part.

For both experiments, the number of nearest neighbors *k* had a value of 15, together with the Euclidean distance.

In Figure 68, there is the confusion matrix for *ls* command in a) and in b) for *chmod* command. Based on these matrices, some metrics were calculated, as can be seen in Table 20.

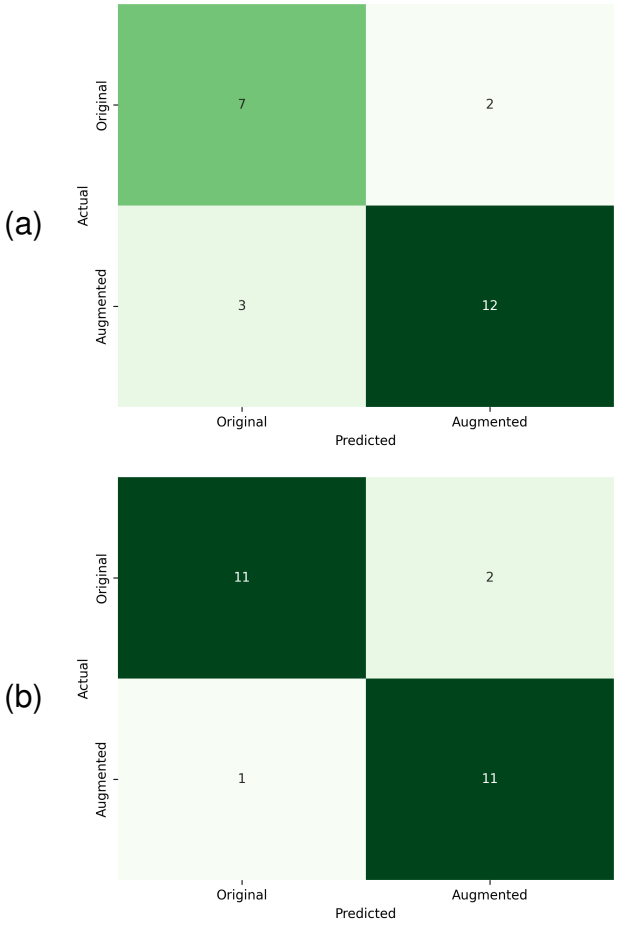


Figure 68: Confusion matrices.

Table 20: ML with KNN metrics.

Metric	***ls***	***chmod***
Accuracy	0.79	0.88
Precision	0.85	0.84
Recall	0.8	0.91
F1-Score	0.81	0.87

The ML algorithm has improved the experimental research by increasing the efficiency and accuracy of the technique. Moreover, it classifies the traces in an automated way, and it helps to validate the hypothesis.

The ML algorithm, together with the correlation thermal analysis, helps to detect an anomaly more effectively.

4.5.7 Detection of commands using Machine Learning

To extend the research, I took into consideration the idea of using Machine Learning to predict different commands based on their thermal behavior. Therefore, scripts similar to the ones for the Beat-the-Heat technique were created for a number of 60 Linux commands. They were called 600 times with a 10 ms pause between them, and for a run as this, a temperature trace was collected. Fifty runs were performed for each command and the temperature traces were collected. Together with these normal-condition runs, each command was run with additional server-based noise, as described in the Chapter 4.2.1 paragraph describing the noise, namely, download images of 1KB, 480KB, 1MB and requesting a web page.

Next, a neural network was created consisting of a hybrid *Conv1D + BiLSTM* architecture, commonly used for time-series classification, signal analysis, or sequence modeling where both local temporal patterns and long-term dependencies matter.

The model used integrates convolutional and recurrent neural network components to capture both local and long-range temporal dependencies in one-dimensional sequential data. Initially, three consecutive one-dimensional convolutional (1D-CNN) layers extract local temporal features from the input signal. Each convolutional layer employs ReLU activation and batch normalization to enhance feature stability and quality of representation. The resulting high-level CNN features are concatenated with a downsampled version of the raw input sequence. This combined feature representation is then processed by a bidirectional long short-term memory (BiLSTM) layer with 128 hidden units per direction, enabling the model to learn contextual dependencies in both temporal directions. Finally, the concatenated BiLSTM outputs are passed through a fully connected layer to produce the final class predictions. The code is presented in Listing 14.

Listing 14: Neural network model

```
class Conv1D_LSTM_Model(nn.Module):
    def __init__(self, num_classes=num_class):
        super(Conv1D_LSTM_Model, self).__init__()
         self.conv1 = nn.Conv1d(in_channels=1, out_channels=32,
            kernel_size=5,     stride=2, padding=2)
         self.conv2 = nn.Conv1d(in_channels=32, out_channels=64,
            kernel_size=5, stride=2, padding=2)
         self.conv3 = nn.Conv1d(in_channels=64, out_channels=128,
            kernel_size=3, stride=2, padding=1)
         self.bn1 = nn.BatchNorm1d(32)
         self.bn2 = nn.BatchNorm1d(64)
         self.bn3 = nn.BatchNorm1d(128)
         self.lstm_input_size = 128 + 1
         self.lstm = nn.LSTM(input_size=self.lstm_input_size,
            hidden_size=128, num_layers=1, batch_first=True,
            bidirectional=True)
         self.fc = nn.Linear(128 * 2, num_classes)


     def forward(self, x):
```

```
        x_raw = x.unsqueeze(1)
        x_cnn = F.relu(self.conv1(x_raw))
        x_cnn = F.relu(self.conv2(x_cnn))
        x_cnn = F.relu(self.bn3(self.conv3(x_cnn)))
        x_cnn = x_cnn.permute(0, 2, 1)
        x_val = F.adaptive_avg_pool1d(x_raw, x_cnn.size(1))
        x_val = x_val.permute(0, 2, 1)
        x_combined = torch.cat([x_cnn, x_val], dim=2)
        lstm_out, _ = self.lstm(x_combined)
        out = self.fc(lstm_out[:, -1, :])
        out = self.fc(tanh_out)
        return out
```

For using this algorithm, the thermal traces were divided in an 80-20 ratio, 80% of the traces were selected for training and 20% for testing. The selection was done by taking the traces from 5 to 5 (e.g. 1, 6, 11, etc.).

The best results for using 15 commands (i.e. *df, dig, ip, less, ls, lspci, mkdir, ping, ps, rm, rsync, scp, time, w, wget*) were an Accuracy of 0.7533 with a Precision of 0.7730, with the Confusion matrix in Figure 69 for system *D-150b*, each command being a class.

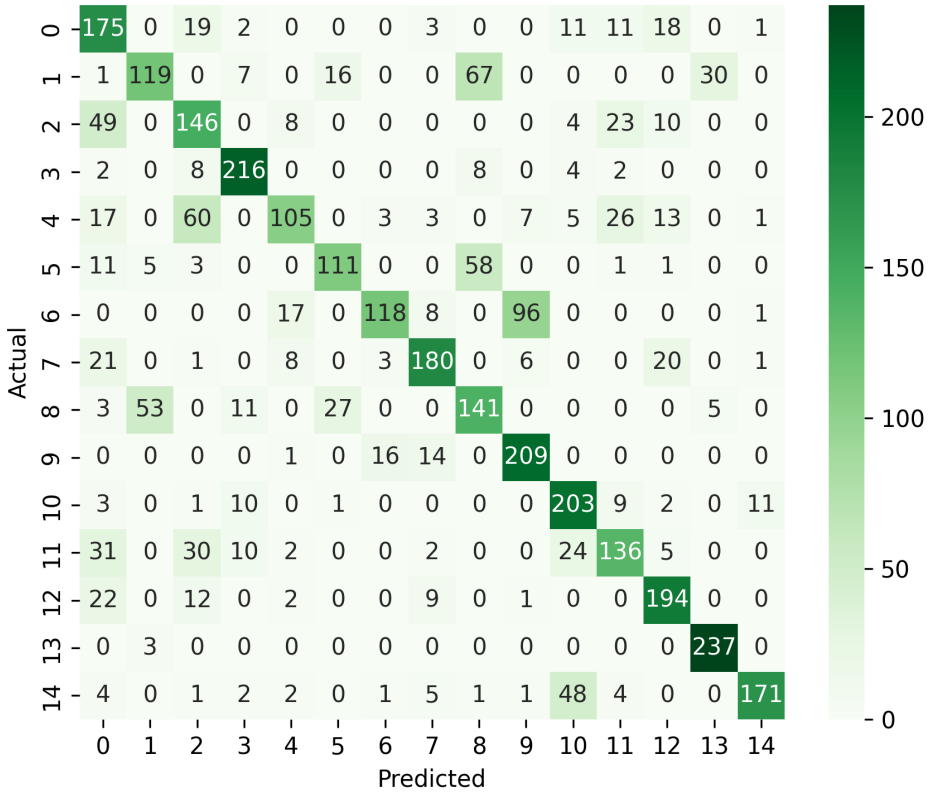


Figure 69: Confusion matrix for 15 classes on system*D-150b*.

Similar was done for 4 commands (i.e. *lspci, mkdir, w, wget*), where the accuracy was higher 0.9250 and precision was 0.9423. Figure 70 shows the Confusion matrix.

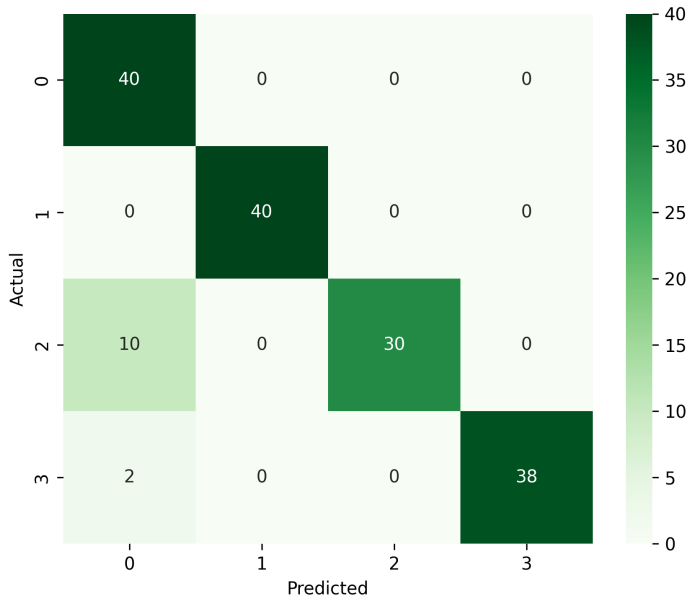


Figure 70: Confusion matrix for 4 classes on system *D-150b*.

The same was done for the other two systems, namely *D-152b* and *D-153b*, however, results were not as good as for the first system, as seen in Figure 71. Accuracy is 0.6500 and precision is 0.4856 for *D-152b* using *dig, ping, printenv, rmdir* commands and 0.6000 and 0.5583 respectively, for *D-153b* using *dir, mkdir, rm, w* commands.

However, while analyzing the results, I realized that this is not a correct experiment for my hypothesis because the network predicts the behavior of a series of commands running one after another. Thereby, I decided to redo the predictions, this time using the login experiments form Chapter 4.3.2, because one command is not rising the temperature visibly as a login does. Here, three classes were used, *0* for change user command (i.e. *su*) described in Listing 13, *1* for login using terminal as in Listing 11, and *2* for login without terminal as in Listing 12. The Confusion matrix for *D-150b* can be seen in Figure 72, where the accuracy is 0.7000 and the precision is 0.7218.

Moreover, to make training and predictions on sampled login traces, the original login traces containing 30 logins each were cut to traces that contain only one login each. the results can be seen in Figure 73, with the accuracy of 0.8533 and a precision of 0.8520. The accuracy was increased in this case because there were multiple traces for training and prediction. These experiments will be further developed as future work.

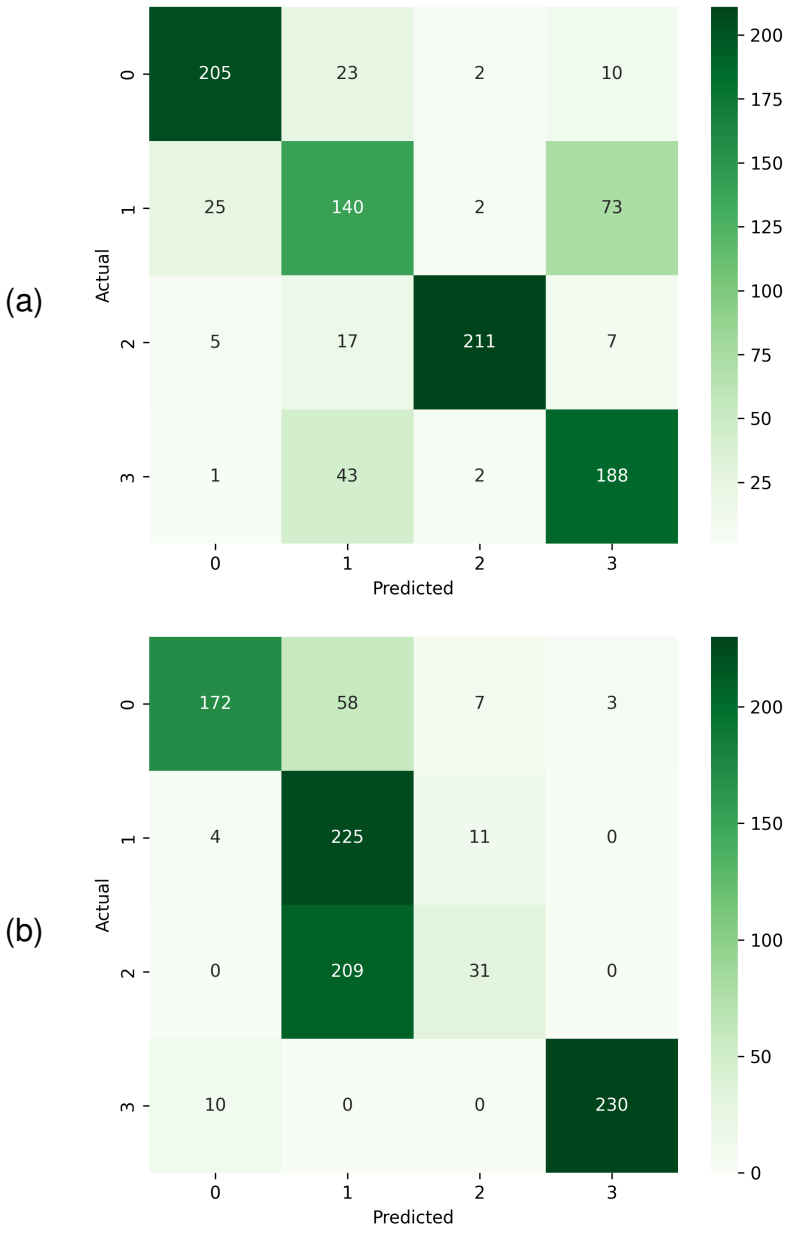


Figure 71: Confusion matrix for 4 classes on system *D-152b* a) and system *D-153b* b).

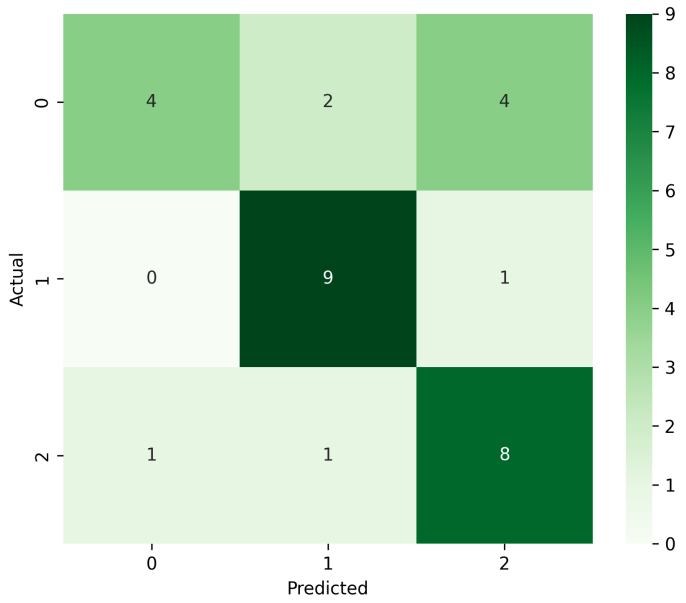


Figure 72: Confusion matrix for login classes on system *D-150b*.

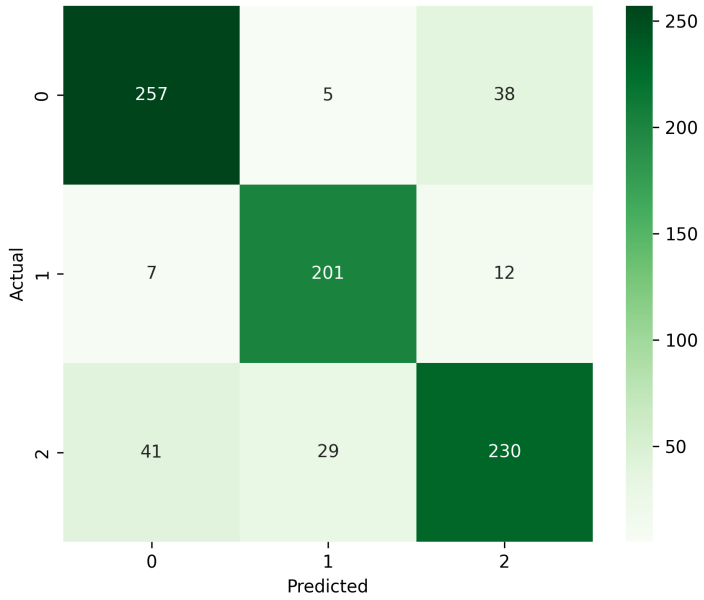


Figure 73: Confusion matrix for login classes sampled on system *D-150b*.

4.6 Original Contributions

As part of this research, the findings have been disseminated through two peer-reviewed publications. Notably, the article ***Beat the Heat: Syscall Attack Detection via Thermal Side Channel*** was published in Future Internet 2024; 16(8):301, MDPI, where the *BeatTheHeat* anomaly detection technique was introduced, applying it in circumstances

much closer to the real-world computational environments; and ***When Things Heat Up: Detecting Malicious Activity Using CPU Thermal Sensors***, which was published in the Journal of Cybersecurity and Privacy 2025; 5(3):56, which demonstrates that anomalies in CPU heat dissipation can be effectively identified, even under high-load or noisy conditions.

This research presents several original and significant contributions to the domain of anomaly detection through thermal sensors of a CPU by **advancing and generalizing the *Hot-n-Cold* technique into a new one called *BeatTheHeat***. Although the original *Hot-n-Cold* approach demonstrated the feasibility of detecting anomalies through thermal traces in controlled, low-noise environments, *BeatTheHeat* extends this concept into more complex and realistic computational settings, thereby improving its practical applicability. The proposed method utilizes CPU temperature variations, captured via Intel built-in digital thermal sensors through the *hwmon* interface, to detect the presence of malicious code injections in standard Linux commands. The underlying hypothesis is that even minor augmentations to a command such as the insertion of crafted system calls, can result in distinguishable changes in thermal behavior, despite producing nearly identical execution results. To assess this, the technique compares the thermal profiles of the original and augmented commands using Correlation Thermal Analysis, an adaptation of the Correlation Power Analysis (CPA) technique, along with Pearson correlation to quantify thermal deviations.

A major contribution of this part lies in the integration of five distinct types of environmental and computational noise, including file operations, extended arithmetic computations, media playback, web browsing, and keyboard presses, designed to simulate real-world user activity and background processes. Despite this added complexity, *BeatTheHeat* reliably identifies injected code through its thermal footprint, confirming the resilience and viability of thermal-based anomaly detection outside controlled laboratory settings. The experiments were performed on three types of CPUs which show high correlations, approaching 0.96, between the original command and an augmented one. This research demonstrates the high potential of detecting attacks through thermal sensors even in a real-world scenario. Furthermore, this work incorporates a supervised machine learning model trained on thermal data, which achieves classification accuracies of up to 88% in distinguishing between clean and modified commands. This integration of statistical analysis and learning-based techniques provides a hybrid detection model that is both accurate and adaptable. Additionally, the research explores the heat propagation across CPU cores by designing and executing a covert-channel attack based on thermal leakage. This investigation successfully reconstructs up to 80% of transmitted secret data and provides empirical insights into the physical layout of CPU cores and their thermal interactions.

Another key contribution is the advanced evaluation of the Hot-n-Cold anomaly detection technique in even more realistic environments with server-noise and varying system loads. While the original method was demonstrated only under controlled and quiet conditions and the second one under user-like noise, *WhenThingsHeatUp* validates its robustness in more complex, real-world scenarios, by using different types of noise as repeatedly loading an HTML page, downloading files of different sizes, and combinations of these two.

This work extends the applicability of thermal side-channel analysis by analyzing thermal footprints associated with specific system-level events. Notably, it identifies the thermal fingerprint of cryptographic operations linked to the CryptoTrooper ransomware, demonstrating the potential for thermal sensor data to support behavioral profiling. Moreover, it can distinguish thermal patterns corresponding to both authorized and unauthorized system login attempts. These results lead to a new way of threat detection beyond traditional software monitoring techniques. Moreover, a Machine Learning technique that uses Convolutional Neural Networks was used to distinguish between commands, with an accuracy of up to 90%.

These findings confirm that thermal monitoring can serve as a viable and complementary layer for system security. The ability to obtain processor temperature data for intrusion and anomaly detection opens new opportunities for research. More research is needed, and such methods can be incorporated into real-time monitoring systems to enhance data protection.

CHAPTER 5

5 Conclusion & Future Work

5.1 Conclusions

Security is a significant problem in both home computers as well as in servers, cloud, and other such systems. Some components in a system are created to increase performance, improve execution timings, and decrease power consumption; however, these components are subject to security breaches. In this thesis, I have presented an extensive survey of the most cited software-based attacks and their countermeasures in the last decade. I have split the attacks into different categories based on the affected components and compared them based on important features.

Moreover, the temperature of different Linux commands on an Intel architecture using thermal sensors was studied. It is demonstrated that these sensors can leak information about running Linux commands to unprivileged users. I proposed the *Hot-n-Cold* technique, which exploits the information leakage from thermal sensors' and detects differences between an original Linux command that is running in different scenarios and a command that was augmented with syscalls that have been reported to be common used in attacks.

Next, the *BeatTheHeat* anomaly detection technique was introduced, applying it in circumstances much closer to the real-world computational environments. It was demonstrated that it detects irregularities in the Linux command behavior through CPU temperature monitoring, even when noise is in place. Moreover, a machine learning algorithm was presented, with an accuracy of up to 88% which, in combination with the thermal correlation analysis, helps us to detect the anomaly in a more efficient and accurate way.

Furthermore, it was demonstrated that anomalies in CPU heat dissipation can be effectively identified through the use of Intel Digital Thermal Sensors, even under high-load or noisy conditions. Three detection methods were explored to support this claim. The first *WhenThingsHeatUp* involved the Hot-n-Cold technique, which was successfully detected if additional lines of code were injected into Linux commands by analyzing thermal patterns. The second approach identified the presence of CryptoTrooper ransomware by capturing the thermal footprint of cryptographic processes. The third method detected system login attempts through their distinct thermal signatures.

This work paves the way for novel anomaly detection and mitigation techniques for side-channel attacks via thermal traces. This research demonstrates the high potential for detecting attacks through thermal sensors even in a real-world scenario. Nevertheless, the techniques can be extended to check for anomalies in CPU temperature during normal operations. These findings confirm that thermal monitoring can serve as a viable and complementary layer for system security. The ability to obtain processor temperature data for intrusion and anomaly detection opens new opportunities for research. More research is needed and such methods should be incorporated into real-time monitoring systems to

enhance data protection.

5.2 Dissemination of research results

The research completed during the PhD has resulted in multiple publications in international scientific conferences and journals, including four first-author papers:

1. T. Vasilas, T. Jakobsche and F. M. Ciorba, "Hot-n-Cold: Mapping the Syscall Attack Surface Using Thermal Side Channels," 2023 22nd International Symposium on Parallel and Distributed Computing (ISPDC), Bucharest, Romania, 2023, pp. 93-100. https://doi.org/10.1109/ISPDC59212.2023.00022

2. T. Vasilas, C. Bacila, R. Brad, "Beat the Heat: Syscall Attack Detection via Thermal Side Channel," in Future Internet. 2024; 16(8):301. https://doi.org/10.3390/fi16080301

3. T. Vasilas and R. Brad, "A Decade in Software-Based Side and Covert Channel Attacks and Countermeasures: A Survey," in IEEE Access, vol. 13, pp. 56587-56606, 2025. https://doi.org/10.1109/ACCESS.2025.3555139

4. T. Vasilas, R. Brad, "When Things Heat Up: Detecting Malicious Activity Using CPU Thermal Sensors," in Journal of Cybersecurity and Privacy. 2025; 5(3):56. https://doi.org/10.3390/jcp5030056

5.3 Future Work

The distinct syscalls parameters can lead to different CPU core temperatures, a matter that it was not considered in *Hot-n-Cold* technique. The future work will focus on that direction, taking into account different types of syscall parameters. This work can be extended to study frequent patterns of consecutive syscalls and create thermal tracing techniques to detect anomalies or attacks in the code execution.

Another important future work stands in increasing the automation of this technique and integrating the Machine Learning algorithm together with the thermal correlation analysis can make it applicable for the continuous execution and detection of anomalies based on temperature and other factors, enabling the automatic identification of execution anomalies as they occur.

Furthermore, using Machine Learning to predict and classify commands based on their thermal behavior is also in the target.

References

[1] "miniHPC," https://hpc.dmi.unibas.ch/en/research/minihpc/.

[2] J. Loughry and D. A. Umphress, "Information leakage from optical emanations," *ACM Transactions on Information and System Security (TISSEC)*, vol. 5, no. 3, pp. 262–289, 2002.

[3] Y. Abdelrahman, M. Khamis, S. Schneegass, and F. Alt, "Stay cool! understanding thermal attacks on mobile-based user authentication," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2017, pp. 3751–3763.

[4] D. Asonov and R. Agrawal, "Keyboard acoustic emanations," in *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004.* IEEE, 2004, pp. 3–11.

[5] M. G. Kuhn, "Compromising emanations: eavesdropping risks of computer displays," University of Cambridge, Computer Laboratory, Tech. Rep., 2003.

[6] M. Guri, M. Monitz, Y. Mirski, and Y. Elovici, "Bitwhisper: Covert signaling channel between air-gapped computers using thermal manipulations," in *2015 IEEE 28th Computer Security Foundations Symposium*. IEEE, 2015, pp. 276–289.

[7] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*. Springer, 1999, pp. 388–397.

[8] J.-J. Quisquater and D. Samyde, "Electromagnetic analysis (ema): Measures and counter-measures for smart cards," in *Smart Card Programming and Security: International Conference on Research in Smart Cards, E-smart 2001 Cannes, France, September 19–21, 2001 Proceedings*. Springer, 2001, pp. 200–210.

[9] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: the case of aes," in *Topics in Cryptology–CT-RSA 2006: The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2005. Proceedings*. Springer, 2006, pp. 1–20.

[10] Y. Yarom and K. Falkner, "FLUSH+ RELOAD: A high resolution, low noise, l3 cache Side-Channel attack," in *23rd USENIX security symposium (USENIX security 14)*, 2014, pp. 719–732.

[11] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ flush: A fast and stealthy cache attack," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13*. Springer, 2016, pp. 279–299.

[12] M. Mushtaq, M. A. Mukhtar, V. Lapotre, M. K. Bhatti, and G. Gogniat, "Winter is here! a decade of cache-based side-channel attacks, detection & mitigation for rsa," *Information Systems*, vol. 92, p. 101524, 2020.

[13] C. Percival, "Cache missing for fun and profit," 2005.

[14] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," 2019.

[15] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," *arXiv preprint arXiv:1801.01207*, 2018.

[16] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient Out-of-Order execution," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 991–1008.

[17] M. Schwarz, M. Schwarzl, M. Lipp, J. Masters, and D. Gruss, "Netspectre: Read arbitrary memory over network," in *Computer Security–ESORICS 2019: 24th European Symposium on Research in Computer Security, Luxembourg, September 23–27, 2019, Proceedings, Part I 24*. Springer, 2019, pp. 279–299.

[18] C. Trippel, D. Lustig, and M. Martonosi, "Meltdownprime and spectreprime: Automatically-synthesized attacks exploiting invalidation-based coherence protocols," *arXiv preprint arXiv:1802.03802*, 2018.

[19] E. M. Koruyeh, K. N. Khasawneh, C. Song, and N. Abu-Ghazaleh, "Spectre returns! speculation attacks using the return stack buffer," in *12th USENIX Workshop on Offensive Technologies (WOOT 18)*, 2018.

[20] C. Canella, J. Van Bulck, M. Schwarz, M. Lipp, B. Von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss, "A systematic evaluation of transient execution attacks and defenses," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 249–266.

[21] D. Gullasch, E. Bangerter, and S. Krenn, "Cache games–bringing access-based cache attacks on aes to practice," in *2011 IEEE Symposium on Security and Privacy*. IEEE, 2011, pp. 490–505.

[22] K. Mowery, S. Keelveedhi, and H. Shacham, "Are aes x86 cache timing attacks still feasible?" in *Proceedings of the 2012 ACM Workshop on Cloud computing security workshop*, 2012, pp. 19–24.

[23] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," *Journal of Cryptographic Engineering*, vol. 8, pp. 1–27, 2018.

[24] J. Szefer, "Survey of microarchitectural side and covert channels, attacks, and defenses," *Journal of Hardware and Systems Security*, vol. 3, no. 3, pp. 219–234, 2019.

[25] M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss, "Platypus: Software-based power side-channel attacks on x86," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 355–371.

[26] Z. Zhang, S. Liang, F. Yao, and X. Gao, "Red alert for power leakage: Exploiting intel rapl-induced side channels," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 2021, pp. 162–175.

[27] T. Kim and Y. Shin, "Thermalbleed: A practical thermal side-channel attack," *IEEE Access*, vol. 10, pp. 25 718–25 731, 2022.

[28] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive Last-Level caches," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 897–912.

[29] C. Disselkoen, D. Kohlbrenner, L. Porter, and D. Tullsen, "Prime+ Abort: A Timer-Free High-Precision l3 cache attack using intel TSX," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 51–67.

[30] G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cross processor cache attacks," in *Proceedings of the 11th ACM on Asia conference on computer and communications security*, 2016, pp. 353–364.

[31] S. Briongos, P. Malagón, J. M. Moya, and T. Eisenbarth, "RELOAD+ REFRESH: Abusing cache replacement policies to perform stealthy cache attacks," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1967–1984.

[32] M. Lipp, V. Hadžić, M. Schwarz, A. Perais, C. Maurice, and D. Gruss, "Take a way: Exploring the security implications of amd's cache way predictors," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, 2020, pp. 813–825.

[33] A. Purnal, F. Turan, and I. Verbauwhede, "Prime+ scope: Overcoming the observer effect for high-precision cache contention attacks," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2906–2920.

[34] N. Benger, J. Van de Pol, N. P. Smart, and Y. Yarom, ""ooh aah... just a little bit": a small amount of side channel can go a long way," in *Cryptographic Hardware and Embedded Systems–CHES 2014: 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings 16*. Springer, 2014, pp. 75–92.

[35] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar, "Wait a minute! a fast, cross-vm attack on aes," in *Research in Attacks, Intrusions and Defenses: 17th International Symposium, RAID 2014, Gothenburg, Sweden, September 17-19, 2014. Proceedings 17*. Springer, 2014, pp. 299–319.

[36] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-tenant side-channel attacks in paas clouds," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 990–1003.

[37] Y. Yarom and N. Benger, "Recovering openssl ecdsa nonces using the flush+ reload cache side-channel attack," *Cryptology ePrint Archive*, 2014.

[38] J. Van de Pol, N. P. Smart, and Y. Yarom, "Just a little bit more," in *Cryptographers' Track at the RSA Conference*. Springer, 2015, pp. 3–21.

[39] G. Irazoqui, T. Eisenbarth, and B. Sunar, "S$a: A shared cache attack that works across cores and defies vm sandboxing–and its application to aes," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 591–604.

[40] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *2015 IEEE symposium on security and privacy*. IEEE, 2015, pp. 605–622.

[41] Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis, "The spy in the sandbox: Practical cache attacks in javascript and their implications," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1406–1418.

[42] B. Gülmezoğlu, M. S. Inci, G. Irazoqui, T. Eisenbarth, and B. Sunar, "A faster and more realistic flush+ reload attack on aes," in *Constructive Side-Channel Analysis and Secure Design: 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers 6*. Springer, 2015, pp. 111–126.

[43] T. Allan, B. B. Brumley, K. Falkner, J. Van de Pol, and Y. Yarom, "Amplifying side channels through performance degradation," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, 2016, pp. 422–435.

[44] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, "ARMageddon: Cache attacks on mobile devices," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 549–564.

[45] C. Pereida García, B. B. Brumley, and Y. Yarom, "Make sure dsa signing exponentiations really are constant-time," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1639–1650.

[46] M. S. Inci, B. Gulmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cache attacks enable bulk key recovery on the cloud," in *Cryptographic Hardware and Embedded Systems–CHES 2016: 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings 18*. Springer, 2016, pp. 368–388.

[47] M. Kayaalp, N. Abu-Ghazaleh, D. Ponomarev, and A. Jaleel, "A high-resolution side-channel attack on last-level cache," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[48] D. J. Bernstein, J. Breitner, D. Genkin, L. Groot Bruinderink, N. Heninger, T. Lange, C. van Vredendaal, and Y. Yarom, "Sliding right into disaster: Left-to-right sliding windows leak," in *Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*. Springer, 2017, pp. 555–576.

[49] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, "Software grand exposure:SGX cache attacks are practical," in *11th USENIX workshop on offensive technologies (WOOT 17)*, 2017.

[50] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard, "Malware guard extension: Using sgx to conceal cache attacks," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings 14*. Springer, 2017, pp. 3–24.

[51] H. Jain, D. A. Balaraju, and C. Rebeiro, "Spy cartel: Parallelizing evict+ time-based cache attacks on last-level caches," *Journal of Hardware and Systems Security*, vol. 3, no. 2, pp. 147–163, 2019.

[52] M. Kurth, B. Gras, D. Andriesse, C. Giuffrida, H. Bos, and K. Razavi, "Netcat: Practical cache attacks from the network," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 20–38.

[53] G. Haas, S. Potluri, and A. Aysu, "itimed: Cache attacks on the apple a10 fusion soc," in *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2021, pp. 80–90.

[54] J. Yu, A. Dutta, T. Jaeger, D. Kohlbrenner, and C. W. Fletcher, "Synchronization storage channels (S2C): Timer-less cache side-channel attacks on the apple m1 via hardware synchronization instructions," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 1973–1990.

[55] D. Cock, Q. Ge, T. Murray, and G. Heiser, "The last mile: An empirical study of timing channels on sel4," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 570–581.

[56] F. Liu, Q. Ge, Y. Yarom, F. Mckeen, C. Rozas, G. Heiser, and R. B. Lee, "Catalyst: Defeating last-level cache side channel attacks in cloud computing," in *2016 IEEE international symposium on high performance computer architecture (HPCA)*. IEEE, 2016, pp. 406–418.

[57] Z. Zhou, M. K. Reiter, and Y. Zhang, "A software approach to defeating side channels in last-level caches," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 871–882.

[58] D. Gruss, J. Lettner, F. Schuster, O. Ohrimenko, I. Haller, and M. Costa, "Strong and efficient cache Side-Channel protection using hardware transactional memory," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 217–233.

[59] V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, and J. Emer, "Dawg: A defense against cache timing attacks in speculative execution processors," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 974–987.

[60] J. Yu, L. Hsiung, M. El Hajj, and C. W. Fletcher, "Data oblivious isa extensions for side channel-resistant and high performance computing," *Cryptology ePrint Archive*, 2018.

[61] G. Dessouky, T. Frassetto, and A.-R. Sadeghi, "HybCache: Hybrid Side-Channel-Resilient caches for trusted execution environments," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 451–468.

[62] F. Liu and R. B. Lee, "Random fill cache architecture," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2014, pp. 203–215.

[63] M. K. Qureshi, "Ceaser: Mitigating conflict-based cache attacks via encrypted-address and remapping," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 775–787.

[64] M. Qureshi, "New attacks and defense for encrypted-address cache," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 360–371.

[65] M. Werner, T. Unterluggauer, L. Giner, M. Schwarz, D. Gruss, and S. Mangard, "ScatterCache: thwarting cache attacks via cache set randomization," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 675–692.

[66] Q. Tan, Z. Zeng, K. Bu, and K. Ren, "Phantomcache: Obfuscating cache conflicts with localized randomization." in *NDSS*, 2020.

[67] A. Purnal, L. Giner, D. Gruss, and I. Verbauwhede, “Systematic analysis of randomization-based protected cache architectures,” in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 987–1002.

[68] W. Song, Z. Xue, J. Han, Z. Li, and P. Liu, “Randomizing set-associative caches against conflict-based cache side-channel attacks,” *IEEE Transactions on Computers*, 2024.

[69] G. Doychev, B. Köpf, L. Mauborgne, and J. Reineke, “Cacheaudit: A tool for the static analysis of cache side channels,” *ACM Transactions on information and system security (TISSEC)*, vol. 18, no. 1, pp. 1–32, 2015.

[70] Y. Yarom, D. Genkin, and N. Heninger, “Cachebleed: a timing attack on openssl constant-time rsa,” *Journal of Cryptographic Engineering*, vol. 7, pp. 99–112, 2017.

[71] V. Varadarajan, T. Ristenpart, and M. Swift, “Scheduler-based defenses against Cross-VM side-channels,” in *23rd USENIX security symposium (USENIX security 14)*, 2014, pp. 687–702.

[72] D. Gruss, M. Lipp, M. Schwarz, R. Fellner, C. Maurice, and S. Mangard, “Kaslr is dead: long live kaslr,” in *Engineering Secure Software and Systems: 9th International Symposium, ESSoS 2017, Bonn, Germany, July 3-5, 2017, Proceedings 9*. Springer, 2017, pp. 161–176.

[73] J. Ge and F. Zhang, “Flushtime: Towards mitigating flush-based cache attacks via collaborating flush instructions and timers on armv8-a,” in *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*, 2023, pp. 190–204.

[74] Z. Wu, Z. Xu, and H. Wang, “Whispers in the hyper-space: high-bandwidth and reliable covert channel attacks inside the cloud,” *IEEE/ACM Transactions on Networking*, vol. 23, no. 2, pp. 603–615, 2014.

[75] L. Liu, A. Wang, W. Zang, M. Yu, M. Xiao, and S. Chen, “Shuffler: Mitigate cross-vm side-channel attacks via hypervisor scheduling,” in *Security and Privacy in Communication Networks: 14th International Conference, SecureComm 2018, Singapore, Singapore, August 8-10, 2018, Proceedings, Part I*. Springer, 2018, pp. 491–511.

[76] T. Zhang, Y. Zhang, and R. B. Lee, “Cloudradar: A real-time side-channel attack detection system in clouds,” in *Research in Attacks, Intrusions, and Defenses: 19th International Symposium, RAID 2016, Paris, France, September 19-21, 2016, Proceedings 19*. Springer, 2016, pp. 118–140.

[77] M. Chiappetta, E. Savas, and C. Yilmaz, “Real time detection of cache-based side-channel attacks using hardware performance counters,” *Applied Soft Computing*, vol. 49, pp. 1162–1174, 2016.

[78] S. Chen, X. Zhang, M. K. Reiter, and Y. Zhang, “Detecting privileged side-channel attacks in shielded execution with déjá vu,” in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017, pp. 7–18.

[79] M.-W. Shih, S. Lee, T. Kim, and M. Peinado, “T-sgx: Eradicating controlled-channel attacks against enclave programs.” in *NDSS*, 2017.

[80] C. Li and J.-L. Gaudiot, “Detecting spectre attacks using hardware performance counters,” *IEEE Transactions on Computers*, vol. 71, no. 6, pp. 1320–1331, 2021.

[81] M. Mushtaq, M. M. Yousaf, M. K. Bhatti, V. Lapotre, and G. Gogniat, “The kingsguard os-level mitigation against cache side-channel attacks using runtime detection,” *Annals of Telecommunications*, vol. 77, no. 11, pp. 731–747, 2022.

[82] M. Wu, S. McCamant, P.-C. Yew, and A. Zhai, “Predator: A cache side-channel attack detector based on precise event monitoring,” in *2022 IEEE International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE, 2022, pp. 25–36.

[83] H. Yan and C. Cui, “Cachehawkeye: Detecting cache side channel attacks based on memory events,” *Future Internet*, vol. 14, no. 1, p. 24, 2022.

[84] L. Wang, L. Bu, and F. Song, “Scaguard: Detection and classification of cache side-channel attacks via attack behavior modeling and similarity comparison,” in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[85] H. Kim, C. Hahn, H. J. Kim, Y. Shin, and J. Hur, "Deep learning based detection for multiple cache side-channel attacks," *IEEE Transactions on Information Forensics and Security*, 2023.

[86] M. Ghaniyoun, K. Barber, Y. Xiao, Y. Zhang, and R. Teodorescu, "Teesec: Pre-silicon vulnerability discovery for trusted execution environments," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–15.

[87] Y. Yang, T. Bourgeat, S. Lau, and M. Yan, "Pensieve: Microarchitectural modeling for security evaluation," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–15.

[88] M. Kapotoglu Koc and D. T. Altilar, "Selection of best fit hardware performance counters to detect cache side-channel attacks," in *Proceedings of the 2023 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems*, 2023, pp. 17–22.

[89] G. Barthe, M. Böhme, S. Cauligi, C. Chuengsatiansup, D. Genkin, M. Guarnieri, D. M. Romero, P. Schwabe, D. Wu, and Y. Yarom, "Testing side-channel security of cryptographic implementations against future microarchitectures," *arXiv preprint arXiv:2402.00641*, 2024.

[90] W.-M. Hu, "Reducing timing charmers with fuzzy time," in *Proceedings. 1991 IEEE Computer Society Symposium on Research in Security and Privacy*. IEEE Computer Society, 1991, pp. 8–8.

[91] R. Hund, C. Willems, and T. Holz, "Practical timing side channel attacks against kernel space aslr," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 191–205.

[92] J. Van Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx, "Telling your secrets without page faults: Stealthy page Table-Based attacks on enclaved execution," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 1041–1056.

[93] B. Gras, K. Razavi, E. Bosman, H. Bos, and C. Giuffrida, "Aslr on the line: Practical cache attacks on the mmu." in *NDSS*, vol. 17, 2017, p. 26.

[94] J. Koschel, C. Giuffrida, H. Bos, and K. Razavi, "Tagbleed: Breaking kaslr on the isolated kernel address space using tagged tlbs," in *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2020, pp. 309–321.

[95] A. Tatar, D. Trujillo, C. Giuffrida, and H. Bos, "TLB; DR: Enhancing TLB-based attacks with TLB desynchronized reverse engineering," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 989–1007.

[96] B. Gras, K. Razavi, H. Bos, and C. Giuffrida, "Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 955–972.

[97] Z. Zhang, Y. Cheng, D. Liu, S. Nepal, Z. Wang, and Y. Yarom, "Pthammer: Cross-user-kernel-boundary rowhammer through implicit accesses," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 28–41.

[98] Y. Jang, S. Lee, and T. Kim, "Breaking kernel address space layout randomization with intel tsx," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 380–392.

[99] D. Gruss, C. Maurice, A. Fogh, M. Lipp, and S. Mangard, "Prefetch side-channel attacks: Bypassing smap and kernel aslr," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 368–379.

[100] C. Bienia, *Benchmarking modern multiprocessors*. Princeton University, 2011.

[101] "The postgresql global development group: pgbench." https://www.postgresql.org/docs/9.6/static/pgbench.html(2016).

[102] P. Group *et al.*, "A memo on exploration of splash-2 input sets," *Princeton University*, 2011.

[103] S. Deng, W. Xiong, and J. Szefer, "Secure tlbs," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 346–359.

[104] F. Stolz, J. P. Thoma, P. Sasdrich, and T. Güneysu, "Risky translations: Securing tlbs against timing side channels," *Cryptology ePrint Archive*, 2022.

[105] "gem5," https://www.gem5.org/documentation/gem5art/tutorials/microbench-tutorial.

[106] D. Guttman, M. Arunachalam, V. Calina, and M. T. Kandemir, "Prefetch tuning optimizations," in *High Performance Parallelism Pearls: Multicore and Many-core Programming Approaches*. Elsevier Inc., 2015, pp. 401–419.

[107] J.-L. Baer and T.-F. Chen, "An effective on-chip preloading scheme to reduce data access penalty," in *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, 1991, pp. 176–186.

[108] Y. Chen, L. Pei, and T. E. Carlson, "Afterimage: Leaking control flow data and tracking load operations via the hardware prefetcher," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, pp. 16–32.

[109] M. Bakhshalipour, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Domino temporal data prefetcher," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 131–142.

[110] Y. Guo, X. Xin, Y. Zhang, and J. Yang, "Leaky way: a conflict-based cache covert channel bypassing set associativity," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 646–661.

[111] Y. Guo, A. Zigerelli, Y. Zhang, and J. Yang, "Adversarial prefetch: New cross-core cache side channel attacks," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1458–1473.

[112] Y. Chen, A. Hajiabadi, L. Pei, and T. E. Carlson, "Prefetchx: Cross-core cache-agnostic prefetcher-based side-channel attacks," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2024.

[113] D. Wang, Z. Qian, N. Abu-Ghazaleh, and S. V. Krishnamurthy, "Papp: Prefetcher-aware prime and probe side-channel attack," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[114] C. Xiao, M. Tang, and S. Guilley, "Exploiting the microarchitectural leakage of prefetching activities for side-channel attacks," *Journal of Systems Architecture*, vol. 139, p. 102877, 2023.

[115] T. Zhang, S. Chen, F. Liu, and R. B. Lee, "Side channel vulnerability metrics: the promise and the pitfalls," 2013.

[116] M. Lipp, D. Gruss, and M. Schwarz, "AMD prefetch attacks through power and time," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 643–660.

[117] C. Canella, M. Schwarz, M. Haubenwallner, M. Schwarzl, and D. Gruss, "Kaslr: Break it, fix it, repeat," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, 2020, pp. 481–493.

[118] "Apache http server benchmarking tool," https://httpd.apache.org/docs/2.4/programs/ab.html.

[119] M. A. Mukhtar, M. Mushtaq, M. K. Bhatti, V. Lapotre, and G. Gogniat, "Flush+ prefetch: A countermeasure against access-driven cache-based side-channel attacks," *Journal of Systems Architecture*, vol. 104, p. 101698, 2020.

[120] B. David, "misc-cache-attacks," https://github.com/polymorf/misc-cache-attacks/.

[121] L. Li, J. Huang, L. Feng, and Z. Wang, "Prefender: A prefetching defender against cache side channel attacks as a pretender," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 1509–1514.

[122] Standard Performance Evaluation Corporation (SPEC), "SPEC CPU2006 Benchmark Suite," 2006. [Online]. Available: https://www.spec.org/cpu2006/

[123] G. Maisuradze and C. Rossow, "ret2spec: Speculative execution using return stack buffers," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 2109–2122.

[124] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 142–157.

[125] C. Canella, D. Genkin, L. Giner, D. Gruss, M. Lipp, M. Minkin, D. Moghimi, F. Piessens, M. Schwarz, B. Sunar *et al.*, "Fallout: Leaking data on meltdown-resistant cpus," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 769–784.

[126] A. Bhattacharyya, A. Sandulescu, M. Neugschwandtner, A. Sorniotti, B. Falsafi, M. Payer, and A. Kurmus, "Smotherspectre: exploiting speculative execution through port contention," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 785–800.

[127] J. Wikner and K. Razavi, "RETBLEED: Arbitrary speculative code execution with return instructions," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 3825–3842.

[128] D. Moghimi, "Downfall: Exploiting speculative data gathering," in *32th USENIX Security Symposium (USENIX Security 2023)*, vol. 1, no. 5, 2023.

[129] "Intel® 64 and ia-32 architectures software developer's manual, combined volumes," https://cdrdv2.intel.com/v1/dl/getContent/671200.

[130] M. Yan, J. Choi, D. Skarlatos, A. Morrison, C. Fletcher, and J. Torrellas, "Invisispec: Making speculative execution invisible in the cache hierarchy," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 428–441.

[131] K. Barber, A. Bacha, L. Zhou, Y. Zhang, and R. Teodorescu, "Specshield: Shielding speculative data from microarchitectural covert channels," in *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2019, pp. 151–164.

[132] K. N. Khasawneh, E. M. Koruyeh, C. Song, D. Evtyushkin, D. Ponomarev, and N. Abu-Ghazaleh, "Safespec: Banishing the spectre of a meltdown with leakage-free speculation," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.

[133] J. Bucek, K.-D. Lange, and J. v. Kistowski, "SPEC CPU2017: Next-generation compute benchmark," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, 2018, pp. 41–42.

[134] P. Li, L. Zhao, R. Hou, L. Zhang, and D. Meng, "Conditional speculation: An effective approach to safeguard out-of-order execution against spectre attacks," in *2019 IEEE international symposium on high performance computer architecture (HPCA)*. IEEE, 2019, pp. 264–276.

[135] E. M. Koruyeh, S. H. A. Shirazi, K. N. Khasawneh, C. Song, and N. Abu-Ghazaleh, "Speccfi: Mitigating spectre attacks using cfi informed speculation," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 39–53.

[136] "Running average power limit energy reporting," https://docs.kernel.org/hwmon/coretemp.html, accessed: 2024-05-17.

[137] A. Kogler, J. Juffinger, L. Giner, L. Gerlach, M. Schwarzl, M. Schwarz, D. Gruss, and S. Mangard, "Collide+ Power: Leaking inaccessible data with software-based power side channels," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 7285–7302.

[138] M. FUSI, "Information-leakage analysis based on hardware performance counters," 2016.

[139] L. Caviglione, M. Gaggero, J.-F. Lalande, W. Mazurczyk, and M. Urbański, "Seeing the unseen: revealing mobile malware hidden communications via energy consumption and artificial intelligence," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 4, pp. 799–810, 2015.

[140] D. R. Dipta and B. Gulmezoglu, "Mad-en: Microarchitectural attack detection through system-wide energy consumption," *IEEE Transactions on Information Forensics and Security*, 2023.

[141] H. Mantel, J. Schickel, A. Weber, and F. Weber, "How secure is green it? the case of software-based energy side channels," in *Computer Security: 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I 23*. Springer, 2018, pp. 218–239.

[142] Y. Wang, R. Paccagnella, E. T. He, H. Shacham, C. W. Fletcher, and D. Kohlbrenner, "Hertzbleed: Turning power Side-Channel attacks into remote timing attacks on x86," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 679–697.

[143] C. Liu, A. Chakraborty, N. Chawla, and N. Roggel, "Frequency throttling side-channel attack," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 1977–1991.

[144] M. Hutter and J.-M. Schmidt, "The temperature side channel and heating fault attacks," in *Smart Card Research and Advanced Applications: 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers 12*. Springer, 2014, pp. 219–235.

[145] A. Aljuffri, M. Zwalua, C. R. W. Reinbrecht, S. Hamdioui, and M. Taouil, "Applying thermal side-channel attacks on asymmetric cryptography," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 11, pp. 1930–1942, 2021.

[146] R. J. Masti, D. Rai, A. Ranganathan, C. Müller, L. Thiele, and S. Capkun, "Thermal covert channels on multi-core platforms," in *24th USENIX security symposium (USENIX security 15)*, 2015, pp. 865–880.

[147] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4 (Cat. No. 01EX538)*. IEEE, 2001, pp. 3–14.

[148] D. B. Bartolini, P. Miedl, and L. Thiele, "On the capacity of thermal covert channels in multicores," in *Proceedings of the Eleventh European Conference on Computer Systems*, 2016, pp. 1–16.

[149] Z. Long, X. Wang, Y. Jiang, G. Cui, L. Zhang, and T. Mak, "Improving the efficiency of thermal covert channels in multi-/many-core systems," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1459–1464.

[150] S. Dey, A. K. Singh, and K. McDonald-Maier, "Thermalattacknet: Are cnns making it easy to perform temperature side-channel attack in mobile edge devices?" *Future Internet*, vol. 13, no. 6, p. 146, 2021.

[151] N. Mishra, T. L. Dutta, S. Shukla, A. Chakraborty, and D. Mukhopadhyay, "Too hot to handle: Novel thermal side-channel in power attack-protected intel processors," in *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2024, pp. 378–382.

[152] X. Zhang, Z. Zhang, Q. Shen, W. Wang, Y. Gao, Z. Yang, and Z. Wu, "Thermalscope: A practical interrupt side channel attack based on thermal event interrupts," in *2024 Design Automation Conference*, 2024.

[153] T. Claeys, F. Rousseau, B. Simunovic, and B. Tourancheau, "Thermal covert channel in bluetooth low energy networks," in *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, 2019, pp. 267–276.

[154] K. Dhananjay, V. F. Pavlidis, A. K. Coskun, and E. Salman, "High bandwidth thermal covert channel in 3-d-integrated multicore processors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 11, pp. 1654–1667, 2022.

[155] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: Characterization and methodological considerations," *ACM SIGARCH computer architecture news*, vol. 23, no. 2, pp. 24–36, 1995.

[156] P. Gu, D. Stow, R. Barnes, E. Kursun, and Y. Xie, "Thermal-aware 3d design for side-channel information leakage," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*. IEEE, 2016, pp. 520–527.

[157] H. Huang, X. Wang, Y. Jiang, A. K. Singh, M. Yang, and L. Huang, "On countermeasures against the thermal covert channel attacks targeting many-core systems," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[158] Q. Wu, X. Wang, and J. Chen, "Defending against thermal covert channel attacks by task migration in many-core system," in *2021 IEEE 3rd International Conference on Circuits and Systems (ICCS)*. IEEE, 2021, pp. 111–120.

[159] H. Huang, X. Wang, Y. Jiang, A. K. Singh, M. Yang, and L. Huang, "Detection of and countermeasure against thermal covert channel in many-core systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 2, pp. 252–265, 2021.

[160] P. Rahimi, A. K. Singh, and X. Wang, "Selective noise based power-efficient and effective countermeasure against thermal covert channel attacks in multi-core systems," *Journal of Low Power Electronics and Applications*, vol. 12, no. 2, p. 25, 2022.

[161] X. Wang, S. Wang, Y. Jiang, A. K. Singh, M. Yang, and L. Huang, "Combating stealthy thermal covert channel attack with its thermal signal transmitted in direct sequence spread spectrum," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4064–4075, 2022.

[162] X. Wang, H. Huang, R. Chen, Y. Jiang, A. K. Singh, M. Yang, and L. Huang, "Detection of thermal covert channel attacks based on classification of components of the thermal signal features," *IEEE Transactions on Computers*, vol. 72, no. 4, pp. 971–983, 2022.

[163] J. González-Gómez, M. B. Sikal, H. Khdr, L. Bauer, and J. Henkel, "Smart detection of obfuscated thermal covert channel attacks in many-core processors," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[164] T. Vasilas, T. Jakobsche, and F. M. Ciorba, "Hot-n-cold: Mapping the syscall attack surface using thermal side channels," in *2023 22nd International Symposium on Parallel and Distributed Computing (ISPDC)*. IEEE, 2023, pp. 93–100.

[165] T. Vasilas, C. Bacila, and R. Brad, "Beat the heat: Syscall attack detection via thermal side channel," *Future Internet*, vol. 16, no. 8, p. 301, 2024.

[166] S. Forrest, S. Hofmeyr, and A. Somayaji, "The evolution of system-call monitoring," in *2008 Annual Computer Security Applications Conference (ACSAC)*. IEEE, 2008, pp. 418–430.

[167] M. Bagherzadeh, N. Kahani, C.-P. Bezemer, A. E. Hassan, J. Dingel, and J. R. Cordy, "Analyzing a decade of Linux system calls," *Empirical Software Engineering*, vol. 23, pp. 1519–1551, 2018.

[168] D. Zhan, Z. Yu, X. Yu, H. Zhang, and L. Ye, "Shrinking the Kernel Attack Surface Through Static and Dynamic Syscall Limitation," *IEEE Transactions on Services Computing*, 2022.

[169] Y. Xing, J. Cao, K. Sun, F. Yan, and S. Wan, "The devil is in the detail: Generating system call whitelist for Linux seccomp," *Future Generation Computer Systems*, vol. 135, pp. 105–113, 2022.

[170] H.-W. Hung, Y. Liu, and A. A. Sani, "Sifter: protecting security-critical kernel modules in Android through attack surface reduction," in *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, 2022, pp. 623–635.

[171] R. R. Karn, P. Kudva, H. Huang, S. Suneja, and I. M. Elfadel, "Cryptomining detection in container clouds using system calls and explainable machine learning," *IEEE transactions on parallel and distributed systems*, vol. 32, no. 3, pp. 674–691, 2020.

[172] A. J. Gaidis, V. Atlidakis, and V. P. Kemerlis, "Sysxchg: Refining privilege with adaptive system call filters," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 1964–1978.

[173] S. Song, S. Suneja, M. V. Le, and B. Tak, "On the value of sequence-based system call filtering for container security," in *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*. IEEE, 2023, pp. 296–307.

[174] S. Yang, B. B. Kang, and J. Nam, "Optimus: association-based dynamic system call filtering for container attack surface reduction," *Journal of Cloud Computing*, vol. 13, no. 1, p. 71, 2024.

[175] T. Vasilas and R. Brad, "A decade in software-based side and covert channel attacks and countermeasures: A survey," *IEEE Access*, 2025.

[176] "What is high-performance computing," https://www.oracle.com/cloud/hpc/what-is-hpc/#industries-hpc.

[177] A. Shah, "Top HPC Players: It's Time to Get Serious About Security," https://www.hpcwire.com/2023/03/09/top-hpc-players-its-time-to-get-serious-about-security/.

[178] Y. Guo, R. Chandramouli, L. Wofford, R. Gregg, G. Key, A. Clark, C. Hinton, A. Prout, A. Reuther, R. Adamson *et al.*, "High-performance computing (hpc) security: Architecture, threat analysis, and security posture," National Institute of Standards and Technology, Tech. Rep., 2023.

[179] "Etp4hpc's strategic research agenda 2015 update, european technology multi-annual roadmap towards exascale," https://www.etp4hpc.eu/image/fotos/2016/01/ETP4HPC-SRA-2-Single-Page.pdf, 2013.

[180] R. Marek, "Kernel driver coretemp," https://docs.kernel.org/hwmon/coretemp.html.

[181] "Common Vulnerabilities and Exposures," https://cve.mitre.org/cve/search_cve_list.html.

[182] "Intel® 64 and ia-32 architectures software developer's manual combined volumes: 1, 2a, 2b, 2c, 2d, 3a, 3b, 3c, 3d, and 4 - order number: 325462-080us june 2023," https://www.intel.com/content/www/us/en/content-details/782158/intel-64-and-ia-32-architectures-software-developer-s-manual-combined-volumes-1-2a-2b-2c-2d-3a-3b-3c-3d-and-4.html?wapkw=intel%2064%20and%20ia-32%20architectures%20software%20developer%27s%20manual&docid=782158, online; accessed 04 June 2024.

[183] "Intel xeon processor e5 v4 product family - thermal mechanical specifications and design guide," https://en.wikichip.org/w/images/d/d2/Intel_Xeon_Processor_E5_v4_Product_Family_Thermal_Mechanical_Specification_and_Design_Guide.pdf, 2016.

[184] "Intel cpu temperature guide," https://forums.tomshardware.com/threads/intel-cpu-temperature-guide.1488337/, online; accessed 21 June 2024.

[185] "Techpowerup - intel core i7-10700 review," https://www.techpowerup.com/review/intel-core-i7-10700/3.html, online; accessed 16 June 2024.

[186] "Intel® xeon® processor e5-2640 v4 25m cache, 2.40 ghz," https://www.intel.com/content/www/us/en/products/sku/92984/intel-xeon-processor-e52640-v4-25m-cache-2-40-ghz/specifications.html.

[187] R. Love, *Linux system programming: talking directly to the kernel and C library*. O'Reilly Media, Inc., 2013.

[188] J. L. Peterson and A. Silberschatz, *Operating system concepts*. Addison-Wesley Longman Publishing Co., Inc., 1985.

[189] "Linux manual page: syscalls," https://man7.org/linux/man-pages/man2/syscalls.2.html.

[190] "GitHub Coreutils," https://github.com/coreutils/coreutils.

[191] "Cpuburn-in," http://cpuburnin.com/.

[192] "Mersenne prime search," https://www.mersenne.org/.

[193] "Common vulnerabilities and exposures," https://cve.mitre.org/cve/search_cve_list.html, online; accessed 29 May 2024.

[194] “Clonezilla - the free and open source software for disk imaging and cloning,” https://clonezilla.org/, online; accessed 19 April 2024.

[195] “Coreutils - github,” https://github.com/coreutils/coreutils, online; accessed February 2023.

[196] “ffplay documentation,” https://ffmpeg.org/ffplay.html, online; accessed 03 May 2024.

[197] “taskset(1) — linux manual page,” https://man7.org/linux/man-pages/man1/taskset.1.html, online; accessed 02 February 2024.

[198] A. U. Keith G. Calkins, “Applied statistics - lesson 5. correlation coefficients,” https://www.andrews.edu/~calkins/math/edrm611/edrm05.htm, online; accessed 18 July 2024.

[199] Intel, “Intel® core™ i7-10700f processor,” https://www.intel.com/content/www/us/en/products/sku/199318/intel-core-i710700f-processor-16m-cache-up-to-4-80-ghz/specifications.html (Accessed: 2025-06-07).

[200] C. Intel, “Intel® core™ i7-13700 processor,” https://www.intel.com/content/www/us/en/products/sku/230490/intel-core-i713700-processor-30m-cache-up-to-5-20-ghz/specifications.html (Accessed: 2025-06-07).

[201] Intel, “Intel® core™ i9-11900k processor,” https://www.intel.com/content/www/us/en/products/sku/212325/intel-core-i911900k-processor-16m-cache-up-to-5-30-ghz/specifications.html (Accessed: 2025-06-07).

[202] C. I. King, “stress-ng,” https://github.com/ColinIanKing/stress-ng (Accessed: 2025-05-30).

[203] P. Alcorn, “Intel core i9-11900k and core i5-11600k review: Rocket lake blasts off.” https://www.tomshardware.com/reviews/intel-core-i9-11900k-and-i5-11600k-review (Accessed: 2025-06-07).

[204] A. S. Foundation, “Apache http server project,” https://httpd.apache.org/ (Accessed: 2025-06-01).

[205] A. Szent-Gyorgyi, “Cryptotrooper: the first linux white-box ransomware for learning purpose,” https://github.com/jdsecurity/CryptoTrooper/tree/master (Accessed: 2025-05-01).

[206] O. Project, “Openssl: The open source toolkit for ssl/tls,” https://www.openssl.org/ (Accessed: 2025-06-01).

[207] M. Kerrisk, “ssh — linux manual page,” https://man7.org/linux/man-pages/man1/ssh.1.html (Accessed: 2025-05-30).

[208] C. G. Ltd, “Openssh server,” https://documentation.ubuntu.com/server/how-to/security/openssh-server/index.html (Accessed: 2025-06-01).

[209] M. Haahr, “Random.org,” https://www.random.org/ (Accessed: 2025-06-01).

[210] “stress-ng - a tool to load and stress a computer system,” https://manpages.ubuntu.com/manpages/jammy/man1/stress-ng.1.html, online; accessed 30 November 2023.