Security Breachers: Hardware Performance Counter (HPC) and Malware

Kenneth Chen, Srushti Gaikaiwari, Daniel Horan, Alejandro Torres, Nicholas Tran
Department of Electrical and Computer Engineering, Texas A&M University
College Station, Texas
Email: kchen02@tamu.edu, daniel_horan@tamu.edu, srushtigaikaiwari@tamu.edu, atorr0523@tamu.edu, nicktran1102@tamu.edu

I. INTRODUCTION

Many modern processors today are being attacked by malicious software (malware) that infiltrates and causes damage to computer systems posing a threat to many processors.Various forms of malware include “viruses, worms, Trojan horses, rootkits, and ransomware,” and are designed to affect these machines in different ways [2]. Some other ways these malware would affect the system are by “damaging the targeted system, allowing remote code execution, and stealing confidential data” [1]. The malware that is used today “continues to be successful because it is just as varied as the systems and users they intend to compromise” [11]. Many professionals use countermeasures to defend against malware such as setting up firewalls to monitor network traffic, or antivirus software to detect and remove known malware. As the usage of the internet has increased, so has the presence of malicious software along with their next-generation attempts to infiltrate machines undetected. Currently, different malware detection approaches are used to identify the different types of malware.

To counteract these various malware, some modern malware detection approaches are used to detect them. There are a variety of malware detection approaches, each with its own specific approach to detecting this malware, some of which include signature-based detection, behavioral-, heuristic-, and model checking-based detection” [1]. In addition to these different malware detection approaches, there have been some newly proposed approaches some of which include “deep learning-, cloud-, mobile devices-, and IoT-based detection” [2]. However, as we use these malware detection approaches, their limitations in detecting malware are taken into account. Most approaches can detect newer generations of malware, but some can only detect the more well-known ones.

To combat these difficulties when using these malware detectors, Hardware Performance Counters can be used. Hardware Performance Counters (HPCs) are specialized registers or components within a computer’s central processing unit (CPU) or other hardware components that are designed to monitor and measure various aspects of a computer's performance [1]. HPCs are programmed to count specific events or occurrences, such as the number of instructions executed, cache misses, or branch mispredictions. HPCs have been used in modern-day processors and can provide many advantages when detecting malware. These counters can track both hardware-level events and software-level events. HPCs “monitor and measure events that occur at the CPU level and can identify and understand the behavior of code hotspots” [1]. Although they have their advantages in detecting malware, the “lack of determinism in performance counter values and the lack of probability of HPC events” are the main disadvantages when using an HPC for “different application domains” [1].

In this paper, we will further examine the basis of malware detection and understand the behaviors that go along with malware and benign-ware with the usage of HPCs. By writing a simple benign-ware and a simple malware, we can mimic their performances and gather data from these. Through the methods described and the information provided, we will be able to perform an analysis of the performances of both malware and benign-ware and compare each performance from the data collected. With this, we can come up with a conclusion of the similarities and differences of the behaviors that malware and benign-ware perform.

In this paper, we will detail how we created a simple malware and a simple benign-ware program. We will demonstrate how we used Hardware Performance Counters to identify several behaviors that differed between our simple malware and benign-ware, and how those results may apply to other malware and benign-ware as well.

II. BACKGROUND

Malicious software, or malware, consists of software that intentionally harms machines, like phones, computers, and data networks, by releasing payloads which are specific pieces of code that attack when infiltrating a computer system. They can cause harm to machines by stealing sensitive data, damaging security measures, and accessing unapproved systems to cause financial losses [2]. Malware originated with a traditional method of using one technique or process, like code mutation, to remain undetected. Next-generation malware employs multiple processes at the same time to be a continuing threat [2]. New and sophisticated countermeasures to enhance detection are needed to protect current and future machines against next-generation malware attacks. One possible solution could be hardware performance counters (HPCs) [1].

Studies assert that virus detection is NP-complete [3], with no efficient solution. New-generation malware uses many obfuscation techniques to further complicate the detection process, bypassing common kernel mode protection software. Oligomorphic malware uses different keys for encryption and decryption. Polymorphic does the same but also creates slightly modified copies of itself to increase detection difficulty. Metamorphic viruses don't use encryption but transform themselves into drastically different versions, making detection extremely difficult. The complexity of the problem and rapidly evolving malware make developing effective anti-virus software a monumental task.

To combat this rapidly evolving malware, HPCs can be used since they provide valuable insights into how a system is executing software, helping developers and system administrators optimize applications and improve overall system performance. Performance counters can be read one of two ways, either sampling or polling, but "they must be first configured according to the events of interest" [1]. For sampling, instead of counting every occurrence of an event, HPCs can be configured to sample events at regular intervals. Polling can be read instantly but must be configured properly to be written into model-specific registers (MSRs) [1]. "Although performance counters were initially introduced for debugging purposes, they have been used in a myriad of applications, such as vulnerability research and malware defense." [1].

HPCs offer a unique perspective on system behavior by providing low-level insights into a computer's hardware operations. Being hardware-based, they come with several advantages. One of the primary benefits is real-time monitoring, which can potentially detect zero-day exploits or new malware variants more swiftly than traditional signature-based antivirus solutions that depend on frequently updated databases. Furthermore, HPCs can monitor micro-architectural events. It's worth noting that malware often exhibits patterns detectable at this micro-architectural level, such as frequent cache misses due to code obfuscation techniques. Another advantage of HPCs is their potential to monitor system behavior with less overhead, leading to smoother system performance. Since they operate at a different level than most antivirus solutions, they might detect malware designed to evade traditional software-based detection mechanisms. As a result, methods like anomaly detection, signature-less detection, and contextual analysis can be implemented and yield promising results.

However, HPCs also come with a set of challenges. One of the primary concerns is their non-deterministic nature [4], which attackers can exploit to conceal malicious activities or generate false positives, making legitimate activities appear malicious. Overcounting [4] is another issue associated with HPCs. This can arise due to various factors, including hardware interrupts like PMIs (Performance Monitoring Interrupts). Attackers can exploit this by intentionally triggering events leading to overcounting, effectively masking their malicious activities. In essence, if an attacker is well-versed in the behavior of HPCs, they could potentially evade detection.

Hardware Performance Counters are essential tools for understanding and improving the performance of software and hardware systems. They offer detailed insights into system behavior, enabling developers and system administrators to optimize applications and enhance overall system performance. While HPCs present a promising avenue for malware detection, they are not devoid of challenges. The non-deterministic nature of HPCs, issues with overcounting, and the potential for attackers to manipulate HPC readings can pose significant challenges. Moreover, it's crucial to note that while HPCs can potentially detect malware behavior, they might not possess the capability to remove or quarantine the malware, a task that traditional antivirus solutions are specifically designed for.

III. BEHAVIOR ANALYSIS OF MALWARE AND BENIGN-WARE

A. Description of the system used

In conducting our behavioral analysis of benign-ware and malware, we established a secure testing environment on a MacBook Air 2015 [9]. This specific choice of hardware was motivated by our goal to ensure the highest accuracy in our research. Utilizing the MacBook Air allowed us to closely replicate real-world conditions, as many users operate on similar software and hardware configurations. This approach enhances the relevance and applicability of our findings, as it mirrors the environments where benign-ware and malware typically operate.

For the operating system, we selected Ubuntu 22.04.3 LTS to run on the MacBook Air [10]. This decision was based on several factors. The extensive use of Ubuntu in the technology sector provides a wealth of documentation and user experiences, which is invaluable for troubleshooting and refining our experimental setup. Its open-source nature offers an added layer of transparency, enabling us to conduct a thorough examination of any unusual interactions or anomalies observed during our tests. Additionally, Ubuntu's compatibility with a wide range of software tools, including the Linux 'perf' tool for performance analysis, further affirmed its suitability for our research objectives.

B. Description of Malware

In this project, we developed a Python-based ransomware prototype to study its operational mechanisms. The software utilized the cryptography.fernet library, which employs URL-safe encoding and a 128-bit AES encryption algorithm. Upon activation, this ransomware encrypted files within a specified directory, while intentionally excluding certain scripts to avoid self-encryption. The encryption key, generated dynamically, was stored separately in a '.key' file, essential for any subsequent decryption.

After the encryption phase, the ransomware displayed a notification to the user, indicating that their files were encrypted. This step is reflective of common ransomware attacks where victims are informed of the encryption and are typically given instructions for paying a ransom to regain access to their data. In our prototype, the decryption process depended on the user entering a correct passcode. Matching this passcode with a pre-set secret phrase in the script triggered the decryption, allowing the files to return to their original state. Incorrect passcode entry resulted in the files remaining encrypted.

For analytical clarity, we divided the ransomware into two scripts: the encryption script and the decryption script. The encryption script managed tasks such as scanning the directory, targeting files, generating the encryption key, and executing the encryption process. The decryption script, conversely, was responsible for key retrieval and file decryption, contingent upon the correct passcode input by the user. This division ensured streamlined functionality while embodying the typical characteristics of ransomware, which often includes a dual structure of encryption and conditional decryption.

C. Description of Benign-ware

In our research, we explored the behavior of benign software in embedded systems. Our objective was to gain insights into how benign software, or benign-ware, operates, particularly its interactions with system files and resources. Understanding the operation of benign-ware is important for distinguishing it from malicious software, and highlighting its role in digital environments.

The core of our study was the development of a File Backup Program, created using Python. This benign-ware is designed for standard file management tasks like scanning directories and reading files. It utilizes Python's *os* module for directory operations and *shutil* for file manipulation. The program works by identifying files in a directory, excluding script files, and then generating backup copies in a specified folder. It also includes a file restoration function, which allows the reversal of files to their original state.

A key aspect of our study was comparing the File Backup Program with a piece of ransomware also developed in Python. The comparison was based on the number of instructions each program executes. This approach was chosen because it allowed us to use Hardware Performance Counters (HPCs) to analyze their behavior. By comparing programs with a similar number of instructions, we were able to precisely assess how benign and malicious software differ in their interaction with system hardware. This comparison was instrumental in highlighting the distinct operational characteristics of benign and malicious software.

D. Hardware Performance Counters

In our research methodology, we harnessed the capabilities of HPCs by employing the Linux 'perf' tool, a performance monitoring and analysis utility. The 'perf' tool is known for its proficiency in collecting data about software performance, utilizing HPCs to offer insights into both CPU and memory behaviors. Our strategy involved profiling the system while executing both the ransomware and the benign backup software, aiming to distinguish between their unique microarchitectural traces. Notably, we paid particular attention to CPU usage spikes and context switches, as these metrics can often illuminate nuanced differences in software behavior. By collecting and interpreting these metrics, our team established a comprehensive understanding of how HPCs might serve as discerning tools in detecting malevolent activities within embedded systems.

IV. Inferences on the pros and cons of the approach used

When the code was run the data that was recorded is shown in the following tables below. Table 1 shows the data for the performance of malware and Table 2 shows the data for the performance of benign-ware.

	Encryption	Decryption
Instructions	327,971,639	327,399,851
Cycles	370,321,425	356,236,688
Instructions Per Cycle	0.894	0.92
CPU Spikes (Cycles)	70,552,590	70,473,257.2
Context Switches	4.2	1.4
Cache Misses L1	2,343,022	2,354,683
Cache Misses L3	163,114	164,005

Table 1: Average Malware Performance

	Backup	Recovery
Instructions	330,635,506	78,194,133
Cycles	378,708,948	103,735,896
Instructions Per Cycle	0.881	0.784
CPU Spikes (Cycles)	313,905,394	82,428,445
Context Switches	7	36
Cache Misses L1	6,761,024	1,831,242
Cache Misses L3	700,783	193,294

Table 2: Average Benign-ware Performance

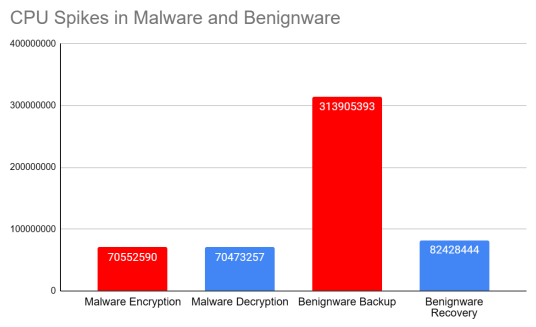

Figure 1: CPU Spike Cycle Performance

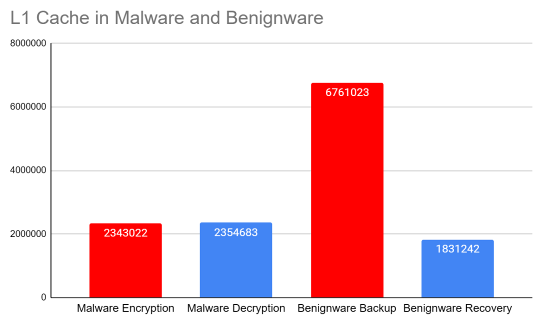

Figure 2: L1 Cache Miss Performance

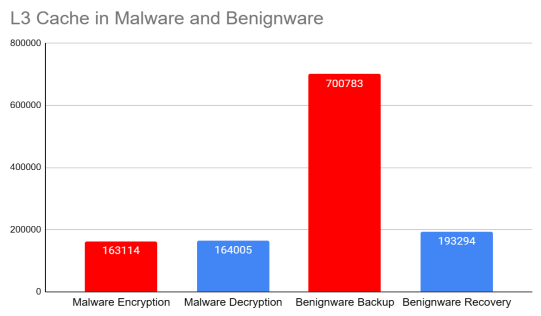

Figure 3: L3 Cache Miss Performance

We can see from the average performance of both programs that the benign-ware backup stage has significantly higher L1 Cache Misses, almost three times as many as malware's encryptions stage L1 Cache misses. We can also see that benign-ware has higher Cache Misses L3 and CPU spike cycles, both by about 4.5 times higher. These characteristics may be used to identify malware from benign-ware. Without Hardware Performance Counters providing this information, security software may allow malicious programs to compromise an unwitting user's machine.

V. Lessons Learned

In this project, we learned how to utilize virtual machines to create controlled testing environments. We learned to use Linux perf to monitor CPU performance and gained a more in-depth understanding of malware detection and Hardware Performance Counters. We also learned how to use the Fernet from the cryptography python library to create malware, and created simple malware and a simple benign-ware. Finally, we learned how to collaborate and delegate in the research and writing of this paper.

VI. Conclusion

In conclusion, throughout this paper, we examined the behavior of malware and benign-ware using HPCs. Through research into the basic principles of malware and HPCS, we outlined the flaws in detecting modern malware using previous methods. We inferred that using HPCs would be able to combat the new-generation malware practices through increased collection of data regarding system performance[1]. Our main goal was to analyze the differences and similarities between malware and benign-ware using HPCs.

We began by creating both a malware and a benign-ware program. We developed a dual-structure ransomware program in Python that encrypted files in a specified directory rendering them unusable as well as decrypting these files later. Our benign-ware program was a Python file backup program that generates backup copies of previously scanned and read files in a specific directory as well as restoring them if needed. Our study also focused on the comparison of the number of instructions performed by both the benign ware and ransomware programs to understand their interactions with the HPCs. Using HPCs, we could highlight the similarities and differences pertaining to the performance of the malware and benign-ware programs created. The HPC we utilized was the Linux 'perf' tool to monitor the performance of both software created and give insight into hardware discrepancies within the memory and CPU of the environment used.

A performance profile was created using the *perf* tool while executing both the encryption ransomware program and the file backup benign-ware program [6]. We collected the results of various metrics including several instructions performed, the number of cycles, CPU spikes, context switches, and memory misses in the L1 and L3 caches. In Table 1, the data outlined these metrics for the malware performance in the process of encryption and decryption. In Table 2, the data outlined the metrics for the benign-ware performance in the process of backup and recovery. We analyzed these tables to understand how HPCs can be important tools in the detection of malicious activity in embedded systems.

The significant comparisons of the average performance of the benign-ware and malware programs done by the HPCs are shown in Figures 1, 2, and 3. Here, the benign-ware backup stage was shown to have significantly higher L1 Cache Misses than the malware's encryptions stage L1 Cache misses. The benign-ware performance also had higher L3 Cache Misses and CPU spike cycles than the malware we created. On the other hand, the number of instructions performed and the other metrics did not seem to have significant differences between the malware and benign-ware software. We can conclude that the L1 and L3 cache misses along with the CPU spike cycles were important metrics identified by the HPCs that allowed us to differentiate between malware and benign-ware executions on the system. HPCs' ability to examine hardware performance metrics of benign ware and malware executions allowed us to detect potential malware attacks more readily.

REFERENCES

[1] S. Das, J. Werner, M. Antonakakis, M. Polychronakis and F. Monrose, "SoK: The Challenges, Pitfalls, and Perils of Using Hardware Performance Counters for Security," 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2019, pp. 20-38, doi: 10.1109/SP.2019.00021.

[2] Ö. A. Aslan and R. Samet, "A Comprehensive Review on Malware Detection Approaches," in IEEE Access, vol. 8, pp. 6249-6271, 2020, doi: 10.1109/ACCESS.2019.2963724.

[3] D. Spinellis, "Reliable identification of bounded-length viruses is NP-complete", IEEE Trans. Inf. Theory, vol. 49, no. 1, pp. 280-284, Jan. 2003.

[4]. V. M. Weaver, D. Terpstra and S. Moore, "Non-determinism and overcount on modern hardware performance counter implementations", IEEE International Symposium on Performance Analysis of Systems and Software, pp. 215-224, 2013.

[5] A. P. Namanya, A. Cullen, I. U. Awan and J. P. Disso, "The World of Malware: An Overview," 2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud), Barcelona, Spain, 2018, pp. 420-427.

[6] Dancuk, Milica. "Linux Perf: How to Use the Command and Profiler: Phoenixnap KB." Knowledge Base by phoenixNAP, 24 Aug. 2023, phoenixnap.com/kb/linux-perf#ftoc-heading-18.

[7] Gregg, Brendan. "Perf Examples." *Linux Perf Examples*, www.brendangregg.com/perf.html.

[8] "Tutorial." *Tutorial - Perf Wiki*, perf.wiki.kernel.org/index.php/Tutorial..

[9] "Apple." Official Apple Support, support.apple.com/kb/SP714?locale=ru_RU&amp;viewlocale=en_US.

[10] Canonical. *Ubuntu 22.04.3 Lts (Jammy Jellyfish)*, releases.ubuntu.com/jammy/.

[11] *Friend or Foe: Discerning Benign vs Malicious ... - NSF Public Access*, par.nsf.gov/servlets/purl/10295062. Accessed 6 Dec. 2023.