future internet

MDPI

Article

Beat the Heat: Syscall Attack Detection via Thermal Side Channel

Teodora Vasilas *, Claudiu Bacila and Remus Brad *

Department of Computer Science, Electrical and Electronics Engineering, University of Sibiu, 4 Emil Cioran Street, 550025 Sibiu, Romania; claudiu.bacila@ulbsibiu.ro

* Correspondence: teodora.vasilas@ulbsibiu.ro (T.V.); remus.brad@ulbsibiu.ro (R.B.)

Abstract: As the complexity and integration of electronic devices increase, understanding and mitigating side-channel vulnerabilities will remain a critical area of cybersecurity research. The new and intriguing software-based thermal side-channel attacks and countermeasures use thermal emissions from a device to extract or defend sensitive information, by reading information from the built-in thermal sensors via software. This work extends the Hot-n-Cold anomaly detection technique, applying it in circumstances much closer to the real-world computational environments by detecting irregularities in the Linux command behavior through CPU temperature monitoring. The novelty of this approach lies in the introduction of five types of noise across the CPU, including moving files, performing extended math computations, playing songs, and browsing the web while the attack detector is running. We employed Hot-n-Cold to monitor core temperatures on three types of CPUs utilizing two commonly used Linux terminal commands, *ls* and *chmod*. The results show a high correlation, approaching 0.96, between the original Linux command and a crafted command, augmented with vulnerable system calls. Additionally, a Machine Learning algorithm was used to classify whether a thermal trace is augmented or not, with an accuracy of up to 88%. This research demonstrates the potential for detecting attacks through thermal sensors even when there are different types of noise in the CPU, simulating a real-world scenario.

Keywords: Linux OS; thermal sensors; syscalls; side channels; security

check for updates

Citation: Vasilas, T.; Bacila, C.; Brad, R. Beat the Heat: Syscall Attack Detection via Thermal Side Channel. *Future Internet* **2024**, *16*, 301. https://doi.org/10.3390/fi16080301

Academic Editor: Xuebin Ren

Received: 29 June 2024
Revised: 8 August 2024
Accepted: 19 August 2024
Published: 21 August 2024
Corrected: 21 October 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Monitoring the temperature of a CPU is a very important detail in order to provide a reliable computing environment, execute at an optimal performance and preserve the health of the hardware. Temperature monitoring prevents overheating and thus a reduced lifespan for the hardware, maintains the performance by optimizing the cooling process, ensures the stability of the system to avoid crashes and unexpected shutdowns, improves the energy efficiency by dynamically adjust the power consumption, and improves the user experience by reducing fan noise. However, the spotlight of the benefits introduced by temperature monitoring is diminished by an interface that can be accessed by anyone, providing an attack opening as well as a detection tool.

A user process can use any of the existing system calls in an operation system while executing varied 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 one can detect the differences in temperature for the identical set of instructions but different operands [1]. 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.

Hot-and-Cold [2] is a technique that detects anomalies using the *ls* and *chmod* Linux commands, which shows a correlation up to 0.8 between an original and an augmented command, the second having a higher temperature due to the additional code. However, this technique was employed in a controlled environment, with zero noise, which in fact could not lead to a strong conclusion about the application of the technique in real-life scenarios.

Therefore, this research 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 pattern of the original and the modified command will be very similar. The temperature monitoring is performed via Intel digital sensors, which are integrated within Intel processors to monitor the temperature of the CPU. These sensors provide accurate and real-time temperature readings through the *hwmon* interface [3], which allows both users and system administrators to monitor and manage hardware health. The augmentation is performed based on the information from the Common Vulnerability and Exposures (CVE) database [4]. To process and compare the two temperature vectors of the original and augmented with code commands, we 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.

The novelty of *BeatTheHeat* stands in the used approach, which reads the CPU temperature in order to *detect a possible attack*. To the best of our knowledge, until now, no research was conducted in this direction. Some studies using CPU temperature were presented in [1,5]; however, they focus their study to *create attacks* using the heat dissipation of a CPU.

Furthermore, in addition to the Correlation Thermal Analysis, we introduce a Machine Learning (ML) algorithm which helps us classify whether a command was augmented or not based on its thermal traces. This algorithm has an classification accuracy of up to 88%.

Moreover, to study the heat propagation through cores, we created a Covert-channel Attack, and we were able to reconstruct 80% of the secret data. Additionally, we proposed the layout of the CPU on the die by exploring how heat propagates to the nearby cores.

Certainly, the above technique can be further developed and applied while a program is running to automatically detect any anomalies throughout execution. This enhancement would involve converting the correlation, originally determined at the end of the experiments, into an automatic computation that runs concurrently with temperature readings and is recomputed in a recurrent way.

1.1. Organization

The paper is organized as follows: the first section contains the Introduction in Section 1 and some Related Work in Section 1.2, Section 2 contains some Background Information and the Methodology of this study, and Section 3 includes the Results, followed by Discussion in Section 4, Conclusion in Section 5 and Appendix A.

1.2. Related Work

Lately, two types of cybersecurity attacks have emerged, namely covert-channel and side-channel attacks. The first category, initially presented as an isolation issue [6], uses processing to transmit (from a sender to a receiver), even though it was not created to communicate in this way [7]. It can use all kinds of channels to transmit, channels that

were not created in this scope, as thermal emanations, power, timing, and others [8]. In the second category, the secret information is not transmitted from one to another. The leakage is created based on the normal usage of a system, as the unintended consequence of a system's way of working [9]. The difference between the two categories is that, in covert-channel attacks, both the sender and the receiver are managed by the attacker at some point. On the other hand, the side-channel attacks are more difficult to create because the victim is not managed by the attacker at any point, and it is not aware of the leakage [8].

1.2.1. Thermal-Based Attacks and Countermeasures

The related work contains attacks which are constructed based on the different types of measuring the thermal dissipation of the CPU. The attacks are measured from hardware or software, and they use side channels or covert channels. RSA private key was recovered in [10], using a fault side-channel attack which risen the temperature at a very high rate on AVR and PIC microcontrollers. Covert-channels were investigated in the context of IoT in [11]; however, they did not conduct any further experiment regarding the thermal sensors. Correlation Power Analysis (CPA) and Deep Learning (DL) were adapted for heat dissipation in [12], where the private key for the RSA algorithm was fully recovered; however, this kind of study is a hardware-based one, and physical access to the targeted device is needed, a procedure which is out of our scope. In [5], two versions of covert channel attacks were implemented on a multi-core computing system, reaching a peak data rate of 12.5 bps. The attackers utilized the *coretemp* filesystem in Linux to monitor the temperature of one core, which is affected by the temperature of an adjacent one. By doing so, they achieved a high correspondence between the temperatures of the cores while executing the RSA algorithm. In [13,14], the authors try to improve the error rate of the covert channel in [5] by solving the thermal interference barrier. However, the covert channel limitation was defeated in ThermalBleed [1], which constructed the first software-based thermal side-channel attack from Linux userspace. It leverages Intel's digital thermal sensors on each CPU core to measure temperature under various conditions. Using this method, hits and misses in cache together with TLB address translations were identified, allowing for the breaching of the KASLR (Kernel Address Space Layout Randomization). Another side-channel attack on mobile devices was created in [15], wherein a CNN which was based on computer vision was employed to foretell the passwords of different mobile devices. Thermal traces were used while the passwords were encrypted using an AES algorithm and then they were converted into images. By inputting the resulting images into the CNN network, they were able to predict the passwords. Other techniques test the predisposition of GPUs to leak information via temperature, power or frequency, by mounting a hybrid side-channel attack [16].

1.2.2. Syscall-Based Attacks and Countermeasures

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. 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.

Sysverify [17] 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 [18] 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 [19] 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 [20], which utilizes syscall patterns to identify anomalous events in comparison to the normal algorithms used in cryptocurrency operations. Other filtering techniques are SysXCHG [21] which can adapt during execution, Nimos [22] which explores syscall sequences, and Optimus [23] which identifies the syscalls which are essential for containers, increasing their security.

1.3. Hot-n-Cold Technique

Hot-n-Cold [2] is an anomaly detection technique which checks the correlation between an original command for Linux and an augmented one. Two Linux terminal commands were selected for the experiments, namely *ls* and *chmod*, which were modified by adding the *ioctl* syscall. The method by which they were chosen has two important factors. Firstly, the authors extracted a list with the most used Linux terminal commands from a local system during one month and created a top 50. Additionally, they selected a list of the top most used in attacks and crafted system calls presented in the CVEs [4]. Secondly, from the top of the commands, they extracted the ones that could be augmented with *ioctl* syscall, meaning that they already have a file descriptor in their implementation that can be used as a parameter for *ioctl* call, a list form in which only four commands remained, namely *ls*, *chmod*, *mv*, and *cp*. After the augmentation, only the two commands mentioned above continue to work as expected. The experiments were performed using scripts which run for about 5 min each, containing the calls of the respective command, with a pause of 10 ms, 100 ms, or 1000 ms between them, while the temperature was read every 2 ms. For each experiment, 30 traces were used for the final result. For the final results, they used Pearson correlation to compare the original and augmented command temperature vectors. They found a correlation up to 0.80 between the two commands, with a higher temperature for the augmented ones, leading to the conclusion that one can detect the anomaly based on the temperature footprint.

2. Materials and Methods

2.1. Background

2.1.1. Linux Commands and System Calls

Linux terminal commands are text-based instructions 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. For our experiments, we have selected *ls* and *chmod* commands based on the research performed in [2], and the augmentation was achieved with the *ioctl* syscall.

A 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 [24].

Computer systems have two operation modes, due to security reasons: kernel mode and user mode [25]. The kernel space is a highly privileged area of memory where the core components of the operating system (OS) operate. Access to this space is restricted and requires specific privileges because it contains critical system code and data structures that manage the overall functionality and resources of the computer. In contrast, user space is

where user applications run. These applications are generally unprivileged, meaning that they do not have direct access to the hardware or critical system functions. Instead, they operate within a controlled environment that limits their capabilities to prevent them from inadvertently or maliciously affecting the system's stability or security [26].

The syscalls are important because they bridge the gap between the low-level functions handled by the kernel and the higher-level operations performed by user applications [27]. They can be accessed via assembly functions, direct call, or C libraries [28]. The first two are less secure because there is a lot of context switching going on, e.g., while printing words there will be a context switch for printing every letter, whereas calling syscalls from libraries use a buffer and fewer context switching [19]. However, the multiple context switching can lead to security breaches.

2.1.2. Thermal Sensors

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, have the advantage of being able to accurately read the temperature of the respective core [29]. For each core, the temperature is read from a place near the transistor junctions [30,31], as seen in Figure 1.

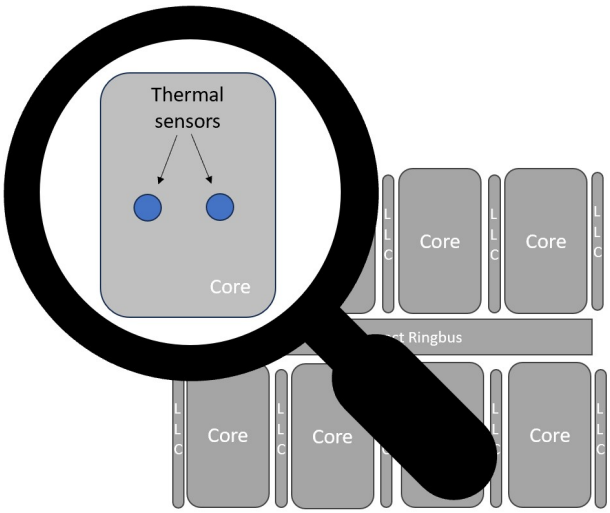

Figure 1. Layout of Intel Core i7-10700's architecture and sensors.

The thermal management of the system is very important to avoid hardware damages. In case a critical temperature is detected, an urgent shut down of the system will take place [29]. The CPU package and its layers can be seen in Figure 2.

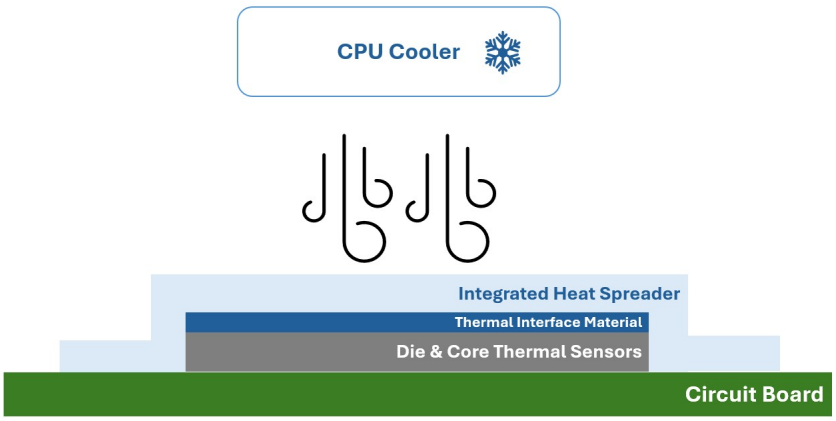

Figure 2. CPU package.

2.1.3. *Hwmon* Interface

The *coretemp* kernel driver enables the reading of digital temperature sensors (DTSs) which are present in all Intel Core CPUs. Using the temperature sensors which are placed on the cores, it will return the temperature of every core and the temperature of the package and write them inside a file which can be accessed via *hwmon* interface. The temperature is measured in degrees Celsius, having a resolution of 1 °C, and can be located under the file system path */sys/class/hwmon/hwmon[*]* in Linux [32], a file which is updated every 2 ms [1].

2.2. Methodology

2.2.1. Experimental Setup

Our 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-150*), and the second one has an Intel Core i5-4590 3.30 GHz with 4 physical cores (which will be further addressed as *D-151*) and the third one an Intel Xeon W3550 3.07 GHz CPU with 4 physical cores (which will be further addressed as *D-152*). All of them have the same and exact OS and installed programs, achieved using Clonezilla [33], 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.

2.2.2. 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. Another factor in choosing them is that they contain file descriptors in their original implementation, which will help us alter them as little as possible, using the already existing file descriptor as a parameter for the augmentation with the *ioctl* syscall.

For the original commands, we used the built-in commands and for the augmented ones, we used the code sources of the original commands from GitHub repository [34], downloaded in February 2023.

To augment the original commands, we 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 per CVE [4] reports in the last 8 years.

The augmentation was performed by calling the *ioctl* C function near the usage of a file descriptor in the original code. The augmentation was performed slightly different than in the original study by minimizing the number of calls of the syscall. Moreover, we tried to avoid using similar data in the code to bypass cache references, since we 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 code for *chmod* command.

2.2.3. The Scripts

The scripts are created in a manner to avoid, as much as possible, any additional syscall or command other than the ones needed for experiments. We mainly use two scripts, one to run the experiments and the second to read the temperature of the cores. The first script *runCommands* contains the Linux command on which the experiments are performed (e.g., *ls* and *chmod*, the original or the augmented version), and a sleep command which will let the CPU take a break between running the commands. These two commands were copied for 600 times each, to avoid using a *for* loop, which would have introduced an additional syscalls. The second script *readTemp* reads the temperature of each core repeatedly with a pause of 2 ms between each reading. The rule of avoiding additional syscall is not used for this script, since it is run on another core and does not influence the

temperature emanated by the commands. For both scripts, we used the *taskset* command to run them on different cores.

2.2.4. The Noise

We wanted to bring the technique from a controlled environment, where one node of the HCP 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 which can listen to music, move files from one folder to another, surf the browser, or write on the keyboard. The *playSong* script is starting the *ffplay* player [35] and plays the Baby Shark Dance song, which is the most viewed video on YouTube. The *moveFiles* script is moving a file from one folder to another in a repeated way. The *startBrowser* script is simulating surfing on the web, opening a list of websites in the Firefox browser. The *simulateKeystrokes* script is simulating the writing of a text with a pause of 100 ms between each key press. The last script *mathComputations* is running different algorithms like division by 6, the sum of series, perfect square, sum of two polynomials, binary to hexadecimal conversion, and the kth prime factor, the factorial of a given number and Fibonacci series.

2.2.5. Methodology of the Experiments

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

As already mentioned, we used three different desktop systems (i.e., *D-150*, *D-151*, and *D-152*) for our experiments, all placed in a controlled temperature environment. Unlike other studies (i.e., [1]), we did not fix the frequency of the desktops, nor did we disable the Intel Turbo Boost, which makes the experiments more similar to a real-life scenario.

First, we started the experiments by measuring the baseline temperature of the systems (see Figure 3). 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 as one can see which is the idle temperature for each core. Additionally, we checked all the deamons 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.

Based on the baseline temperature of the systems and the architecture of the CPUs, we selected the cores on which the experiments are to be run. For each system, we selected two cores with a lower temperature and physical location on the die to be as far as possible from each other, such that the two running scripts do not influence each other. In Figure 4, 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 [36].

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

Next, after the results were reproduced, we continued to run the above mentioned scripts; however, this time, we started to introduce the noise. The experiments which included the noise were run using two different scenarios. The first scenario includes running the noise script on the same core as the core where the commands run, which

leads to a higher noise on that specific core. The second scenario includes running the noise script without selecting a CPU affinity, and let the OS to choose where to run; in this way, we can have a different type of background noise.

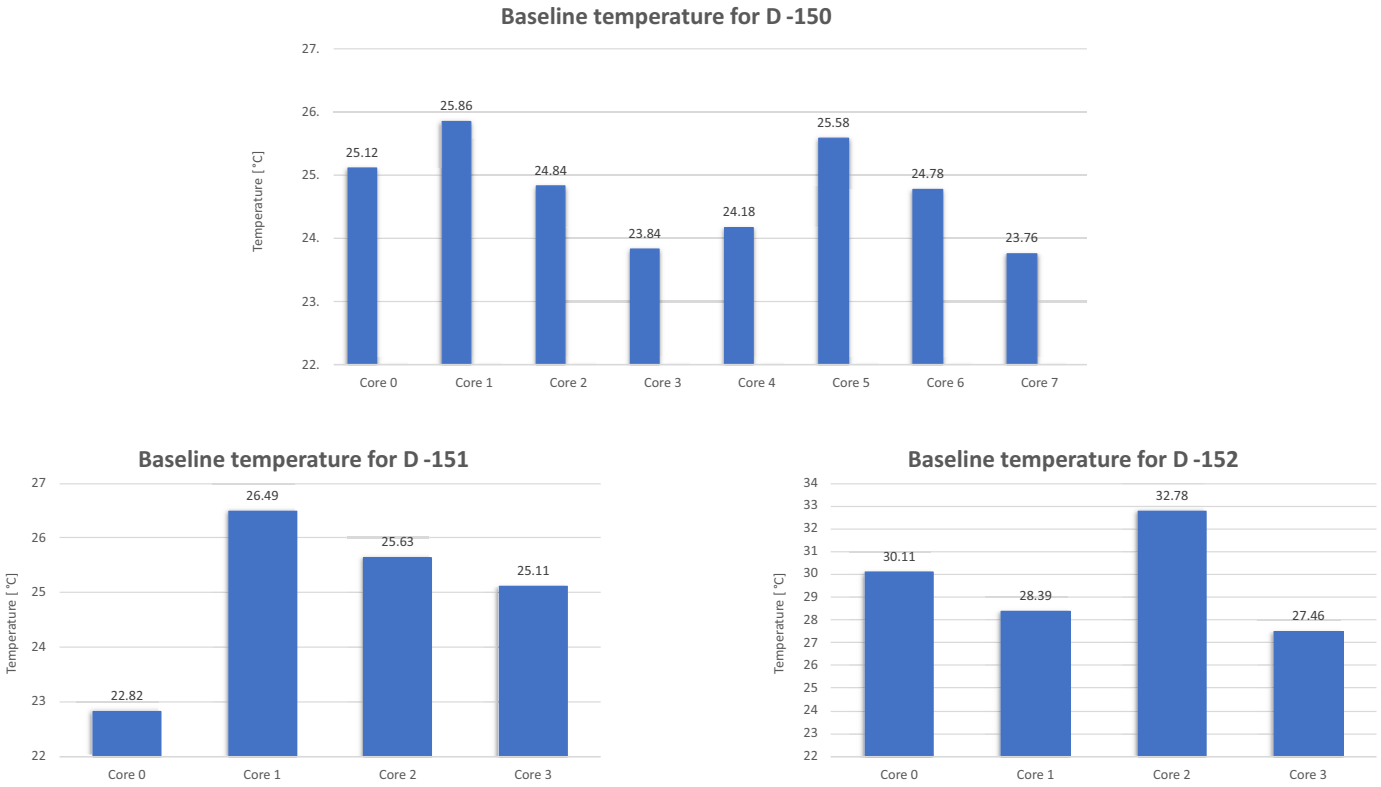

Figure 3. Baseline temperatures of the three systems.

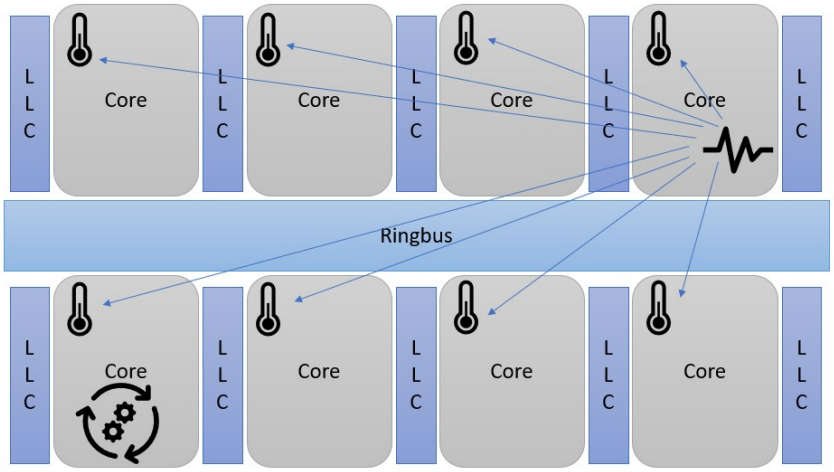

Figure 4. Methodology of the experiments.

3. Results

This 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. We will further present graphs with the results, containing a comparison of the vectors of temperatures for each command in both its states. In the following graphs, the blue line represents the array of values for the authentic command and the red line represents the vector of values for the modified command, augmented with *ioctl* calls. All the figures contain the average temperature of the commands 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, we used the Pearson correlation index. A high correlation is considered if the Pearson correlation index is above 0.7 [37]. 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.

3.1. Reproducing Original Results

First, we aimed to reproduce the results from the original paper and to check whether the selected cores are a good option. In Figures 5–7, there are six plots, one for each command (i.e., *ls* and *chmod*) run on all systems (i.e., *D-150*, *D-151*, and *D-152*).

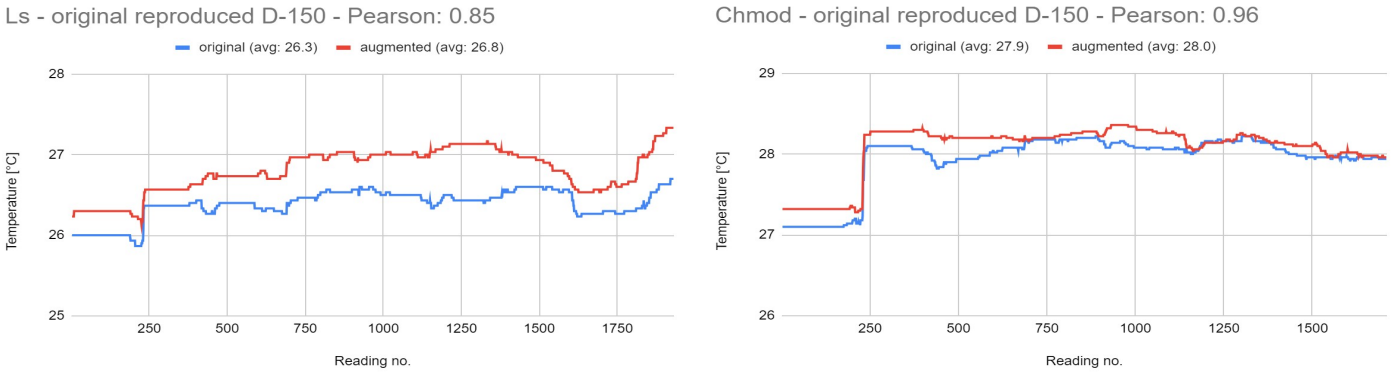

Figure 5. Reproduced results for *ls* and *chmod* commands on D-150.

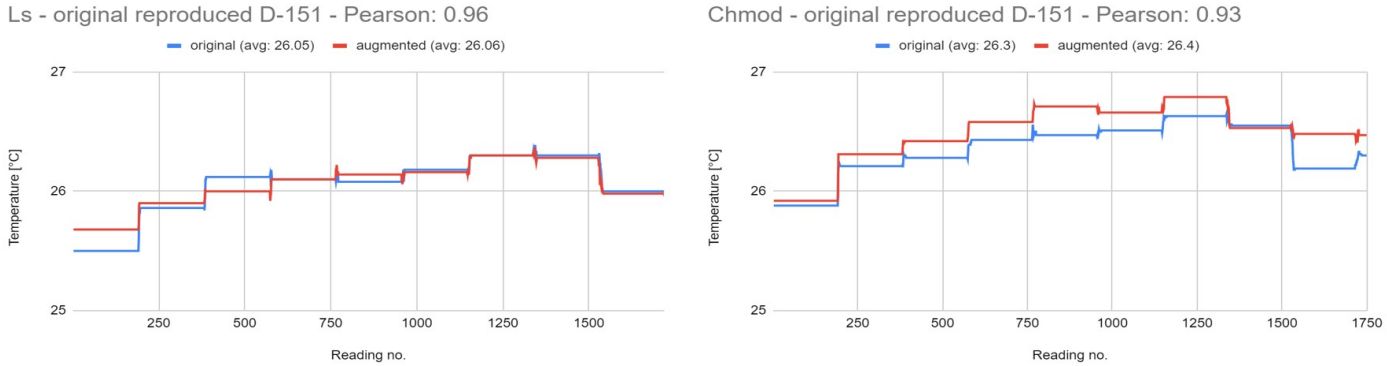

Figure 6. Reproduced results for *ls* and *chmod* commands on D-151.

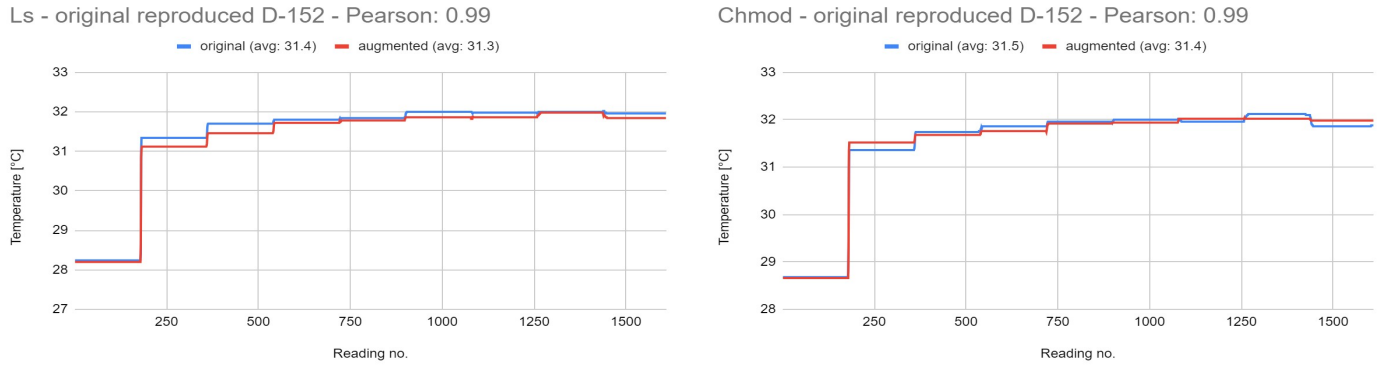

Figure 7. Reproduced results for *ls* and *chmod* commands on D-152.

For *D-150*, 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 authentic and the modified 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 experiments was a successful one, and an anomaly was detected in the behavior of the command.

For the other two desktops, *D-151* and *D-152*, the results in terms of Pearson correlation are all over 0.93. However, they are not so conclusive in terms of difference in the average temperature where we can only see 0.1 °C (see Figures 6 and 7). We attribute these results to the older CPUs, which have thermal sensors with lower accuracy, and a small number of physical cores which can influence each other through heat dissipation. Further, we will present the results of *D-151* and *D-152* systems in the Appendix A.

3.2. Results Using CPU Affinity

As we already mentioned, when we introduced the noise, we 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.

3.2.1. Noise: Playing Song

The experiments where the song was introduced lead to high correlations (see Figure 8) of 0.94 for the *ls* command and 0.98 for the *chmod* command, respectively. We can see a difference in the average temperature between the authentic and augmented commands up to 0.4 °C, as a result of which the augmented command spreads more heat. The high correlation in the pattern of the commands and the augmented command having a higher temperature, lead to the conclusion that an anomaly was found. However, for the *chmod* command, we see more accurate overall results.

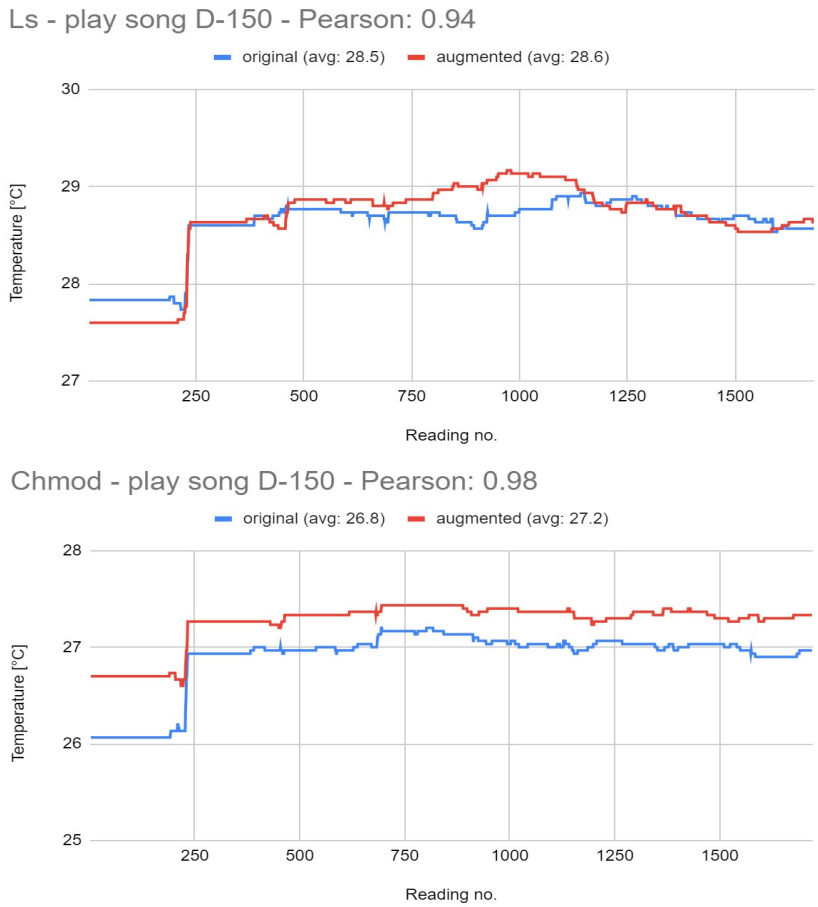

Figure 8. Results with playing song as a noise, selecting CPU affinity.

3.2.2. Noise: Moving Files

For moving files as a noise, we performed two experiments, one with a big file of 18.8 GB in size (results in Figure 9) and one with a small file of 3.3 MB in size (results in Figure 10). As seen in Figure 9, 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 there is or not an anomaly taking place.

However, the results for moving the small file are not necessarily the ones expected (see Figure 10), 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 we attribute these results on the high probability of caching the small file.

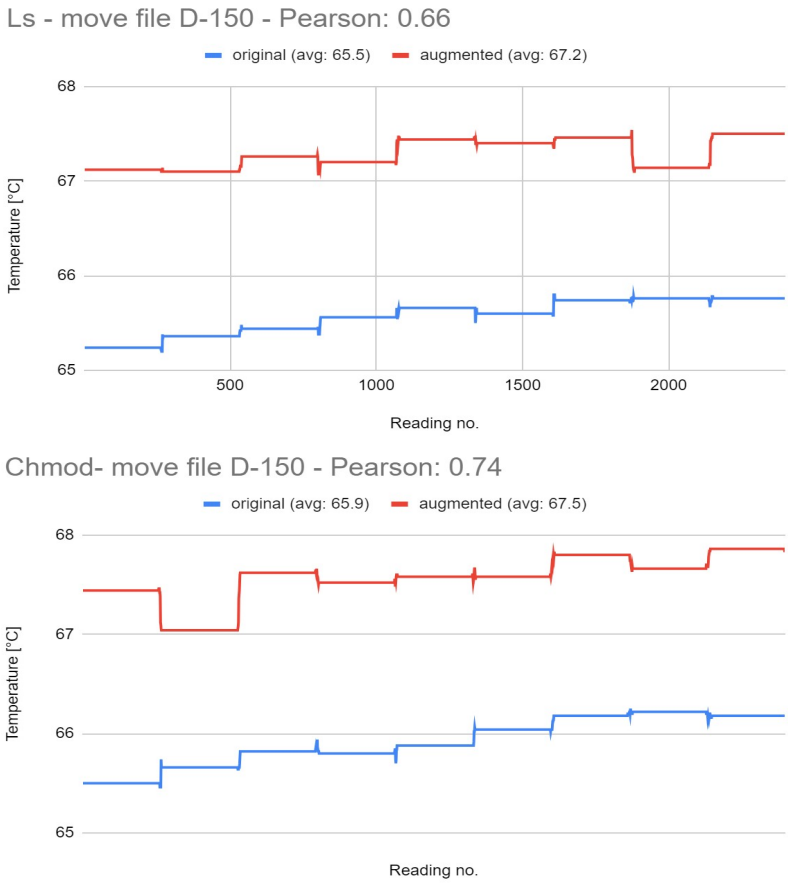

Figure 9. Results with moving a big file as a noise, selecting CPU affinity.

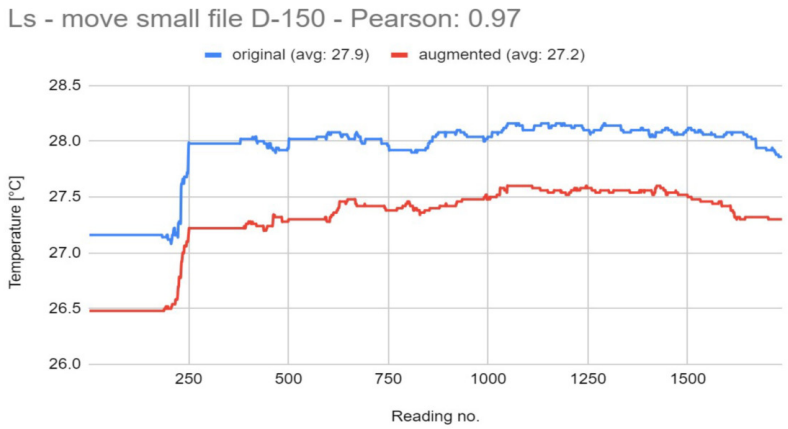

Figure 10. *Cont.*

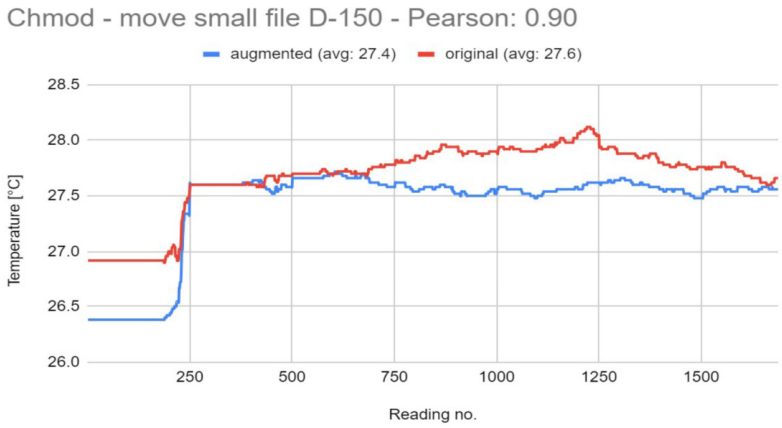

Figure 10. Results with moving a small file as a noise, selecting CPU affinity.

3.2.3. Noise: Browser Surfing

In the browser surfing script, we repeatedly open different sites as Google, Google Maps, StackOverflow, Ubuntu, Geeks for Geeks, Wikipedia, or IEEE, using Firefox browser. The results can be seen in Figure 11, where there is a high correlation of 0.85 for the *ls* command and a good one of 0.75 for the *chmod* command. For both commands, the average temperature for the augmented command is higher with 0.3 °C. Thereby, 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 taking place.

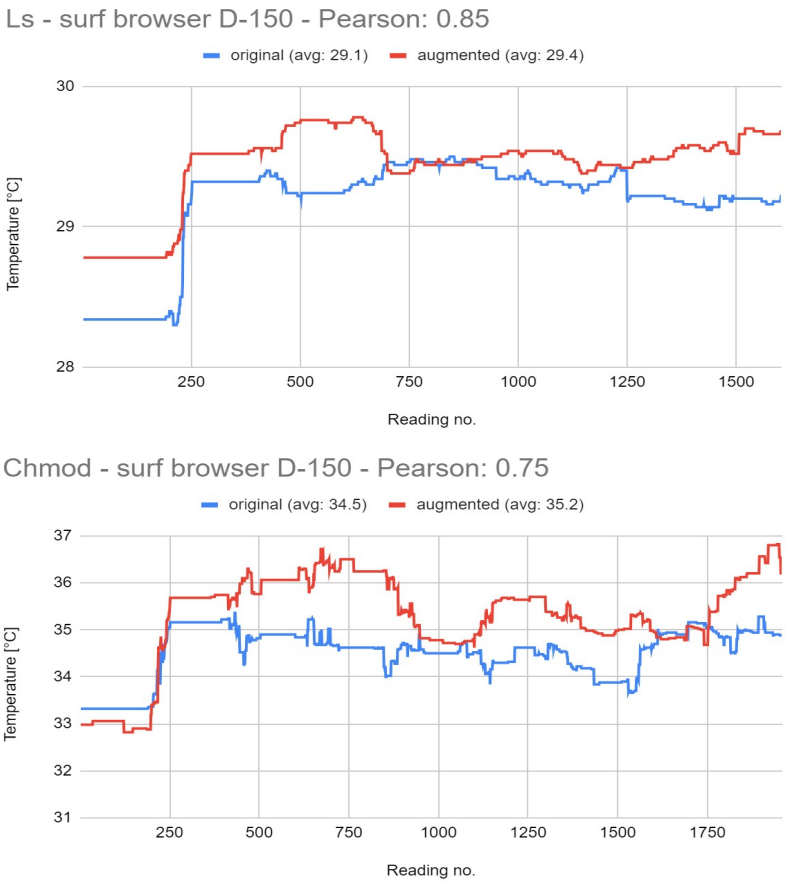

Figure 11. Results with surfing the browser as a noise, selecting CPU affinity.

3.2.4. 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 12.

Moreover, the average temperature between the authentic and augmented command is 0.2 °C for the first one and −0.1 °C 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 can not.

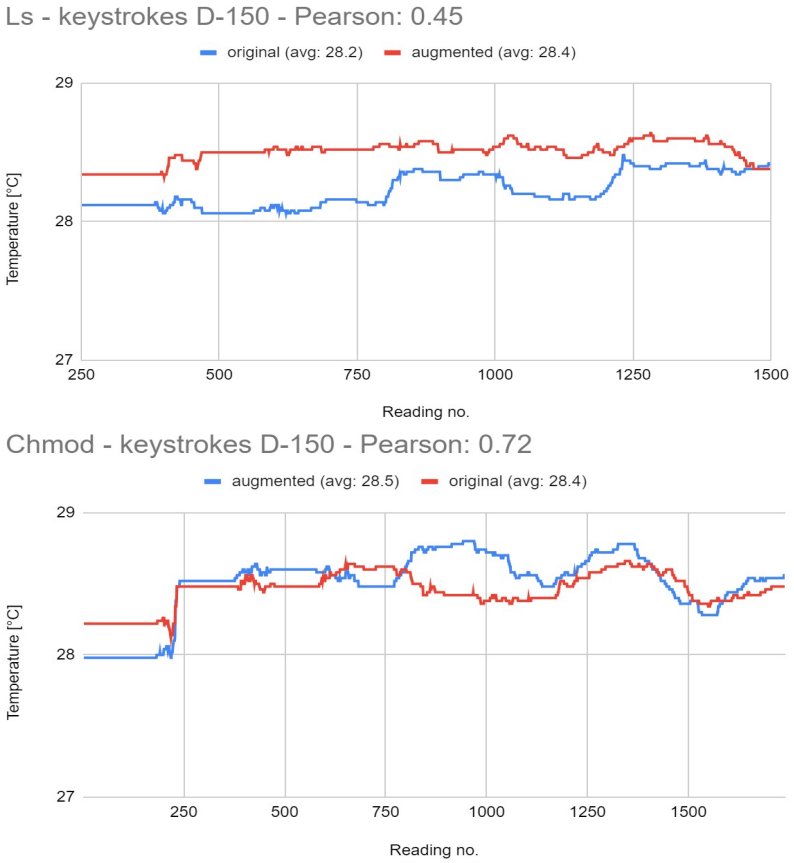

Figure 12. Results with keystrokes as a noise, selecting CPU affinity.

3.2.5. Noise: Mathematical Computations

Similar results to that in the keystroke experiment are seen in the mathematical computations in Figure 13. 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.

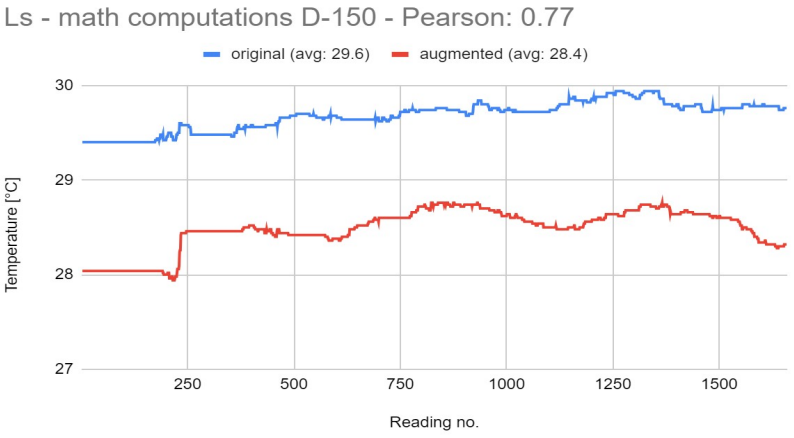

Figure 13. *Cont.*



Figure 13. Results with mathematical computations as a noise, selecting CPU affinity.

3.3. Results without Using CPU Affinity

In this subsection, we will present the results while not selecting CPU affinity for the noise, instead letting the OS decide where to run.

3.3.1. Noise: Playing Song

For the experiments where the noise consists of playing the song while the CPU affinity was not set, it can be seen in Figure 14 that the temperature for the augmented command is lower than the original one, and results were reproduced on almost all the experiments where we did not select a specific core for the noise. This happens most probably because the OS used one of the optimization methods. Still, the Pearson correlation index has good values, 0.7 and 0.8, respectively, with an average difference between temperatures of 0.3 °C.



Figure 14. Results with playing song as a noise, without selecting CPU affinity.

3.3.2. Noise: Moving Files

In the experiments where the noise was moving a file, without selecting the affinity of the CPU, we used the small-sized file. In terms of correlation, we have a high correlation of 0.88 and 0.93, respectively, which shows a high similarity in the pattern of execution, which can also be visually observed in Figure 15. Again, the difference in temperature is 0.2 °C for both experiments, where the original command has a higher temperature.

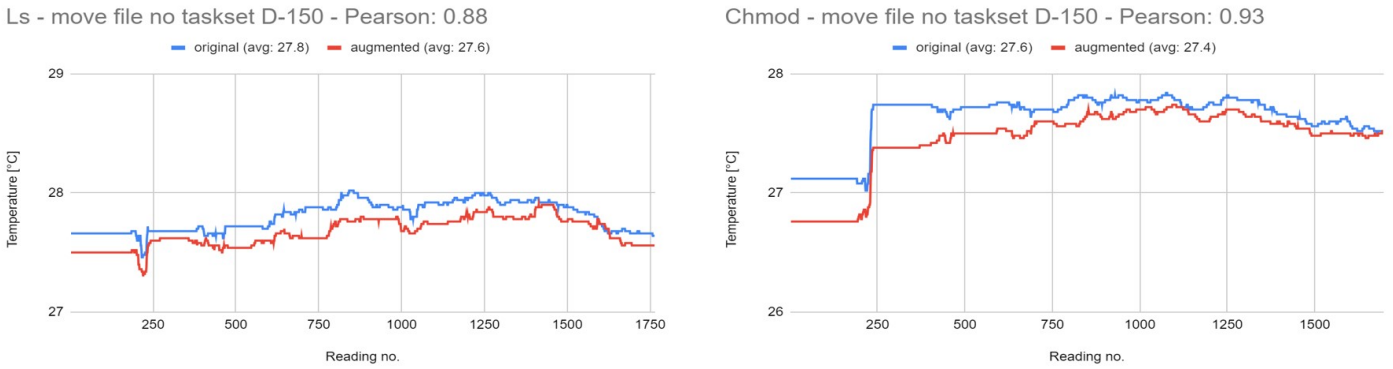

Figure 15. Results with moving the small file as a noise, without selecting CPU affinity.

3.3.3. Noise: Browser Surfing

In the experiments where we used surfing the browser as a noise (see Figure 16), we again saw that most of experiments have the original command with a higher temperature than the augmented one. However, the Pearson correlation indexes have good values up to 0.77.

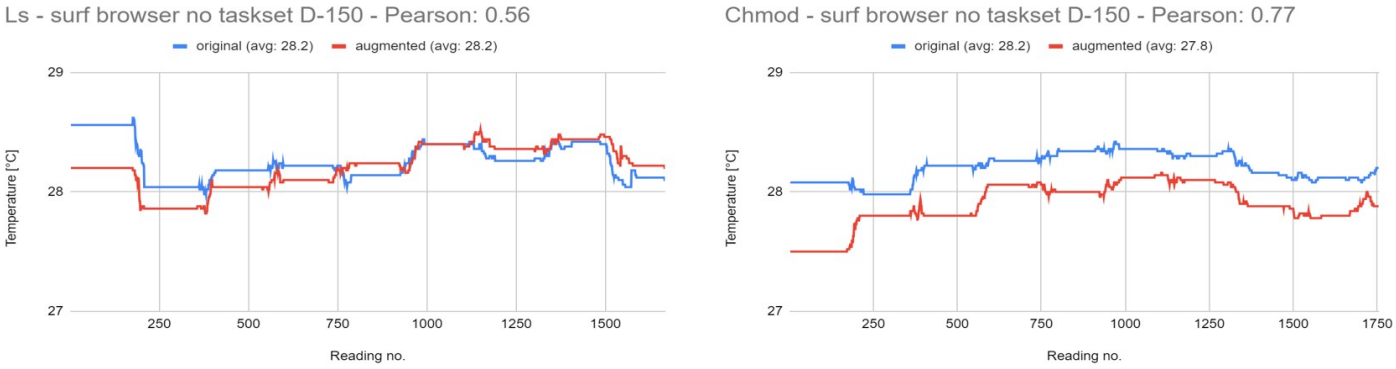

Figure 16. Results with surfing the browser as a noise, without selecting CPU affinity.

3.3.4. Noise: Keystrokes

While using the keystrokes as a noise, we saw better results for the *chmod* command (see Figure 17), 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 an anomaly detection.

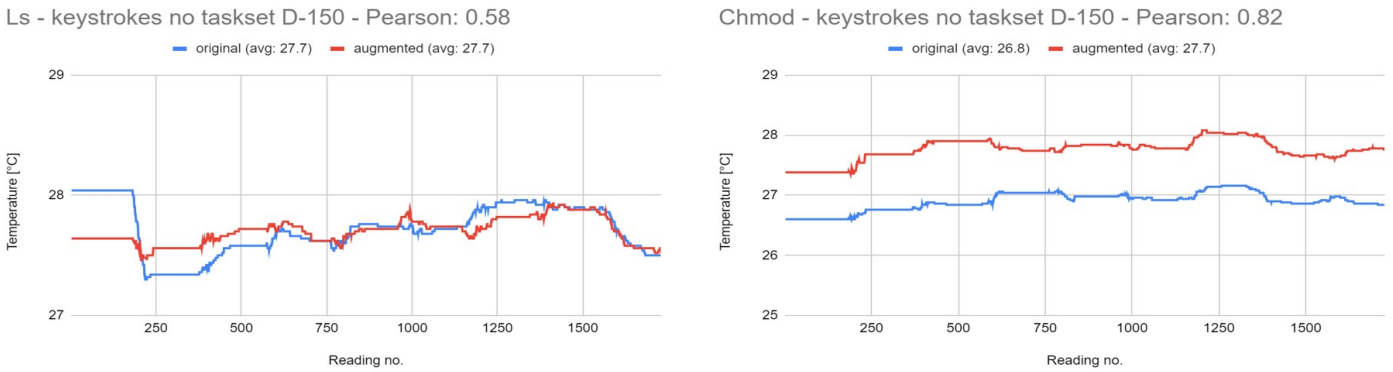

Figure 17. Results with keystrokes as a noise, without selecting CPU affinity.

3.3.5. 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 18. For *chmod*, the correlation index shows a value of 0.68, while the difference between average temperature is half a degree Celsius. Even though the correlation for the *ls* command is a high one of almost 0.80, the temperature of the original command is visibly higher.

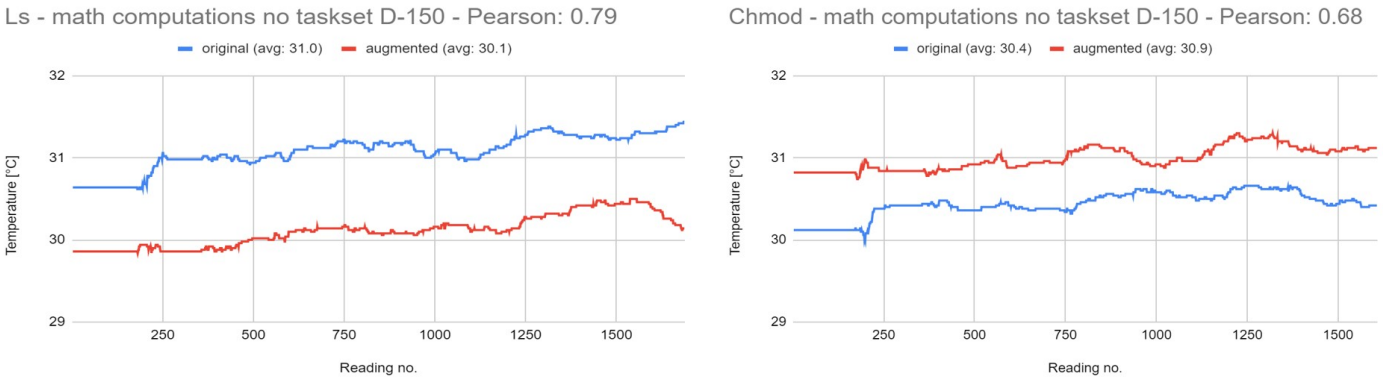

Figure 18. Results with mathematical computations as a noise, without selecting CPU affinity.

4. Discussion

4.1. Anomaly Detection

In this paper, we propose *BeatTheHeat*, a technique that performs anomaly detection through the thermal sensors of the CPU, aiming to distinguish between an authentic Linux command and an augmented one, while a lot of noise appears on the CPU.

In comparison with the initial study, we can see a similar difference in the average temperature of the commands, between 0.5 and 1 degrees Celsius. However, the correlation between the patterns of the original and augmented command is higher in this study. In the case of *ls* command, in this paper, we have a higher correlation of 0.85, instead of 0.73 as in the Hot-n-Cold paper [2], while for the *chmod* command, the correlation was 0.96 instead of 0.83. These results are better due to the increased number of traces for the thermal analysis used in this paper. Hence, a good correlation (above 0.80) leads to the conclusion that the commands are very similar in their pattern, in both their forms (i.e., original and augmented), which makes it possible to detect an attack taking in consideration their temperature values. An abnormal (i.e., higher) temperature for a specific command leads to the conclusion that it has additional code inserted.

As we already stated in the results section, more accurate results were achieved using the *D-150* system. We attribute these results to the year of manufacture of the processor and the larger number of cores. Furthermore, we can see a difference between *D-151* and *D-152*, with better results for the first one, the year of production being again a reason for better results, in our opinion.

This technique is directed towards any system that uses an Intel CPU; however, it can also be extended to any other device, including IoT devices that have and use thermal sensors, from which the temperature can be retrieved.

4.2. Anomaly Detection Using Machine Learning

To increase the effectiveness of our technique, we further implement a Machine Learning (ML) algorithm to help us decide whether we encounter an anomaly or not. The chosen algorithm is K-Nearest Neighbors (KNN), which helps us 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 above experiments.

We used the algorithm 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, we used the number of nearest neighbors, with k having a value of 15, together with the Euclidean distance.

In Figure 19, there are the confusion matrices for *ls* (left) and *chmod* (right) commands. Based on these matrices, we calculated some metrics, as can be seen in Table 1.

Table 1. ML 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

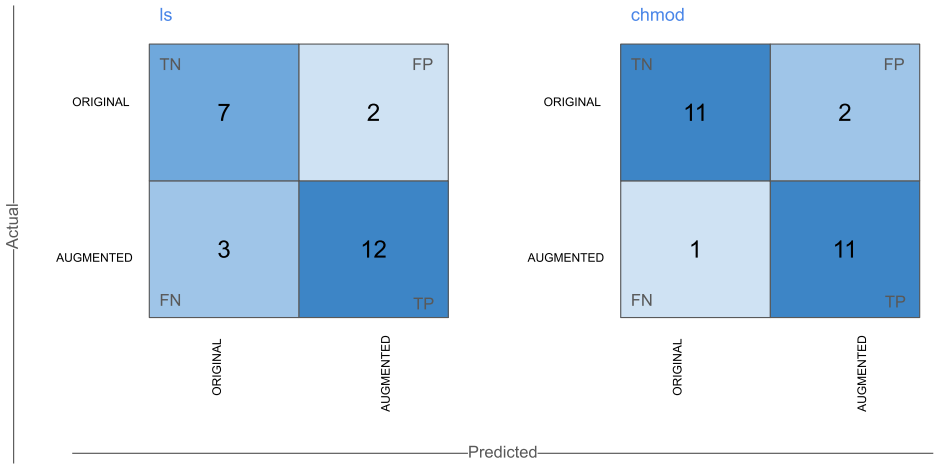

Figure 19. Confusion matrices.

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

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

4.3. Limitations

The nature of the CPU sensors used in this study leads to some limitations in terms of the methodology and the approach. The low resolution of the sensors, of 1 °C, is insufficient for distinguishing the temperatures of individual syscalls, as we originally intended to use in our experiments. Thus, taking in consideration the time required for a syscall to execute (i.e., nanoseconds) and a command to execute (i.e., *chmod* requires 1 ms to execute and *ls* requires 2.1 ms), we selected the commands to continue our experiments with. However, arrays of commands needed to be used, due to their extended execution times.

Another type of limitation is the heat propagation through nearby cores. To avoid heat propagation, we selected the two farthest away cores for each of the two scripts that were run. Consequently, the script for reading the temperature runs on Core 3 and the script for calling commands runs on Core 7, which we chosen based on their low temperature (see Figure 3) and physical distance on the die.

4.4. On Which Core the OS Runs

Furthermore, we wanted to know on which core the operation system runs. However, we did not find any information in the literature; thus, we 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 question above. We restarted the system for 15 times, and we created the baseline temperature of the cores after each restart. After the analysis of the temperatures, the results can be seen in Figure 20, and we can conclude 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, we can conclude that the OS will start every time on the same cores.

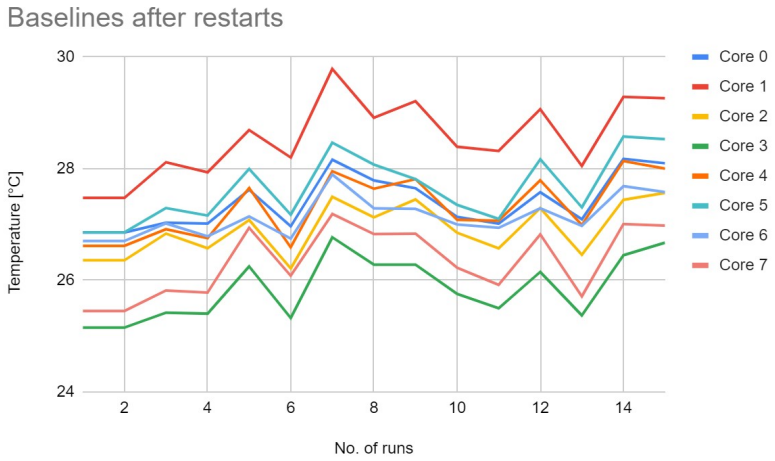

Figure 20. Baselines after restarts of the system.

4.5. Rising and Dropping Temperature

To further test the capabilities of the CPU sensors, we conducted a small experiment regarding the rising and dropping temperatures of a core on the *D-150* system. More specifically, we used a CPU stressing program, Stress-ng [38], which increases the CPU temperature to a high one due to the CPU load of 70%. The temperature was read every 500 ms to see how fast it rises and drops after the program is finished. In Figure 21, on the horizontal axes, the readings of the temperature are represented, which means that between three numbers is one second. Thus, we start at a temperature of 24 °C and in half a second the temperature rise to 34 °C, which means 10 degrees Celsius in 500 ms. Furthermore, in 1.5 s (between reading numbers 60 and 63), the temperature increases by a total of 18 °C. In the case of cooling down, we start at a temperature of 40 °C, and in 1 s, the temperature drops with only 2 °C. However, after another second (between readings 100 and 102), the temperature drops with 11 °C, which leads to the conclusion that the cooling down is slower than the heating up.

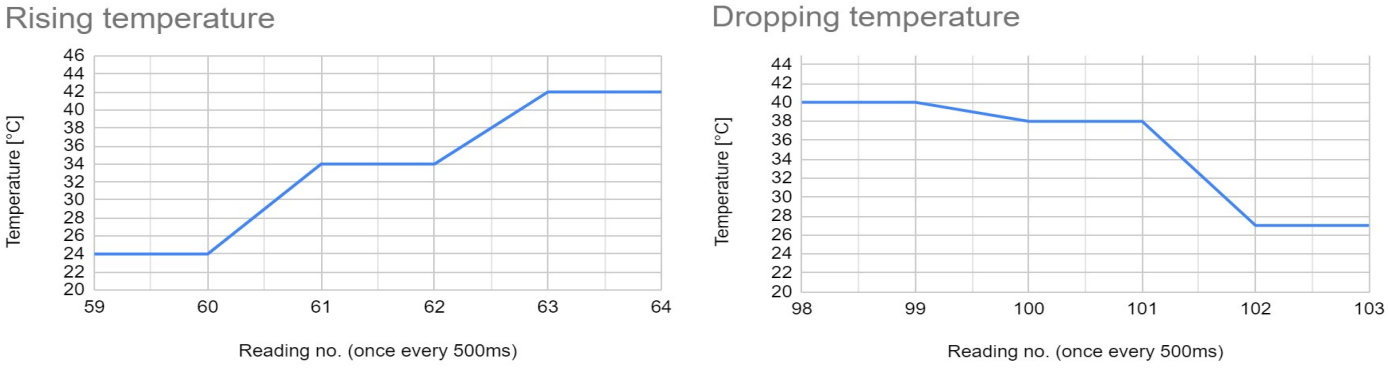

Figure 21. Rising and dropping temperature of a core.

4.6. 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, we conducted an experiment to further see how the heat propagates through cores and analyze the physical locations of the cores. We used the stressing CPU program *Stress-ng* [38] on Core 7 of the *D-150* system to increase the CPU load to 70%. The temperature was read with a script running on Core 0, placing the two scripts as physical far as possible in the CPU. In Figure 22, it can be seen, for each core, the difference between the baseline temperature of the respective core and the temperature measured, while on Core 7, the stress program is running. Therefore, it can be observed that all cores are affected by the high heat (of approximately 50 °C on the peak) of Core 7.

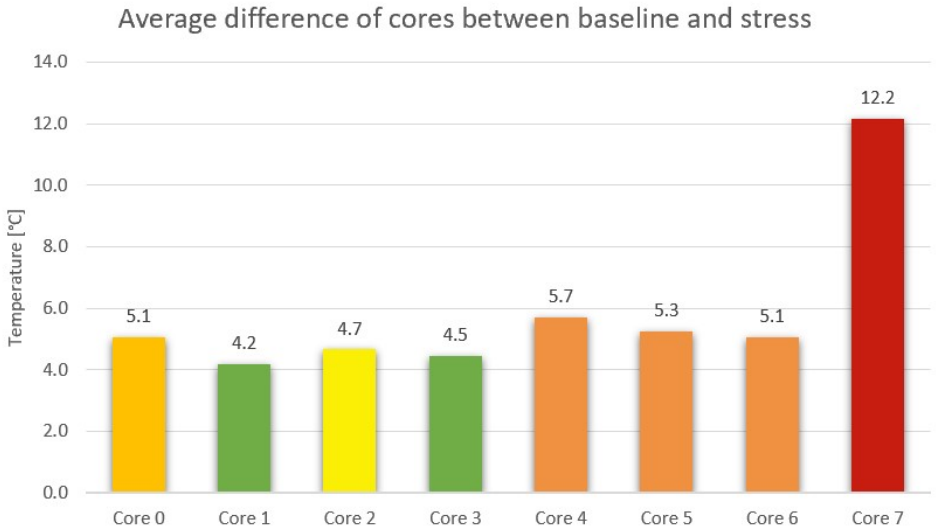

Figure 22. Difference between baseline and run on each core.

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

Taking into consideration what we have stated above, we can assume that the layout of the cores on the CPU die is as seen in Figure 23, where Cores 4, 5, and 6 are the hottest because these are physically placed near Core 7.

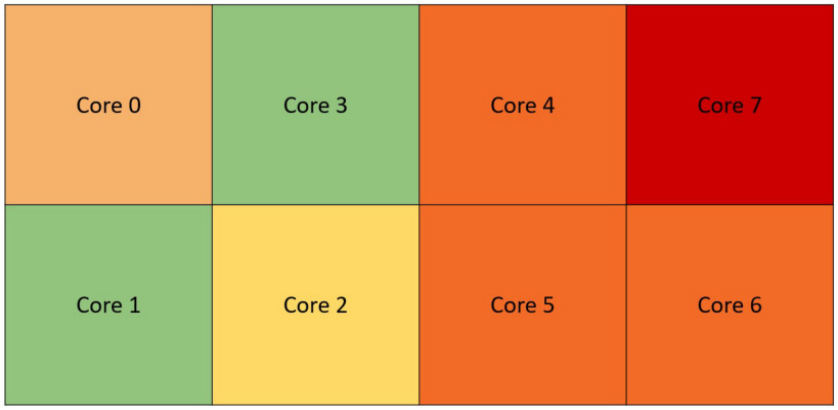

Figure 23. Possible core layout, based on heat propagation experiment.

4.7. Covert Channel Attack

To further expand the research in the direction of heat propagation, we created a small Covert Channel Attack. The idea of a covert channel attack is based on bypassing the current security measures in order to transmit sensitive data without breaking them. In more details, a person (called A) has access to secret data in the interior of an entity; however, due to the 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, we decided that our secret data are 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. We transformed the secret text 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 24, the heat is propagated to the nearby core. We presented both cores in this figure so that the propagation can be seen better. However, B only has access to the data in blue, which represent the propagated data. A further thermal analysis must be applied to these data to extract the secret text. A technique called *binarization* is applied, using a filter which will divide data into values of 0 and 1. After this, we have the binary values, and we will extract the final values by searching for patterns that will help us distinguish the time for transmitting one bit. The results of this experiment are that we were 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.

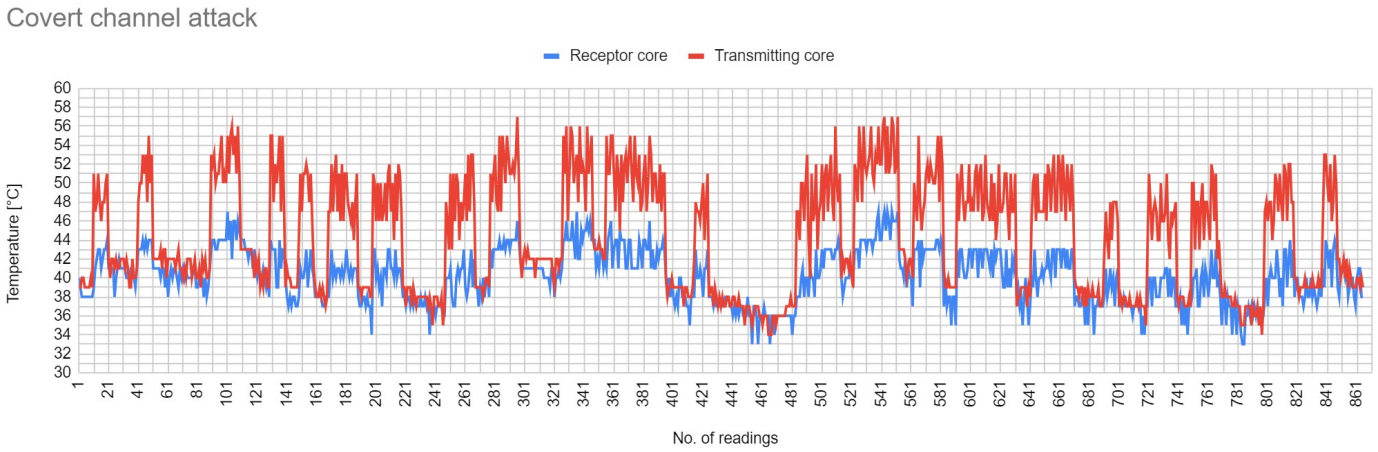

Figure 24. Covert channel attack.

4.8. Future Work

In this technique, we did not manage to detect between commands or syscalls with this small amount of traces. Thus, our future work will include increasing the number of traces together with the number of commands used. Moreover, we will study diverse patterns of system calls that can appear through the execution of a specific command, to get closer to our initial goal. Furthermore, increasing the automation of this technique and integrating the Machine Learning algorithm together with the thermal correlation analysis can in a way make it applicable for the continuous execution and detection of anomalies based on temperature and other factors, which constitutes an important future work.

5. Conclusions

In this work, we have introduced the *BeatTheHeat* anomaly detection technique, applying it in circumstances much closer to the real-world computational environments. We

have demonstrated that it detects irregularities in the Linux command behavior through CPU temperature monitoring, even when noise is in place. Moreover, we have introduced a machine learning algorithm 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.

The experiments were performed on three types of CPUs which show high correlations, approaching 0.96, between the authentic command and an augmented one. This research demonstrates the high potential of detecting attacks through thermal sensors even in a real-world scenario. Nevertheless, the technique can be extended to check for anomalies in the CPU temperature during normal operations.

Author Contributions: The individual contribution for this article is as follows: Conceptualization, T.V. and R.B.; methodology, T.V.; software, C.B. and T.V.; validation, T.V., R.B. and C.B.; formal analysis, T.V. and R.B.; investigation, T.V.; resources, R.B.; data curation, T.V. and C.B.; writing—original draft preparation, T.V.; writing—review and editing, R.B., C.B. and T.V.; visualization, C.B. and T.V.; supervision, R.B.; project administration, T.V. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The scripts for this experiment can be found at https://github.com/teodoravasilas/Beat-the-Heat.git, accessed on 20 August 2024.

Conflicts of Interest: We declare no conflict of interest.

Appendix A. Results

Appendix A.1. Results Using CPU Affinity

Appendix A.1.1. Noise: Playing Song

For *D-151* and *D-152*, we 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-152 are not showing significant results, as can be seen in Figure A1.

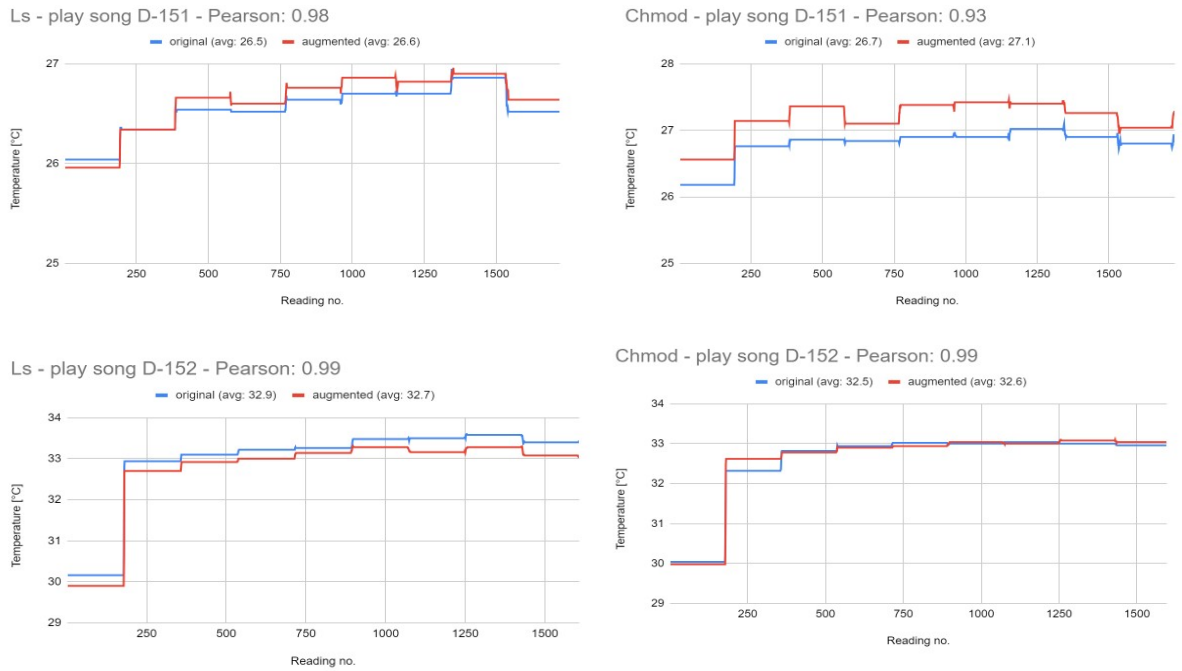

Figure A1. Results obtained with playing song as a noise, selecting CPU affinity.

Appendix A.1.2. Noise: Moving Files

For moving the big file as a noise (see Figure A2), we see good results on the other two desktops as well, with correlation indexes between 0.67 and 0.74 and a difference in the average temperature between 1 and 2 °C.

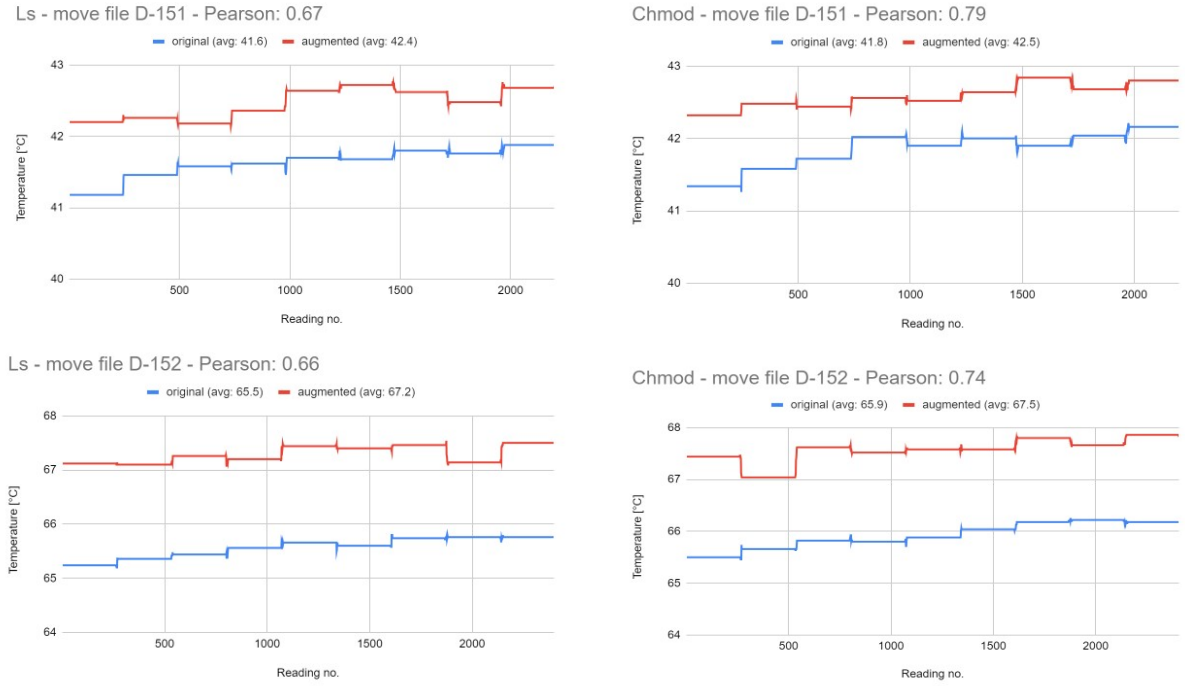

Figure A2. Results obtained with moving the big file as a noise, selecting CPU affinity.

Appendix A.1.3. Noise: Browser Surfing

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 °C. For the *ls* command, the pattern correlation is above 0.80 with a smaller difference between the average of the original and augmented command of 0.2 °C, as can be seen in Figure A3.

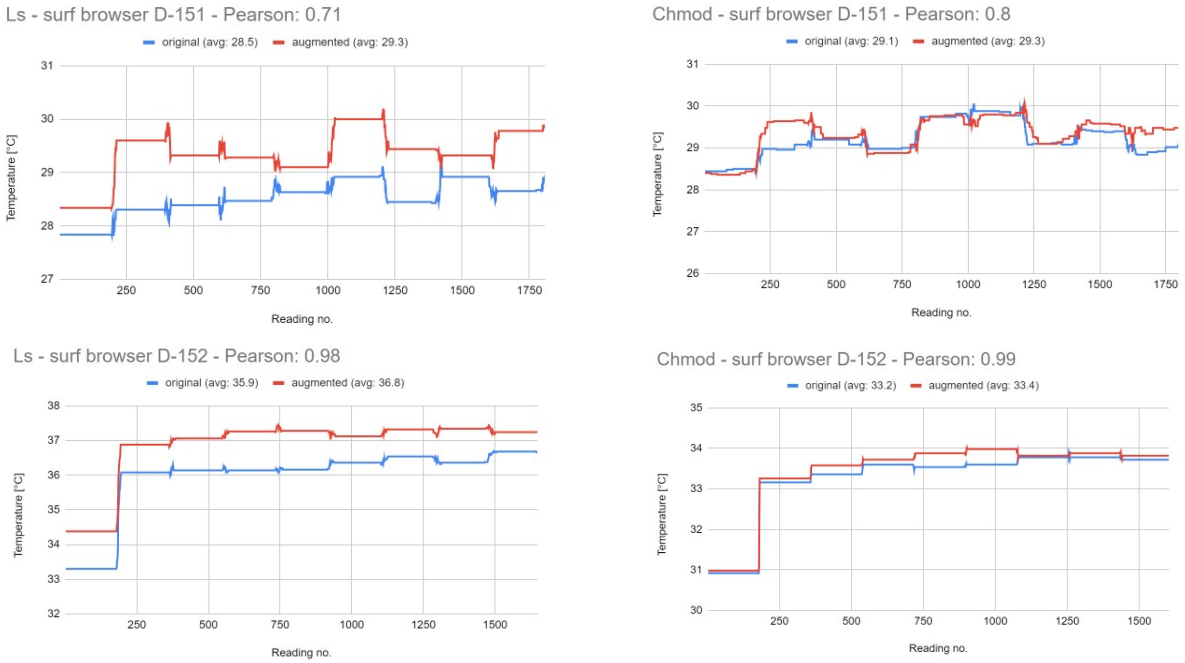

Figure A3. Results obtained with surfing the browser as a noise, selecting CPU affinity.

Appendix A.1.4. Noise: Keystrokes

For the keystrokes, we again see better results for *chmod* command, with a correlation of 0.88 and a difference in the average temperature of 0.3 °C, on *D-151*, as seen in Figure A4.

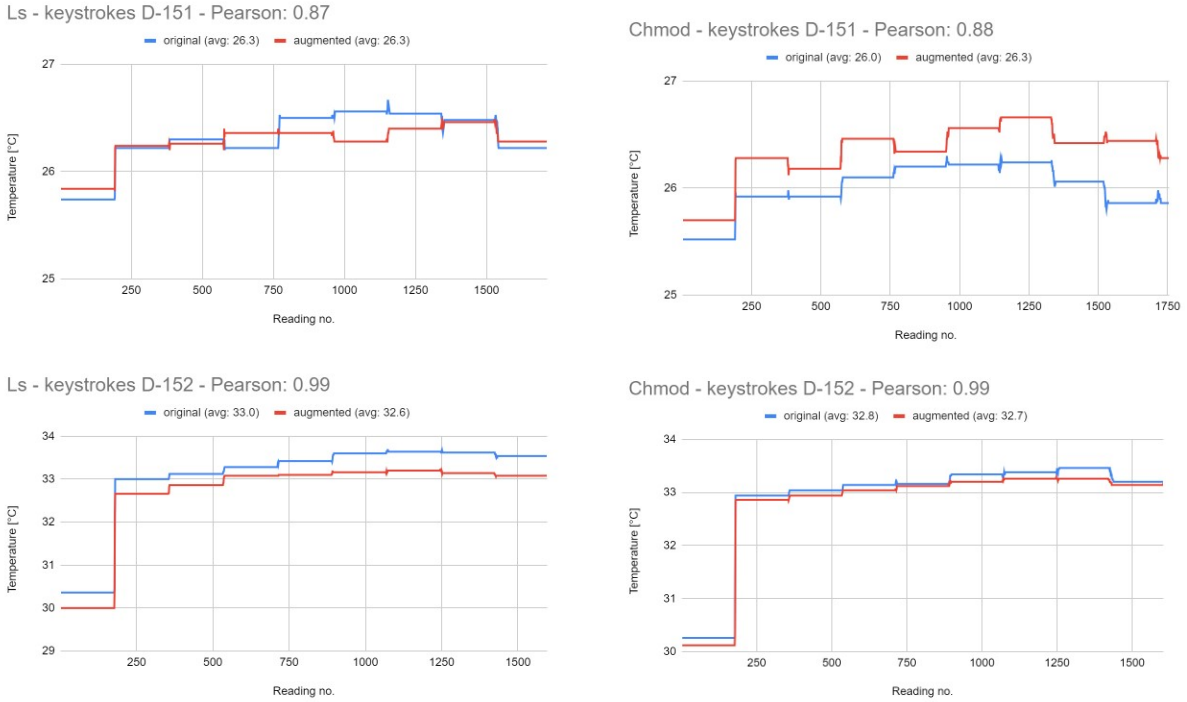

Figure A4. Results obtained with keystrokes as a noise, selecting CPU affinity.

Appendix A.1.5. Noise: Mathematical Computations

In the case of mathematical computations (see Figure A5), as for keystrokes, better results are for the *chmod* command, where the pattern correlation is higher than 88% and the difference in the average temperature up to 0.4 Celsius degrees.

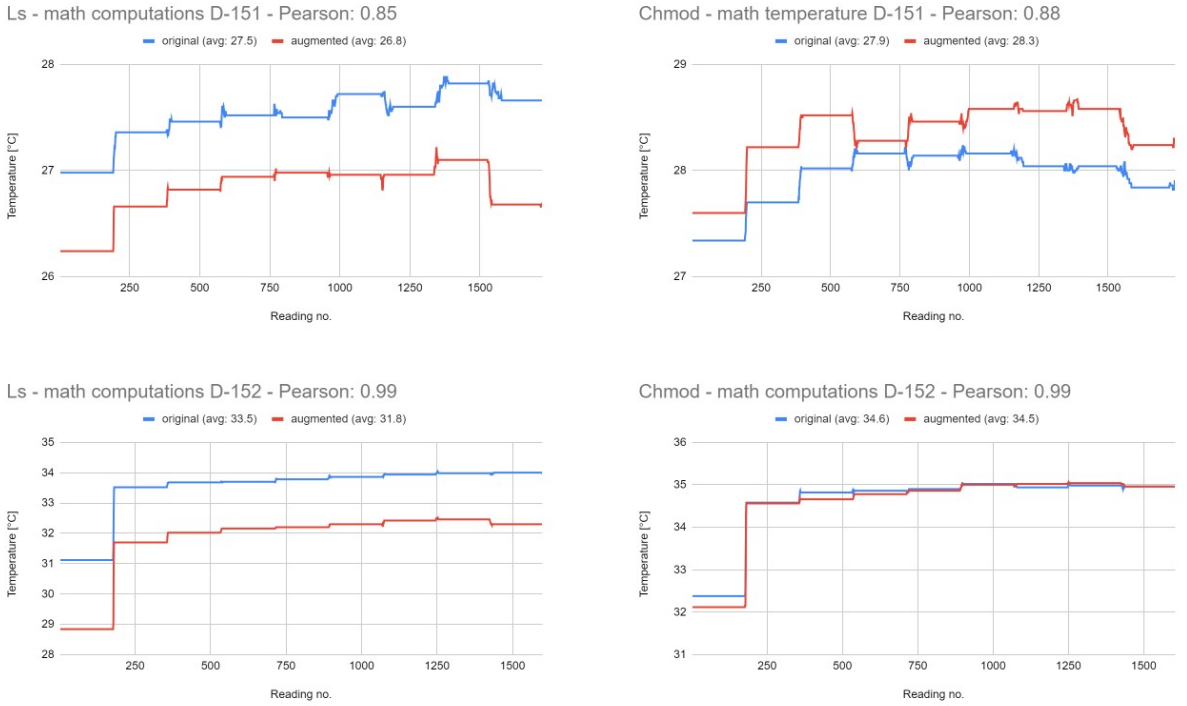

Figure A5. Results obtained with mathematical computations as a noise, selecting CPU affinity.

Appendix A.2. Results without Using CPU Affinity

Appendix A.2.1. Noise: Playing Song

For music playing without selecting on which core to run, we see that the original temperature has higher values than the augmented one, with up to 0.2 °C, and still the correlation has good values as in Figure A6.

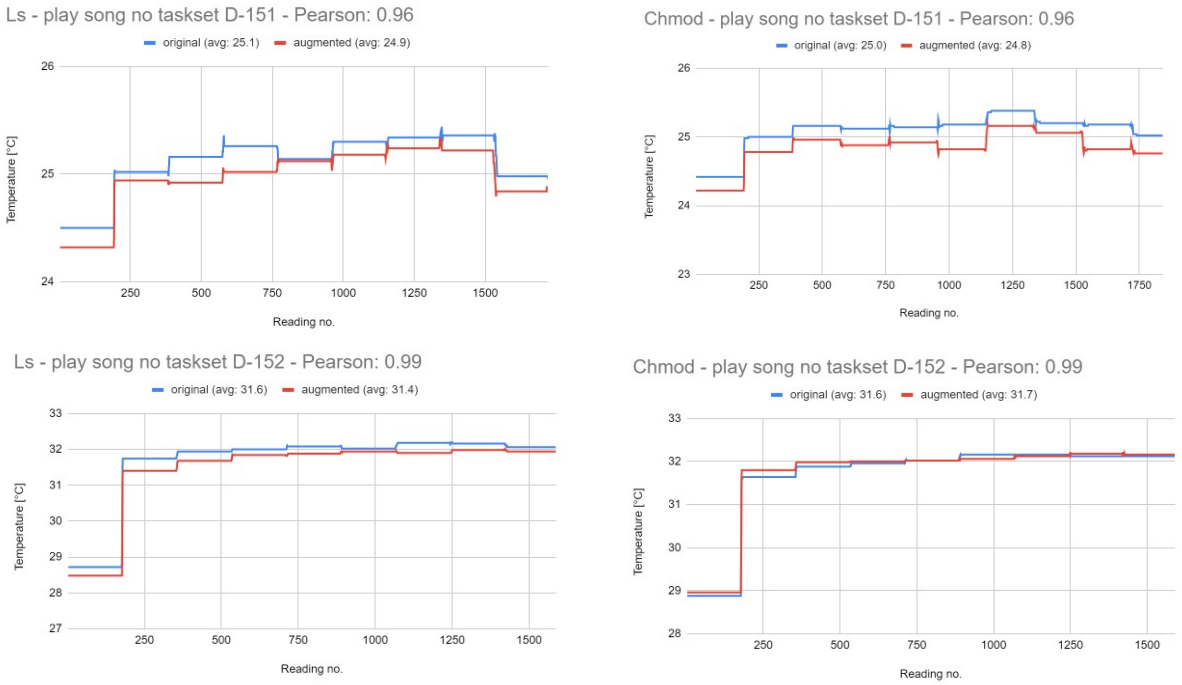

Figure A6. Results obtained with playing song as a noise, without selecting CPU affinity.

Appendix A.2.2. Noise: Moving Files

In the case of moving a small file (see Figure A7), we 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.

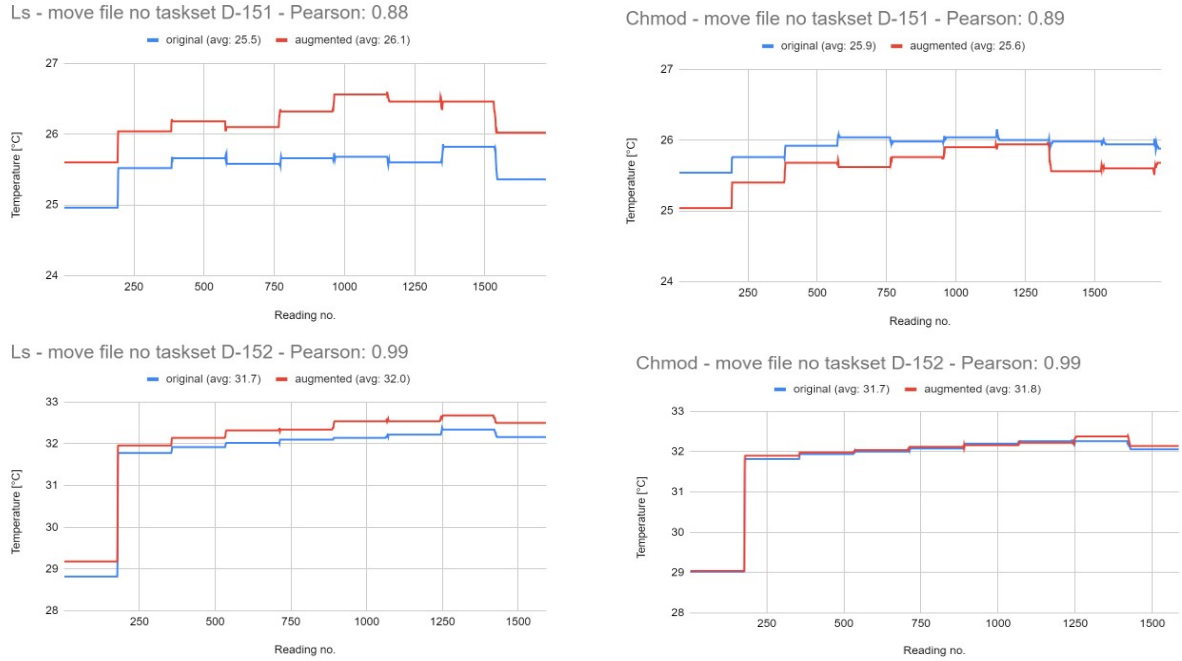

Figure A7. Results obtained with moving small file as a noise, without selecting CPU affinity.

Appendix A.2.3. Noise: Browser Surfing

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 A8.

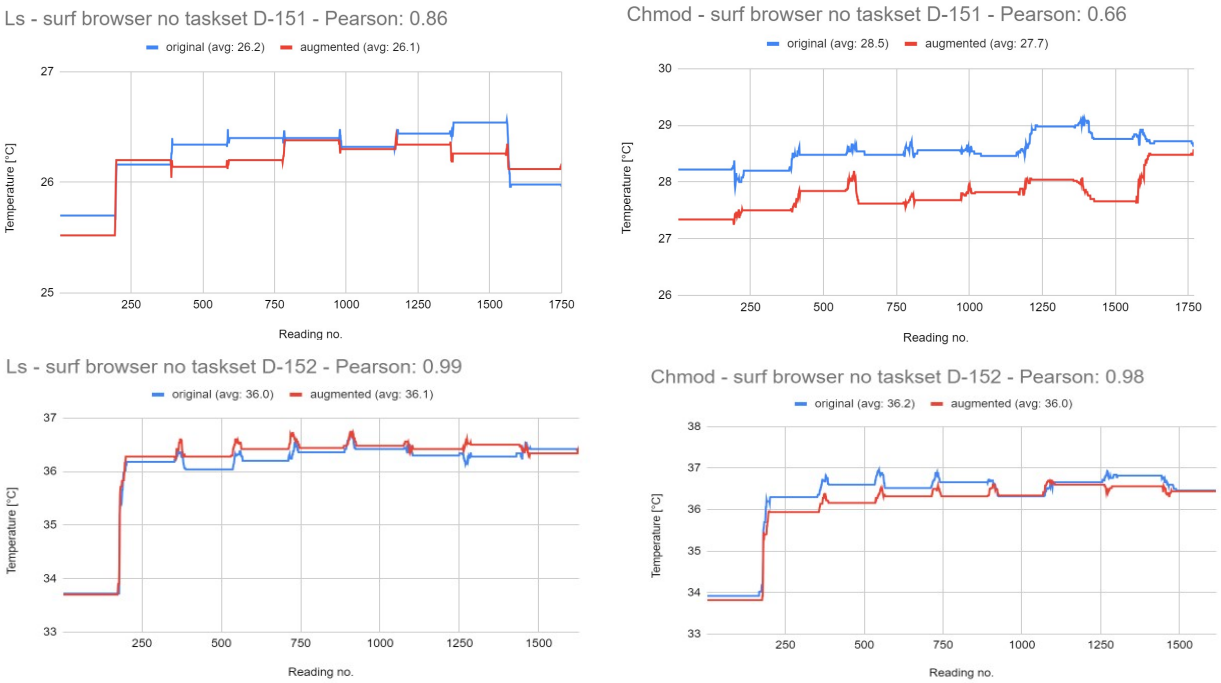

Figure A8. Results obtained with surfing the browser as a noise, without selecting CPU affinity.

Appendix A.2.4. Noise: Keystrokes

Better results for the key presses noise are in the case of *chmod*, which can be seen in Figure A9, where the Pearson correlation is 0.94 and the difference between the original and the augmented command is 0.7 °C.

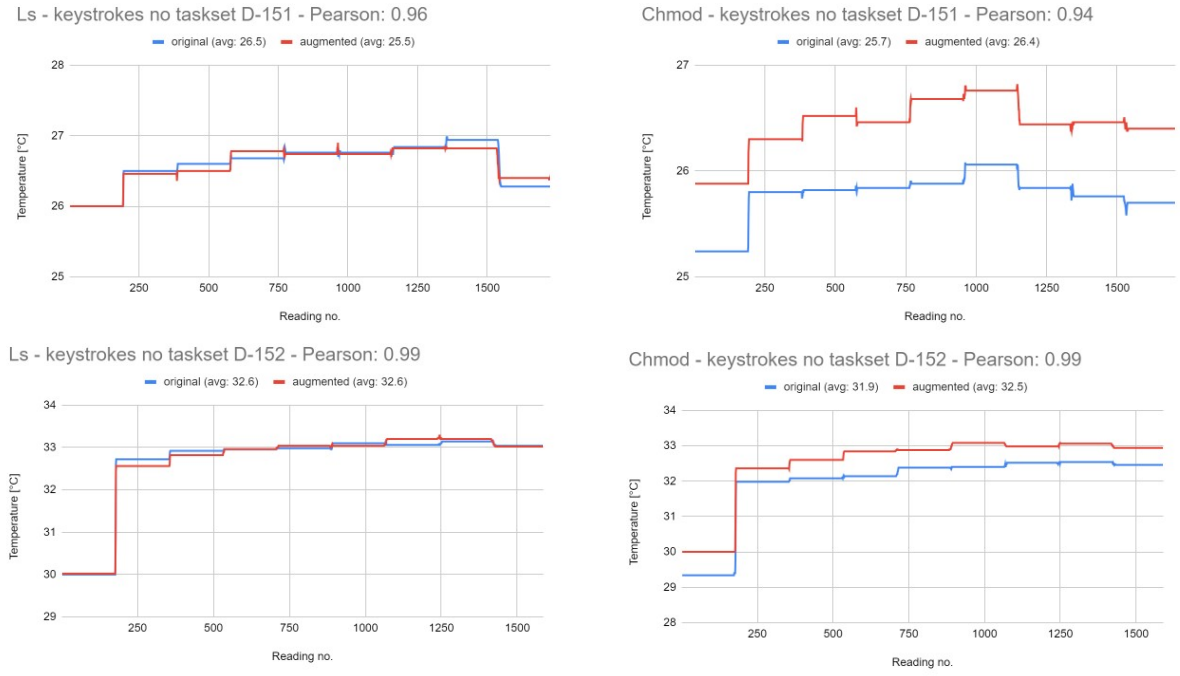

Figure A9. Results obtained with keystrokes as a noise, without selecting CPU affinity.

Appendix A.2.5. Noise: Mathematical Computations

Good results are also found for the mathematical computations on *D-151* 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 A10.

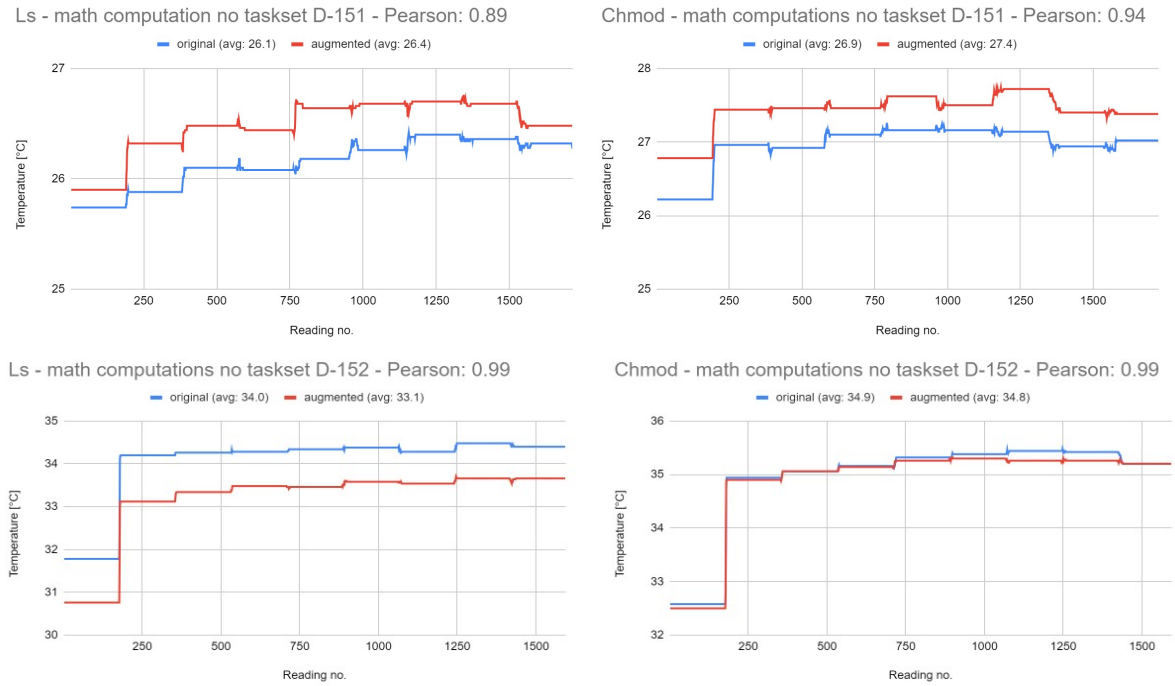

Figure A10. Results obtained with mathematical computations as a noise, without selecting CPU affinity.

References

1. Kim, T.; Shin, Y. ThermalBleed: A practical thermal side-channel attack. *IEEE Access* **2022**, *10*, 25718–25731.
2. Vasilas, T.; Jakobsche, T.; Ciorba, F.M. Hot-n-Cold: Mapping the Syscall Attack Surface Using Thermal Side Channels. In Proceedings of the 2023 22nd International Symposium on Parallel and Distributed Computing (ISPDC), Bucharest, Romania, 10–12 July 2023; pp. 93–100.
3. Marek, R. Kernel Driver Coretemp. Available online: https://docs.kernel.org/hwmon/coretemp.html (accessed on 29 May 2024).
4. Common Vulnerabilities and Exposures. Available online: https://cve.mitre.org/cve/search_cve_list.html (accessed on 29 May 2024).
5. Masti, R.J.; Rai, D.; Ranganathan, A.; Müller, C.; Thiele, L.; Capkun, S. Thermal covert channels on multi-core platforms. In Proceedings of the 24th USENIX Security Symposium (USENIX Security 15), Washington, DC, USA, 12–14 August 2015; pp. 865–880.
6. Lampson, B.W. A note on the confinement problem. *Commun. ACM* **1973**, *16*, 613–615.
7. Yarom, Y.; Ge, Q.; Liu, F.; Lee, R.B.; Heiser, G. Mapping the Intel last-level cache. *Cryptol. ePrint Arch.* **2015**. Available online: https://eprint.iacr.org/2015/905 (accessed on 1 August 2024).
8. Mushtaq, M.; Mukhtar, M.A.; Lapotre, V.; Bhatti, M.K.; Gogniat, G. Winter is here! A decade of cache-based side-channel attacks, detection & mitigation for RSA. *Inf. Syst.* **2020**, *92*, 101524.
9. Lee, R.B. *Security Basics for Computer Architects*; Springer Nature: Berlin, Germany, 2022.
10. Hutter, M.; Schmidt, J.M. The temperature side channel and heating fault attacks. In Proceedings of the Smart Card Research and Advanced Applications: 12th International Conference, CARDIS 2013, Berlin, Germany, 27–29 November 2013; Revised Selected Papers 12. Springer: Berlin/Heidelberg, Germany, 2014; pp. 219–235.
11. Claeys, T.; Rousseau, F.; Simunovic, B.; Tourancheau, B. Thermal covert channel in Bluetooth low energy networks. In Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks, Miami, FL, USA, 15–17 May 2019; pp. 267–276.
12. Aljuffri, A.; Zwalua, M.; Reinbrecht, C.R.W.; Hamdioui, S.; Taouil, M. Applying thermal side-channel attacks on asymmetric cryptography. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2021**, *29*, 1930–1942.
13. Bartolini, D.B.; Miedl, P.; Thiele, L. On the capacity of thermal covert channels in multicores. In Proceedings of the Eleventh European Conference on Computer Systems, London, UK, 18–21 April 2016; pp. 1–16.
14. Long, Z.; Wang, X.; Jiang, Y.; Cui, G.; Zhang, L.; Mak, T. Improving the efficiency of thermal covert channels in multi-/many-core systems. In Proceedings of the 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 19–23 March 2018; pp. 1459–1464.

15. Dey, S.; Singh, A.K.; McDonald-Maier, K. ThermalAttackNet: Are CNNs making it easy to perform temperature side-channel attack in mobile edge devices? *Future Internet* **2021**, *13*, 146. [CrossRef]
16. Taneja, H.; Kim, J.; Xu, J.J.; Van Schaik, S.; Genkin, D.; Yarom, Y. Hot Pixels: Frequency, Power, and Temperature Attacks on {GPUs} and Arm {SoCs}. In Proceedings of the 32nd USENIX Security Symposium (USENIX Security 23), Anaheim, CA, USA, 9–11 August 2023; pp. 6275–6292.
17. Zhan, D.; Yu, Z.; Yu, X.; Zhang, H.; Ye, L. Shrinking the kernel attack surface through static and dynamic syscall limitation. *IEEE Trans. Serv. Comput.* **2022**, *16*, 1431–1443.
18. Hung, H.W.; Liu, Y.; Sani, A.A. Sifter: Protecting security-critical kernel modules in Android through attack surface reduction. In Proceedings of the 28th Annual International Conference on Mobile Computing Furthermore, Networking, Sydney, NSW, Australia, 17–21 October 2022; pp. 623–635.
19. Xing, Y.; Cao, J.; Sun, K.; Yan, F.; Wan, S. The devil is in the detail: Generating system call whitelist for Linux seccomp. *Future Gener. Comput. Syst.* **2022**, *135*, 105–113.
20. Karn, R.R.; Kudva, P.; Huang, H.; Suneja, S.; Elfadel, I.M. Cryptomining detection in container clouds using system calls and explainable machine learning. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *32*, 674–691.
21. Gaidis, A.J.; Atlidakis, V.; Kemerlis, V.P. Sysxchg: Refining privilege with adaptive system call filters. In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, Copenhagen, Denmark, 26–30 November 2023; pp. 1964–1978.
22. Song, S.; Suneja, S.; Le, M.V.; Tak, B. On the value of sequence-based system call filtering for container security. In Proceedings of the 2023 IEEE 16th International Conference on Cloud Computing (CLOUD), Chicago, IL, USA, 2–8 July 2023; pp. 296–307.
23. Yang, S.; Kang, B.B.; Nam, J. Optimus: Association-based dynamic system call filtering for container attack surface reduction. *J. Cloud Comput.* **2024**, *13*, 71.
24. Peterson, J.L.; Silberschatz, A. *Operating System Concepts*; Addison-Wesley Longman Publishing Co., Inc.: Hoboken, NJ, USA, 1985.
25. Tanenbaum, A. *Modern Operating Systems*; Pearson Education, Inc.: London, UK, 2009.
26. Bovet, D.P.; Cesati, M. *Understanding the Linux Kernel: From I/O Ports to Process Management*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2005.
27. Love, R. *Linux System Programming: Talking Directly to the Kernel and C Library*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2007.
28. Bagherzadeh, M.; Kahani, N.; Bezemer, C.P.; Hassan, A.E.; Dingel, J.; Cordy, J.R. Analyzing a decade of Linux system calls. *Empir. Softw. Eng.* **2018**, *23*, 1519–1551.
29. 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. Available online: 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 (accessed on 4 June 2024).
30. Intel CPU Temperature Guide. Available online: https://forums.tomshardware.com/threads/intel-cpu-temperature-guide.1488337/ (accessed on 21 June 2024).
31. TechPowerUp—Intel Core i7-10700 Review. Available online: https://www.techpowerup.com/review/intel-core-i7-10700/3.html (accessed on 16 June 2024).
32. Marek, R. Kernel Driver Coretemp. Available online: https://www.kernel.org/doc/Documentation/hwmon/coretemp (accessed on 4 June 2024).
33. Clonezilla—The Free and Open Source Software for Disk Imaging and Cloning. Available online: https://clonezilla.org/ (accessed on 19 April 2024).
34. Coreutils—GitHub. Available online: https://github.com/coreutils/coreutils (accessed on 1 August 2024).
35. ffplay Documentation. Available online: https://ffmpeg.org/ffplay.html (accessed on 3 May 2024).
36. taskset(1)—Linux Manual Page. Available online: https://man7.org/linux/man-pages/man1/taskset.1.html (accessed on 2 February 2024).
37. Keith, G.; Calkins, A.U. Applied Statistics—Lesson 5. Correlation Coefficients. Available online: https://www.andrews.edu/~calkins/math/edrm611/edrm05.htm (accessed on 18 July 2024).
38. stress-ng—A Tool to Load and Stress a Computer System. Available online: https://manpages.ubuntu.com/manpages/jammy/man1/stress-ng.1.html (accessed on 30 November 2023).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.