Why we need a theory of maliciousness: Hardware Performance Counters in security

Marcus Botacin[1,2] and André Grégio[2]

[1]Texas A&M University, USA
[2]Federal University of Paraná (UFPR), Brazil
{mfbotacin, gregio}@inf.ufpr.br

Abstract. Hardware Performance Counters (HPCs) are at the center of a research discussion: Is their use effective for malware detection? In this paper, we try to clarify the discussion by evaluating prior work presenting HPC criticism and highlighting their implicit assumptions and the potential research opportunities created by them. We discovered that HPCs are particularly good at detecting malware that exploits architectural side-effects, but not as good as traditional detection approaches at detecting pure-software malware, such that detection approaches must be combined. We also identified that most of the controversy about HPCs originates from researchers not clearly stating which type of malware they were considering. Thus, we claim the need for a theory of maliciousness to better state malware threats and evaluate proposed defenses.

Keywords: Performance Counter · Malware · Science of Security

1 Introduction

Modern processors [11] are equipped with special registers (counters) that automatically count the occurrence of some CPU events (e.g., cache hits, branch predictions) during code execution. Researchers have recently pointed HPCs as a promising way to detect malware [1, 5, 6], via the identification of (ab)normal execution profiles. However, no technique presents only benefits and some HPC researchers started to present criticism to HPCs for malware detection. The most significant one is that HPC metadata might not be informative about the execution content. An implicit assumption of most work in the HPC field is that a misbehavior at the software level corresponds to a misbehavior at the hardware level. Whereas this is a plausible hypothesis, it has been revealed not true in many cases. A significant work on HPC criticism (presented in [21] and extended in [22]) demonstrated the case of a ransomware sample that is malicious at the software level but indistinguishable at the hardware level, which led to the authors of these papers claiming HPCs' infeasibility for malware detection.

At the same time that the presented criticism emerged, new publications about HPC kept appearing, and even Intel decided to adopt a HPC-based approach in its security products [12], exactly to tackle the problem of ransomware. This controversy immediately leads to the question: Is there anyone right and/or wrong in this controversy? To answer this broad question, we (i) revisited the problem of detecting malware using HPCs by analyzing the findings of some representative

works published in the academic literature; and (ii) We investigated the root cause of the contradictory results. We discovered that the distinct works adopt different definitions of malware: some involving only software effects, and others involving hardware effects. We present experiments to demonstrate that HPCs are much more effective in identifying hardware than software side effects. Therefore, in our view, the controversy can be solved (or mitigated) if an integrated view is employed in the analyses.

Further, we understand that the differences on the understanding of what is malware and the effectiveness of the proposed detection solutions come from the lack of a widely accepted theory of maliciousness, which should clearly establish which effects are or not considered malicious and thus help judge which defenses are suitable for these cases. In other words, we believe that answering the question of whether malware is well-defined or not via a theory of maliciousness also answers the question of whether HPCs are suitable for malware detection or not by delimiting their scope and goals.

While this theory is not fully elaborated by the security community, we present ideas to help driving this development. More specifically, we propose the concept of attack space reduction, which involves the reduction of the possibilities of action of a given sample, a goal that is clearly accomplished by HPCs, since arbitrary malicious constructions designed to trigger architectural side-effects are blocked by HPC models that enforce standard executions in terms of metadata.

2 Results on HPC for Malware Detection: Is it effective?

What is reported about HPCs for malware detection? To understand whether the discussion about HPC applications for malware detection is justified or not, we surveyed the literature and found examples that demonstrate the origin of the controversy. Our goal in this work is not to present an exhaustive survey of all published work in the field (which is presented in [1, 22]), but to highlight the distinct conclusions about the same aspect. Therefore, we searched papers in the most popular research repositories for computer science (ACM, IEEE, and Springer) and screened the papers with the HPC keyword in the title and/or abstract that was cited at least once. We noticed that each paper represents its result using a different metric, and that they reached distinct conclusions. To illustrate the extreme cases (greatest variation): (i) The work of Botacin et al. [1] presented accuracy rates ranging from 36% to 97% and concluded that using HPCs for malware detection was feasible; (ii) Demme et al. [5] presented FPRs ranging from 35.7% to 83.1% and also concluded that HPCs for malware detection was feasible; (iii) whereas Zhou et al. [22] presented F1-scores ranging from 14.32% to 78.75% and concluded HPC for malware detection was unfeasible. Face that huge variation, a better evaluation of the reasoning made by the distinct authors is required to understand the discussion involving HPCs.

What is behind the conclusions? On the negative side, Zhou et al [22] clarify to us what motivates their uncertainty: "*The underlying assumption for previous HPC-based malware detectors are that malicious behavior affects measured HPC*

values differently than benign behavior. However, it is questionable, and in fact counter-intuitive, why the semantically high-level distinction between benign and malicious behavior would manifest itself in the micro-architectural events that are measured by HPCs." In fact, only a few studies so far investigated the side effects of malware execution at the architectural level, which makes this a reasonable argument until further research is developed. On the positive side, the authors that concluded that HPCs are effective also have a good argument. They indeed presented scenarios in which HPC-based detection was possible. If HPC-based detection is possible in many scenarios, nobody would care if one has already identified the correlation between detection and HPC values or not. It would be simply used. Thus, the discussion turns also into a matter of if the obtained results are robust enough for allowing conclusions about HPCs applicability.

Are these datasets enough? Once the discussion turned into an experimental robustness discussion, it is important to investigate how experiments were performed. Zhou et al [22] again explain their criticism of the studies with positive conclusions: "*the correlations and resulting detection capabilities reported by previous works frequently result from small sample sets and experimental setups that put the detection mechanism at an unrealistic advantage.*". This indeed demonstrates an experimental fragility. the problem with this criticism is that most of the criticizing studies also suffer from the same problem that they point out. Once again to show extreme cases, the works with a positive conclusion for HPC applications used only 4K [1] and 503 [5] samples, whereas the works with a negative conclusion for HPC application used only 2.3K [22] and 313 [4] samples. We notice that the considered datasets are all limited, especially in comparison to studies using other techniques for malware detection, such as typical ML classifiers. Thus, greater conclusions can only be taken if more studies are performed. Therefore, we conclude that the discussion about HPC is worth to be addressed and that further research is warranted. The next step is to understand how to contribute to this discussion.

3 A View on Debunking: Is Malware well defined?

Is a single counter-example enough? Zhou et al [22] developed a ransomware sample not detected by the HPC-based approach to claim its infeasibility. In their own words "*We also demonstrate the infeasibility in HPC-based malware detection with Notepad++ infused with a ransomware, which cannot be detected in our HPC-based malware detection system*". The authors are right in their feeling that some events cannot be differentiated and logically this single counter-example debunks the feasibility of HPCs. So, is the discussion finished? To answer this question, we must take a closer look and try to understand: Why did researchers care about proving it? Cohen's work [3] already proved in the '80s that a perfect malware detector does not exist. Isn't this work just an extension of this conclusion? It happens that security is by nature a practical subject and researchers and companies are always trying to find ways to detect malware in practice, in the average case, regardless of their limitations for specific cases. In this sense, if

we accept that single counter-examples discard entire techniques, such as HPCs, we should have also to discard signatures, since they have already been proven evadable [18], even though they are still used by AVs [20]. Similarly, we should have to stop using ML, since adversarial attacks have been demonstrated [15], even though the ML use has increased over time for ML drawbacks mitigation). Therefore, the discussion about HPC should not be whether mechanisms can be bypassed or not, but in which cases. The HPCs' use would make sense if it is good at detecting some type of malware or in some specific scenario.

When HPC are suitable? When HPC are suitable? In the argumentation against HPCs, Zhou el al [22] state that: "*we believe that there is no causation between low-level micro-architectural events and high-level software behavior.*". Whereas this argument makes sense for malware samples that act maliciously by invoking high-level APIs, this statement misses an entire class of attacks: those that act maliciously by exploiting architectural side-effects. We are currently aware of many attacks that exploit side effects and that can be detected via the architectural anomalies that they cause, such as (i) RowHammer [7], that can be detected via the excessive number of cache flushes [14]; (ii) ROP attacks [9], that can be detected due to the excessive number of instruction misses [19]; and (iii) DirtyCoW [8], that can be detected due to excessive paging activity.

The architectural nature of these attacks suggests that HPCs might be particularly good at detecting samples targeting it. To evaluate this hypothesis, we repeated previous work's strategies and developed two classifiers: The first is based on typical dynamic features used in ML detectors, such as tuples of invoked functions over time [10]; and the second using the performance counters supposed to detect the aforementioned events [1]. We performed all tests in a Linux environment, using the `perf` tool for HPC data collection, and considered the 10-folded evaluation of a set of 1K malware samples (we identified 50 to target the architecture) and 1K goodware collected from the system directories.

To create the test dataset, we relied on a collection of all 5K unique Linux malware samples available in the Virusshare (`virusshare.com`) and Malshare (`malshare.com`) repositories between 2012-2020, thus constituting a representative set of the existing Linux malware threats. At each test run, we randomly sampled a subset of this greater dataset to build a test dataset with an equal number of x86 samples for each year. The random sampling respects the family distribution observed in the original collection: 24% of Exploits, 22% of Virus, 20% of Backdoors, 10% of Rootkits, and 4% of Generic labels. All samples that cause side-effect were labeled as Exploits, according to the application of AVClass [17] over Virustotal (`virustotal.com`) labels.

This distribution between samples that cause and do not cause side-effects was selected to reflect the proportion of samples that we found in the wild during our research, limited by the current number of samples causing side-effects available in the online malware repositories. We fine-tuned hyper-parameters for all tested classifiers and we are reporting the results for the best combination (RandomForest classifier in both models).

Comparing the accuracy results for the tested classifiers (remember that the dataset is balanced), we notice that the traditional (non-HPC) ML approach outperforms the HPC one in the overall scenario (93.4% vs. 85.55%) , which is compatible with Zhou et al's. However, if we isolate the evaluation of the "ordinary" samples from those that cause side-effects, we notice that the whole detection capability of typical ML systems is due to the classification of "ordinary" samples (98% vs. 85%). The HPC approach significantly outperformed it when classifying the samples that cause side effects (6% vs 96%). The impact of these two datasets in the final result is proportional to their relative presence on the dataset. We expect architectural malware to become more popular over time.

4 Towards a definition of malware detection effectiveness

The need for better positioning. The presented experimental results show that one cannot simply claim HPC is not good for malware detection. While this result is somehow expected, why haven't other researchers framed the problem *as such*? In our view, the controversy originated because there is not a consensus in the research community as a whole about what is **objectively** considered malware and therefore how to handle it. We do not find consensus Even if we look to the NIST standards [16]. All definitions are broad and do not characterize samples regarding their form or precise target, even though they all state that malware is something undesired. Whereas most people, including researchers, might have developed a generic feeling about malware as something undesired and that the solution for it is to get rid of malicious files, the few attempts towards formally defining malware have been often revealed as problematic in the detection context due to the multiple corner-cases about the subject. The hardness in defining malware is made clear when we consider the "tricky" cases. Following are some of the multiple examples: (i) can software be considered malicious for some users and not for others?; (ii) can a performed action be considered malicious in the context of one software piece but legitimate for another?; and finally, (vi) in the specific context of this work, if the malware target is the system architecture and not another application or data, is it still malware? And how to handle this case? While these questions are not formally answered, researchers have been working with **operational definitions** of malware, which might suffice for most research work, but sometimes might lead to controversies as the one here presented.

When Zhou et al [22] criticize HPC, their implicit assumption (operational definition) is that malware is the "ordinary" samples employing high-level constructs, as it is clear in "*both ransomware and benignware use cryptographic APIs, but the ransomware maliciously encrypt user files, while the benignware safeguards user information.*", thus discarding the samples that intentionally cause side effects. However, it is hard to not consider these sample's activities as malicious, especially because both types of attacks might be associated: A sample might cause side-effects to escalate privileges and further cause more impacting harm via high-level actions. This highlights the need for better positioning attacks

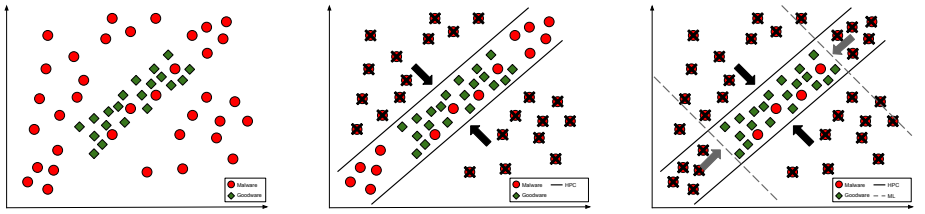
and defenses in a context. So, how to advance toward a definition? And, how to advance towards more defenses against whatever malware can be?

Towards better positioning. In our view, we need a theory of maliciousness to measure malware attacks and defenses. In science, concepts do not exist without a theory [2] (one possible view of science). Thus, one could not measure malware attacks and defenses without a theory of maliciousness–composed **not only** by the definition of what is considered malware, but **it also** brings multiple implicit and explicit consequences, such as how to measure the malware problem. Therefore, the point of this paper is not that we need only an extended taxonomy to include malware samples that cause side effects, but we need a theory of malware that explains why this type of sample is considered malicious and how we detect them (eventually using HPCs). Thus, a good theory of maliciousness should shape our understanding of the problem and provide tools/methods to let us know if we succeeded in controlling it.

Whereas we would like to be able to provide a complete theory of maliciousness, we acknowledge that this can only be achieved in the future by integrated community work. Meanwhile, we can give some hints about how it might look like using HPCs as a good example. We understand that a key contribution to a theory of maliciousness is the concept of attack space, i.e., the possibilities of actions that an attacker has over a resource to be protected. For HPCs, the attack space is defined in two dimensions: software and hardware, as shown in Figure 1. There, an application can be positioned somewhere in the plane defined by the software interactions it performs and the hardware effects it causes to perform these interactions. As defenders, we are interested in blocking both undesired software behaviors as well as their architectural roots and consequences.

Figure 1a illustrates the current scenario, in which we have an unbounded attack space, with malware and goodware samples mixed all over the space, as any malware implementation is allowed. Figure 1b illustrates what happens when HPCs are applied: the space is partially bounded in the performance direction, clearly positioning the (performance anomalous) samples out of the boundaries as malware. Though, there are still some remaining malware and goodware samples mixed in the (non-anomalous performance) bounded space, which explains why additional classifiers (such as typical ML ones) are still required for complete malware detection. When these are applied, the attack space is constrained in the software direction, as shown in Figure 1c. More formally, we can start hypothesizing the definition of the security provided by a solution as to how much it bounds the attack space, even though reducing it from a "greater" infinite. The key insight behind that is that it inverts the incentives. In an unbounded space (e.g., no performance restrictions), malware authors are free to place their samples anywhere, which requires defenders to counteract the attacks. In a more bounded space, such as the ones provided by HPCs, attackers are forced to conform their payloads to the behavior and/or form of the benign/desired applications, which is supposedly harder and more costly.

Attack Surface vs. Attack Space. The attack space concept resembles the attack surface concept [13]. Whereas they have similar goals, they have a signifi-



(a) Completely Unbounded Attack Space.

(b) HPC-bounded Attack Space.

(c) HPC and ML-bounded Attack Space.

Fig. 1: **The role of HPCs in security.** Reducing the attack space.

cant difference: the attack surface concept aims to limit the number of objects susceptible to being attacked, but it does not say anything about the nature of the possible attacks, which might be potentially infinite; the attack space concept does not say anything about the number of susceptible objects, but it limits the characteristics of the attacks to be performed against them. Thus, we understand them as complementary aspects to be evaluated in conjunction.

Making HPC practical. A way to evaluate if a technology contributes to making systems more secure is to verify if its addition to a set of existing techniques reduces the attack space. We intentionally refer to a set of techniques because it is naive to imagine that a standalone technology will reduce the attack space in all dimensions. In light of this definition and of the previous experiments, we believe that HPCs increase security. When HPCs are combined with typical ML detectors, the detection rate is increased to a value (97.9%) that is not reached by any solution individually for both sample datasets. Thus, HPCs should be seen as part of a pipeline of malware detectors that contribute to security by establishing borders in specific dimensions (e.g., architectural).

5 Conclusion

We revisited the problem of malware detection using HPCs to clarify the existing controversy about its feasibility. We discussed the current attempts to support and debunk HPCs and concluded that (i) although the discussion is justified, since research works present contradictory verdicts about the HPCs' feasibility for malware detection based on a wide range of metrics, (ii) we cannot discard HPCs as useful, since they present particularly good results for malware samples that cause side effects. We identified that most of the controversy comes from distinct interpretations of the malware concept, which sometimes considers only software effects and sometimes also includes hardware side effects. In summary, we recommend HPCs application as part of a pipeline of security solutions.

References

1. Botacin, M., Galante, L., Ceschin, F., Santos, P.C., Carro, L., de Geus, P., Grégio, A., Alves, M.A.Z.: The av says: Your hardware defini-

tions were updated! In: 2019 14th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC). pp. 27–34 (2019). https://doi.org/10.1109/ReCoSoC48741.2019.9034972
2. Chalmers, A.: What Is This Thing Called Science? University of Queensland Press (2013)
3. Cohen, F.: Computer viruses: Theory and experiments (1987)
4. Das, S., Werner, J., Antonakakis, M., Polychronakis, M., Monrose, F.: Sok: The challenges, pitfalls, and perils of using hardware performance counters for security. In: 2019 IEEE Sec. and Priv. (SP) (2019)
5. Demme, J., Maycock, M., Schmitz, J., Tang, A., Waksman, A., Sethumadhavan, S., Stolfo, S.: On the feasibility of online malware detection with performance counters. In: ISCA. ACM (2013)
6. DeRose, L.A.: The hardware performance monitor toolkit. In: Euro-Par 2001 Parallel Processing. Springer (2001)
7. ExploitDB: Rowhammer. `https://www.exploit-db.com/exploits/36310` (2015)
8. ExploitDB: Linux kernel 2.6.22 ¡ 3.9 - 'dirty cow' 'ptrace_pokedata' race condition privilege escalation (/etc/passwd method). `https://www.exploit-db.com/exploits/40839` (2016)
9. ExploitDB: Pms 0.42 - local stack-based overflow (rop). `https://www.exploit-db.com/exploits/44426` (2017)
10. Galante, L., Botacin, M., Grégio, A., de Geus, P.: Machine learning for malware detection: Beyond accuracy rates. In: Brazilian Security Symposium (SBSeg). SBC (2019)
11. Intel: Intel® 64 and IA-32 Architectures Software Developer's Manual. Intel (2013)
12. Intel: A new tool for cybersecurity - intel threat detection technology. `https://www.intel.com/content/www/us/en/architecture-and-technology/vpro/idc-security-report.html` (2021)
13. Kurmus, A., Sorniotti, A., Kapitza, R.: Attack surface reduction for commodity os kernels: Trimmed garden plants may attract less bugs. In: Proceedings of the Fourth European Workshop on System Security. EUROSEC '11, ACM (2011)
14. Li, C., Gaudiot, J.L.: Detecting malicious attacks exploiting hardware vulnerabilities using performance counters. In: IEEE COMPSAC (2019)
15. Martins, N., Cruz, J.M., Cruz, T., Henriques Abreu, P.: Adversarial machine learning applied to intrusion and malware scenarios: A systematic review. IEEE Access **8**, 35403–35419 (2020). https://doi.org/10.1109/ACCESS.2020.2974752
16. NIST: Glossary. `https://csrc.nist.gov/glossary/term/malware` (2021)
17. Sebastián, M., Rivera, R., Kotzias, P., Caballero, J.: Avclass: A tool for massive malware labeling. In: Monrose, F., Dacier, M., Blanc, G., Garcia-Alfaro, J. (eds.) Research in Attacks, Intrusions, and Defenses. Springer (2016)
18. Tasiopoulos, V.G., Katsikas, S.K.: Bypassing antivirus detection with encryption. In: 18th Panhellenic Conference on Informatics. p. 1–2. PCI '14, ACM (2014)
19. Wang, X., Backer, J.: Sigdrop: Signature-based rop detection using hardware performance counters (2016)
20. Wressnegger, C., Freeman, K., Yamaguchi, F., Rieck, K.: Automatically inferring malware signatures for anti-virus assisted attacks. In: AsiaCCS. ACM (2017)
21. Zhou, B., Gupta, A., Jahanshahi, R., Egele, M., Joshi, A.: Hardware performance counters can detect malware: Myth or fact? In: AsiaCCS. ACM (2018)
22. Zhou, B., Gupta, A., Jahanshahi, R., Egele, M., Joshi, A.: A cautionary tale about detecting malware using hardware performance counters and machine learning. `https://seclab.bu.edu/papers/perf_cnt_dtsi_2021.pdf` (2021)