(Extended Paper) 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? Whereas some studies suggest that it is, others suggest it is not. Although we do not have a definitive answer for this question, we do have a hint on how to contribute towards advancing the discussion. In our view, the aforementioned question is misleading. In practice, it is not a matter of whether HPCs are generally applicable or not, but a matter to what extent and for which types of malware they (and any solution) are. Therefore, 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 the best detection rates are achieved via a combination of detection approaches. 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

Countering Malware is a continuous problem for most systems, and there is no definitive solution for it, such that researchers are always trying to find new tools, approaches, and techniques to mitigate its impact. The proposed defensive solutions significantly varied over time, ranging from static byte signatures [54], passing through heuristics [7], until reaching the currently widespread Machine Learning (ML)-based models [48].

Recently, researchers started to propose the use of Hardware Performance Counters (HPCs) for malware detection [8, 18]. The usual approach involving HPCs consists of collecting system's metadata about the execution of a software and classifying the data as normal or not so as to infer the maliciousness of the software execution. The name HPCs comes from the fact that modern processors [4, 32] have special registers (counters) that automatically count the occurrence of some events, such as cache hits and misses, branch predictions and mispredictions, and so on, during code execution. Thus, one can collect this information without much overhead and apply classification algorithms to it.

It is hard to say which was the first work proposing the use of HPCs for malware detection, but research on the subject has been certainly impulsed by an ISCA paper that suggested its feasibility in multiple scenarios and architectures [18]. After that, multiple related approaches emerged to highlight the multiple benefits of HPC, such as not imposing significant overhead [19] and even the possibility of collecting data via reconfigurable hardware, making HPC-based approaches updatable via software [8].

However, no technique presents only benefits; approaches are always a matter of trade-off decisions, so researchers started to investigate what were the negative aspects of HPCs application and even question whether its usage was viable. A first criticism involved the non-deterministic nature of the collected data [16]. Since metadata depends on the architectural state on each execution, the classification result might be also biased by it. Whereas this criticism certainly points out the need for developing more robust classification procedures, it does not necessarily invalidate the HPC application.

Another HPC criticism involves the fact that the metadata might not be informative about the execution content. An implicit assumption of most work on 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 [55] and extended in [56]) demonstrated the case of a ransomware sample that is malicious at software level but indistinguishable at the hardware level, which led to the authors of these papers to claim 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 [33], exactly to tackle the problem of ransomware. This controversy immediately leads to the question: Who is right and/or wrong in this controversy? Or even, Is there anyone right and/or wrong in this controversy? These are questions that this work aims to help to answer.

To answer these broad questions, we had to adopt a two-step approach. First, we revisited the problem of detecting malware using HPCs by analyzing the findings of some representative works published in the academic literature. We discovered a discrepancy on the reported results that justify contradictory verdicts. This motivates our next step, which consisted in investigating the root of the contradictory results.

For such, we critically analyzed the published works criticizing HPCs. We discovered that the contradictory results originates in the fact that the concept of malware is implicitly interpreted distinctly by the studies. Therefore, the questions we formally address in this paper are twofold: (i) Can HPCs be used to detect malware? and (ii) Is malware sufficiently well defined?

In the current world we live, there are malware of all types: some are classically recognized as malware, operating from a software-focused perspective; others are emerging threats, being malicious by exploiting architectural side-effects, thus also hardware-focused, such that they are not always understood as malware in the usual sense. On the one hand, the studies claiming HPCs viable are implicitly assuming side-effect malware. On the other hand, the studies claiming HPCs unfeasibility implicitly assume software-focused malware operation.

In our view, the controversy can be solved (or mitigated) if an integrated view is employed in the analyses. We demonstrate it via an example of a scenario which mixes detectors based on typical ML and on HPCs. In it, the combination of both approaches produced a greater detection rate than individual detectors. We understand that the differences on the understanding of what is malware and on the effectiveness of the proposed detection solutions comes from the lack of a widely accepted theory of maliciousness, which should clearly establish which effects are or not considered malicious and thus help judging which defenses are suitable for these cases. In other words, we believe that answering the question 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 its 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 which 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.

Important to say that, when we critically analyze works debunking HPCs, our goal is not try to prove them wrong, even because we agree with most of them, but to show how they focused on a narrow scope and more research can be done from them. The current status of HPC should not be seem as disincentive for researchers but, the opposite, as multiple research opportunities.

We believe this work might help clarifying the scenarios in which HPCs should be considered. In particular, we believe HPCs should be part of a pipeline of detection solutions in which each one reduces the attack space in a distinct direction. The elaboration of this concept is presented over this paper.

In summary, our contributions are as follows:

- We revisit the problem of deciding the feasibility of leveraging HPCs for malware detection aiming to pinpoint its open questions and evaluating the source of the controversy.
- We present a scenario of architecture-aware malware in which HPCs are key for detection.
- We claim the need for establishing a theory of maliciousness to evaluate malware attacks and defenses in a proper context.

This work is organized as follows: In Section 2, we present examples of positive and negative results on HPCs to position the controversy in the context

of the discussion about their efficiency for malware detection; In Section 3, we critically analyze some works aiming to debunk HPCs feasibility and discuss their limitations under the light of the lack of a clear malware definition; In Section 4, we present our proposal to clarify the discussion via a new interpretation of the questions and results around the HPC controversy; In Section 5, we put our study in the context of the science of security discussion; In Section 6, we present related work to support our discussion; In Section 7, we draw our conclusions.
Vocabulary: In this paper, we refer to (i) HPC as short for HPC for malware detection, with no implication for HPCs' usage in other contexts; and (ii) side-effects as the impact of running code on the internal processor structures [42].

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

In this section, we briefly revisit some research works in the HPC for malware detection field and evaluate whether the discussion about them is justified.
What is reported about HPCs for malware detection? To understand whether the discussion about HPC application to malware detection is justified or not, we need first to understand its origin. For doing so, 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 [8, 56]), 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 were cited at least once.

Table 1: **HPC experiment's results.** Selected papers to highlight the distinct conclusions about its viability.

Work	Metric	Conclusion
Botacin et al. [8]	36% to 97% accuracy	Positive
Demme et al. [18]	35.7 to 83.1% AUC FPR	Positive
Zhou et al. [56]	14.32% to 78.75% F1 score	Negative

Table 1 presents the distinct metrics reported for some research works and the respective author's position towards the results (whether they are acceptable for malware detection or not). We opted to represent the papers with the greatest results variation among all screened papers (the extreme cases). Notice that the distinct papers represent their results in terms of different metrics.

We notice that a common characteristic of all research works is the huge variation among the metrics, that change according to the considered dataset, monitored event, or used ML classifier. Face to this variation, the authors made distinct conclusions, some accepting and others rejecting the possibility of leveraging HPCs for the task at hand. Therefore, a better evaluation of the reasoning made by the distinct authors is required to understand the discussion involving HPCs.

What is behind the conclusions? Given the distinct conclusions presented above, it is important to understand what led authors to make their decisions. On the negative side, Zhou et al [56] 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 architectural level (see Section 6), 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 (though it must still be evaluated in real-world settings [5]). If HPC-based detection is possible in many scenarios, nobody would care whether one has already identified the correlation between detection and HPC values or not[1]. It would be simply used. Therefore, the discussion turns also into a matter of whether the obtained results by both researchers groups are robust enough for allowing conclusions about their 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 [56] 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.

Table 2: **HPC experiment's dataset sizes.** Summary of the dataset used in the selected paper's experiments.

Work	**Samples (#)**	**Conclusion**
Botacin et al. [8]	4K samples in 2 disjoint datasets	Positive
Demme et al. [18]	503 malware and 210 benign	Positive
Zhou et al. [56]	1,000 malware and 1,300 benign	Negative
Das et al. [16]	313 malware	Negative

The problem with this criticism is that most of the criticizing studies also suffer from the same problem that they point out. Table 2 shows the dataset size used by the selected studies supporting or criticizing the use of HPCs (also the extreme cases). We notice that the considered datasets are all limited, especially in comparison to studies using other techniques for malware detection, such

[1] We agree it would be great to identify it.

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.

Why datasets are so limited? It is important to understand why the so-far considered datasets are so limited to understand what limits the research on HPC to evolve to stronger conclusions. We hypothesize that a significant reason is that HPC research often requires experiments to be performed on bare-metal machines since performance counters are not often exposed to virtual machines, the most used environment for malware research. The use of a bare-metal system naturally limits the experiments due to the reduced scalability in comparison to virtual machines. Some hypervisors, such as KVM [37], export a reduced performance counters set from host to guest, but the literature is not conclusive in how much the hypervisor events themselves do not affect the statistics, thus causing the uncertainty problem [16]. Therefore, it seems that a required step towards developing more robust experiments is to first develop a more robust testing platform. In the next sections, we discuss other required steps towards more robust HPC experiments.

3 A View on Debunking Attempts: Is Malware well defined?

In science, while something has not turned yet into an acceptable fact, it is common to conduct attempts to debunk the proposition. In this section, we revisit a notable HPC debunk attempt to highlight how, in our view, the discussion has been driven towards an unproductive direction.

Is a single counter-example enough? Zhou et al [56] 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 [15] 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 [51], even though they are still used by AVs [54]. Similarly, we should have to stop using ML, since adversarial attacks have been demonstrated [39], even though the ML use has increased over time (see Section 6 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. For what type of malware are HPCs good?

Which Are the Types of Malware? Malware samples are diverse and there are multiple ways to classify them. Typical classification methods assign samples to clusters based on their construction goals (e.g., Virus, Trojan, and so on) and/or behaviors [29] (e.g., Evaders, Downloaders, and so on). However, these classification schemes do not tell the whole malware story. Malware samples can also be classified according to their operational nature: samples focused on the software abstraction vs. samples focused on the hardware abstraction. The typical classification schemes only focus in the first case, giving no attention to the latter. In the first case, malware samples are constructed as typical software pieces that blindly rely on the hardware abstractions and that target other software abstractions (e.g., files, objects, pipes, and so on). Whereas the execution of these software pieces impacts the hardware (it changes the architectural state), it is not the attacker's goal, but just a side-effect. Therefore, the architectural impact caused by these malware samples is often minimal, because they present predictable execution events in terms of hardware/architectural pieces (e.g., cache accesses, branch predictions, and so on). In the second case, however, the malware samples are designed by looking beyond the hardware abstraction and to abuse it to cause effects on software, such as by exploring the hardware organization to read so-far protected data, flip memory bits, and so on. These caused side-effects are then used by the attackers to attack the software stack itself (e.g., hardware bit flips disabling software protections). This second category also covers software pieces that cause side-effects while exploring software components (e.g., buffer overflows), since the bad behavior caused by the attacker's control flow hijackinng lead to unpredictable behavior in terms of hardware/architectural states (e.g., large number of cache misses, branch mispredictions, and so on). Due to this difference on the nature of the malware execution, it is plausible to hypothesize that different detection technologies can be used to detect them, including HPCs.

When HPC are suitable? In the argumentation against HPCs, Zhou el al [56] 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. Table 3 shows some known attacks that exploit the system's architecture for malicious intents, such as escalating privileges, and/or leaking private data. It also exemplifies how these attacks are available on the Web in standalone binaries, and how these attacks could theoretically be detected by architectural side-effects identification.

The nature of these attacks suggests that HPC might be particularly good at detecting samples leveraging them. To evaluate this hypothesis, we repeated previous work's strategies and developed two classification models: The first based on typical dynamic features used in ML detectors, such as tuples of invoked functions over time [27]; and the second using the performance counters supposed

Table 3: **Attacks with Architectural Side-effects.** The columns show, respectively, the popular attack name, examples of their availability, and potential ways to detect them via HPC events.

Attack	**Example**	**Reason**
RowHammer	[20, 21]	Excessive cache flushing [38]
ROP attacks	[24, 25]	Excessive instruction misses [53]
DirtyCoW	[22, 23]	Excessive Paging

to detect the aforementioned events [8]. All tests were executed 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 (of which 50 we identified to exploit architectural issues) and 1K goodware collected from system directories.

To create the test dataset, we relied on a collection of all 5K unique Linux malware samples available in the Virusshare[2], and Malshare[3] 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 [47] over Virustotal[4] 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).

Table 4: **HPC and ML detectors.** Solutions perform distinctly according to the scenario. The columns represent, respectively, the used approach, the result when all samples are combined, when only traditional samples are considered, and when only samples that cause side-effects are considered.

	All Samples	**All but side-effects**	**Side-effects only**
Typical ML	93,40%	98,00%	6,00%
HPC	85,55%	85,00%	96,00%
Combined	97.9%	98,00%	96,00%

Table 4 shows the accuracy results for the tested classifiers (remind that the dataset is balanced). It shows that the traditional (non-HPC) ML approach

[2] `virusshare.com`
[3] `malshare.com`
[4] `virustotal.com`

outperforms the HPC one in the overall scenario, which is compatible with Zhou et al's reasoning. 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. The HPC approach significantly outperformed it when classifying the samples that cause side effects. The impact of these two datasets in the final result is proportional to their relative presence on the dataset. It is supposed that if malware that causes side-effect becomes more popular over time, as it seems to be with the recent `SPECTRE` and `MELTDOWN`-class attacks–as they start to be observed in the wild [52], the importance of an HPC-based detector will be highlighted.

In our view, the previously-presented experiment is already clear in demonstrating HPCs' advantages in comparison to traditional, binary-based features, since the only difference between the two experimental settings is the data collection approach (binary-based vs. HPCs). However, we decided to perform a second experiment to demonstrate that results are not biased by the small fraction of samples causing architectural side-effects (i.e., no fitting problems). In the new experiment, we trained and tested the same models previously presented with respectively 500 samples that cause and do not cause architectural side-effects (the same number of 1K as in the previous experiment). Since we had only 50 samples that originally caused architectural side-effects, we adopted data augmentation strategies [6] to increase our sampling to this more representative dataset size. In other words, we created malware variants by adding and removing information from the binaries. We adapted the adversarial machine learning strategies described in [17] in this step, such as adding new sections, headers, dead-code, and appending data to the binaries to increase their diversity, and packing the samples with multiple packers to reduce their exposed features. The final result revealed similar characteristics as the previously presented experiment. For the dataset with samples that cause no side-effects, the typical ML model surpassed the HPC model in 11% (accuracy), whereas the HPC model performed 17% better (accuracy) than the ML model when considering side-effects samples. Therefore, we conclude that HPCs might still have a significant role in malware detection procedures. The next step is to discuss how to streamline it.

4 Towards a definition of malware detection effectiveness

In this section, we try to move the discussion forward by presenting a new interpretation of malware attacks and their associated defenses.

The need for better positioning. We previously showed experimental results that suggest that HPC detectors are good for some types of malware but not for others. Thus, they cannot be simply claimed as not good for malware detection. While some might claim that this result is somehow expected after we presented it, why haven't other researchers recognized the problem *as such*? In other words, where does this controversy come from? In our view, it 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. Table 5 shows the multiple definitions found on NIST-indexed documents [43]. 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. More than that, we observe that the definitions are not coherent among the multiple standards defining them. Definitions tend also to not be coherent among research papers, as a previous study showed that the field tends to adopt ad-hoc definitions [10].

Table 5: **Malware Definitions.** Variety observed on NIST-indexed publications [43].

"Hardware, firmware, or software that is intentionally included or inserted in a system for a harmful purpose."
"Software or firmware intended to perform an unauthorized process that will have adverse impact on the confidentiality, integrity, or availability of an information system."
"A program that is inserted into a system, usually covertly, with the intent of compromising the confidentiality, integrity, or availability of the victim's data, applications, or operating system or of otherwise annoying or disrupting the victim."
"Software designed and operated by an adversary to violate the security of a computer (includes spyware, virus programs, root kits, and Trojan horses)."
"A program that is written intentionally to carry out annoying or harmful actions, which includes Trojan horses, viruses, and worms."
"A computer program that is covertly placed onto a computer with the intent to compromise the privacy, accuracy, or reliability of the computer's data, applications, or operating system."

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 revealed often 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 some of multiple examples: (i) can a software be considered malicious for some user and not for another?; (ii) can a performed action be considered malicious in the context of one software piece but legitimate for another?; (iii) can (corrupted/abused) data be considered malicious or the concept only applies for the complete software?; (iv) is a software malicious if it only causes harm/conflicts when operating along with another application?; (v) is a software piece with a known exploitable bug malicious due to the possibility of acting maliciously? 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 [56] present their criticism about HPC, their implicit assumption (operational definition) is that malware is the "ordinary" samples employing high-level constructs, as it is made clear in excerpts that mentions high-level constructions, such as "*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 (and the impact of the effects themselves). 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 [46]. This highlights the need for better positioning attacks and defenses in a context. This discussion might advance with someone trying to distinguish "malware" from "exploits", using some characterization criteria (e.g., phases of a kill chain model [31]), but this discussion can easily return to the initial point, as it only characterizes the threat temporally and not based on their execution nature. This, in our view, only highlights more the need for establishing clear concepts when evaluating attacks and defenses. So, how to advance towards a definition? And, most importantly, how to advance towards more defenses against whatever malware can be?

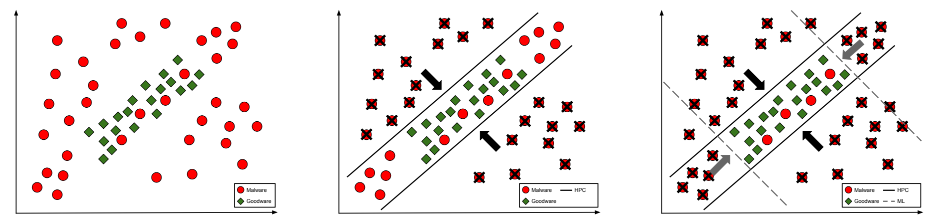
Towards better positioning. Once we identified that the lack of proper definitions is a problem, we need to start thinking about how to define things. In our view, *"if security is to be taken more seriously as a science"* (paraphrasing a paper criticizing HPCs [16]), we need a theory of maliciousness to measure malware attacks and defenses. In science, concepts do not exist without a theory [13] (one possible view of science). In a didactic analogy with physics, the concept of mass only exists in a Newtonian theory, and not in a quantum one, where you only have the energy concept for measuring things. Thus, one could not measure mass without Newton's concepts as reference. Similarly, one cannot precisely measure malware attacks and defenses without a theory of maliciousness.

More than that, it is important to highlight that a theory of malware is 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).

Whenever one proposes a scientific theory, he/she necessarily embeds in it his/her own views and judgments about it. Back to our physics analogy, when Newton proposes gravity to explain why the apple falls, he also poses questions about whether falling is a "problem" for our lives or not, how to measure the fall rate, and what if we could control the fall. Similarly, when one defines malware, he/she implicitly states it as a problem (otherwise would not be defining it), as undesired, and poses questions on how to measure and control it. Therefore, a good theory of maliciousness should shape our understanding about the problem and provide tools/methods to let us know if we succeeded on controlling it.

As authors and researchers, we would like to provide a complete theory of maliciousness, but, unfortunately, we are not able to by this time. In fact, this can only be achieved by an integrated community work. Meanwhile, we can give some hints about how it might look like and we believe HPCs work as a good example. We understand that a key contribution for 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. In the case of HPC, the attack space is defined in two dimensions: software and hardware, as shown in Figure 1. In this example, an applicationn 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 its architectural roots and consequences.

Figure 1a illustrates the current scenario, in which we have an unbounded attack space due to the lack of stronger definitions, with malware and goodware samples mixed all over the space, as any malware implemention 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. Even 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 a distinct direction (software-wise), as shown in Figure 1c. In other words, whereas *"not all attacks are anomalous"* [28], HPCs might help detecting the ones that are.



(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.

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 to a "smaller" infinite space (assuming that the spaces are computationally tractable [49]). 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. Some might notice that the attack space concept resembles the attack surface concept [36]. Whereas they have similar goals, they have a significant difference. The attack surface concept aims to limit the number of objects susceptible to be attacked, but it does not say anything about the nature of the possible attacks, which might be potentially infinite. In turn, the attack space concept does not say anything about the number of objects susceptible to be attacked, but it limits the possibilities of the attacks to be performed against them. Therefore, we envision them as complementary aspects that should be evaluated in conjunction.

Making HPC practical. The above definition is interesting but abstract. What does it mean in practice? It means that a way to evaluate whether a technology contributes to making systems more secure is to verify whether its addition to a set of existing techniques reduces the attack space or not. 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 the light of this definition, we believe that HPCs increase security in the sense presented in Table 4: When HPCs are combined with typical ML detectors, the detection rate is increased to values that are not reached by any solution individually. Thus, HPCs should be seen as part of a pipeline of malware detectors that contribute to security by establishing borders in specific dimensions (the samples that cause side effects).

Future Research Directions. We expect that our contribution might have helped better positioning HPC research in the "big picture" of malware detection technologies. However, this does not mean that the discussion is exhausted. HPCs warrant more studies to bridge the gaps on the definition of the limits of their attack space reduction capabilities. Therefore, we would like to finish pointing out a possible direction for future developments. Zhou et al [56] observed that: "*Modern processors can capture more than 100 micro-architectural events, but a design-imposed strict limit of 4 (on Intel) and 6 (on AMD) counter registers dictates that HPCs can only monitor a small subset of these events at one time.*". Future HPC research warrants investigating whether this limitation translates into an implementation barrier for malware detectors or if even the hundred events were available at the same time (which can be observed via simulations, as done in other scenarios [41]) HPCs would still be unable to reduce the attack surface more significantly than already described in the literature.

5 HPCs and the Science of Security

After we have presented all our concerns and insights about the use of HPCs for malware detection, it is key to face the question: if the required theory of maliciousness is still not complete and the if HPCs cannot be fully integrated or discarded until such theory is developed, why should one care about the proposed discussion? The answer for that is very straightforward: because reflecting about these aspects helps us to advance security as science. Thus, we following position the controversy in terms of the science of security debate.

"There is much discussion on whether security is a science" [1]. Whereas some work claim it is far from being [30], some claim that it might be [10], and some claim that it indeed is [50]. This uncertainty by itself shows that the subject warrants more discussion. Even those who agree that security is/can be science point out many current limitations. Table 6 shows the distinct challenges pointed in [50] towards making security more scientific.

Table 6: **Science of Security.** Challenges acknowledged by the community (pointed by [50]).

"Information security will not be a science until we all agreeon a common language or ontology of terms"
"we do not even have the fundamental concepts, principles,mathematical constructs, or tools to reliably predict or even measure cyber-security"
"Science is taken as a way to systematize knowledge at the appropriate level of generality that it can both be shared and remain useful"
"the most important attributes would be the construction of a common language and a set of basic concepts about which the security community can develop a shared understanding"
"Security science should give us an understanding of the limits of what is possible in some security domain, by providing objective and qualitative or quantifiable descriptions of security properties and behaviors"
"Consider the criticism of failure to seek refutation rather than confirmation. What do we refute in security? ...Such definitions can be argued for or justified, but are always contextual and not something we usually talk about as refuting or confirming."

We believe that all these aspects are present in the HPC discussion. The **terminology** aspect is clearly present in the distinct definitions of malware adopted by the researchers, sometimes including malware that cause side-effects and sometimes excluding them. This aspect is strongly tied to the **information sharing** aspect, since the distinct studies can only be compared if they are used an unified definition of the studied artifacts. Once malware is properly defined, one can start focusing on the **measurement** problem, as we would know what to consider in experiments and how to measure the expected effects. This is not currently as clear as is desirable, such that we introduced the new attack space concept as a way to help towards this direction. The experiments should clearly **define the limits** on the application of the proposed approach/technology. Whereas the studies presented in this paper have this goal in mind, they present their results in an unstructured way, such that they demonstrated weaknesses but did not draw clear limitation frontiers. In this sense, the **refutation** aspect is also present. We particularly discussed the debunking attempts due to the observed need for understanding what is being refuted, if the entire approach, or if it is a matter of drawing limits (our position).

Previous studies have claimed that the security field need to move from *"study cases to experiments"* [12]. We understand that this same need is present in the HPC case, as we need to identify not only specific weaknesses, but also to draw a landscape of their usefulness. More specifically, we need to put them in a greater context. Previous research have already identified the difficulty of putting things in context in many works in the literature [3]. It is mentioned that *"Research Objectives tend to be difficult to locate, which could hinder the ability of other researchers to put the findings in the proper context."*, which turns out to be exactly the difficulty we presented over this paper to compare the distinct works.

Considering the above scenario, we expect that the analyses presented in this work might help clarifying the controversy on HPC usage, give hints on how to advance the discussion on how to measure and evaluate defenses, and, in the last instance, contribute to the science of security field with an example that motivates and highlights the need for more scientific foundations to solve practical problems.

Beyond HPCs. We selected HPC as an illustrative example of current controversy in the academic literature. However, the phenomenon here discussed is not unique to the HPC technology, but it is common to any malware detection technique. Thus, the same might have happened in the past and might happen again in the future. Therefore, it is key to have a structured approach to evaluate the viability of newly proposed malware detectors. In this sense, we believe that approaching the problem via the science of security perspective as discussed is the best strategy.

6 Related Work

In this section, we revisit and discuss previous work to better position the discussion presented in this paper.

Defining Malware is a hard task and one can only find a few works in the literature trying to formalize the concept [35]. In most cases, malware is only defined in terms of its exhibited behaviors [29, 40] and not based in some hypothesized internal "essence". Therefore, it is plausible to argue that malware can also be seem through the lens of the caused side-effects in addition to OS events, which puts HPCs at a promising position for the development of malware detectors.

Non-standard malware is out there. In addition to typical EXE files, it is not rare to find malware samples written in Java, Javascript, and other languages [9]. The more diverse the malware samples, the greater the attacker's chance in evading "one-size-fits-all" malware detectors. To handle the multitude of malware samples, detectors have been diversified, now ranging from web detectors, to network monitors. The same way these detectors are not evaluated using all the same criteria, we believe that HPCs should also be evaluated in their proper context, and not for a generic malware detection task.

Other hardware tracing mechanisms than HPCs exists in modern CPUs. Intel's CPUs [32], for instance, present the Branch Trace Store (BTS) and Processor Tracer (PT) extensions that allow collecting metadata from instruction execution, including taken branches. The academic literature presents multiple research works based on both BTS [2] and PT [45], but none of these works led to the controversy involving HPCs. We believe that a plausible explanation for that is the deterministic nature of these technologies in comparison to HPCs. If this is really the case, more robust data handling procedures [34] should be developed to make HPC-based solutions more practical.
The relations between hardware and software events certainly need to be better clarified, but it is unfair to say that they are completely unknown. Previous work have demonstrated, for instance, the impact of malware packing on CPU's internal structures (e.g., caches, pipelines) due to the execution of self-modifying-code constructions [11]. In the specific case of HPCs, their relation with side-channel attacks and defenses has been already well-established [14].
The controversies on machine learning resemble the scenario we have drawn for HPCs. Whereas ML-based malware detectors become very popular [48], attacks to them have been demonstrated very effective [39]. Despite that, ML was not discarded as a valuable technique, but more robust models have been proposed to mitigate some of the evasion possibilities [26, 44]. We believe that a similar path should be taken for HPC-based approaches.

7 Conclusion

In this work, we revisited the problem of malware detection using Hardware Performance Counters (HPCs) so as to clarify the existing controversy about its feasibility. We discussed the current attempts to support and/or debunk HPCs via two research questions: (i) Can HPCs be used to detect malware? and (ii) Is malware sufficiently well defined? We concluded that:

- The existing controversy is justified, since research works presents contradictory verdicts about the HPCs feasibility for malware detection based on a wide-range of metrics.
- The experimental setup to conduct HPC experiments must be improved so as to provide more conclusive results, as current studies are limited in dataset size and variety, which also limits conclusions.
- The controversy comes from distinct interpretations about the malware concept, which sometimes consider only software effects and sometimes also include hardware side-effects.

Based on these conclusions, we recommend the following set of actionable items to be taken as goals to be pursued by the research community towards clarifying the discussion:

- The development of platforms that allow scaling HPC experiments beyond the limits of current bare-metal systems.

- The realization of experiments with greater datasets, in size and variety, to allow broader and stronger conclusions.
- The development of more robust methods to handle uncertainty and variability effects in HPC measurements.
- The development of a theory of maliciousness to better define which type of malware is evaluated and which criteria defines the success of a detector.

As a suggestion to move forward, we proposed:

- The concept of attack space reduction to complement the attack surface reduction by reducing the actions an attacker can perform over a surface in addition to eliminating exposed surfaces.

References

1. ACM: People of acm - leigh metcalf. `https://www.acm.org/articles/people-of-acm/2020/leigh-metcalf` (2020)
2. Aktas, E., Ghose, K.: Run-time control flow authentication: An assessment on contemporary x86 platforms. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing. p. 1859–1866. SAC '13, Association for Computing Machinery, New York, NY, USA (2013). https://doi.org/10.1145/2480362.2480708, `https://doi.org/10.1145/2480362.2480708`
3. Al-Zyoud, M., Williams, L., Carver, J.C.: Step one towards science of security. In: Proceedings of the 2017 Workshop on Automated Decision Making for Active Cyber Defense. p. 31–35. SafeConfig '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3140368.3140374, `https://doi.org/10.1145/3140368.3140374`
4. AMD: AMD64 Architecture Programmer's Manual Volume 2. AMD (2013)
5. Axelsson, S.: The base-rate fallacy and the difficulty of intrusion detection. ACM Trans. Inf. Syst. Secur. **3**(3), 186–205 (aug 2000). https://doi.org/10.1145/357830.357849, `https://doi.org/10.1145/357830.357849`
6. Bae, J., Lee, C.: Easy data augmentation for improved malware detection: A comparative study. In: 2021 IEEE International Conference on Big Data and Smart Computing (BigComp). pp. 214–218 (2021). https://doi.org/10.1109/BigComp51126.2021.00048
7. Bazrafshan, Z., Hashemi, H., Fard, S.M.H., Hamzeh, A.: A survey on heuristic malware detection techniques. In: The 5th Conference on Information and Knowledge Technology. pp. 113–120 (2013). https://doi.org/10.1109/IKT.2013.6620049
8. 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 definitions 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
9. Botacin, M., Aghakhani, H., Ortolani, S., Kruegel, C., Vigna, G., Oliveira, D., Geus, P.L.D., Grégio, A.: One size does not fit all: A longitudinal analysis of brazilian financial malware. ACM Trans. Priv. Secur. **24**(2) (2021). https://doi.org/10.1145/3429741, `https://doi.org/10.1145/3429741https://secret.inf.ufpr.br/papers/marcus_tops_br.pdf`

10. Botacin, M., Ceschin, F., Sun, R., Oliveira, D., Grégio, A.: Challenges and pitfalls in malware research. Computers & Security **106**, 102287 (2021). https://doi.org/https://doi.org/10.1016/j.cose.2021.102287, `https://www.sciencedirect.com/science/article/pii/S0167404821001115`
11. Botacin, M., Zanata, M., Grégio, A.: The self modifying code (smc)-aware processor (sap): a security look on architectural impact and support. Journal of Computer Virology and Hacking Techniques (2020). https://doi.org/10.1007/s11416-020-00348-w, `https://doi.org/10.1007/s11416-020-00348-whttps://secret.inf.ufpr.br/papers/SMC_marcus.pdf`
12. Carver, J.C., Burcham, M., Kocak, S.A., Bener, A., Felderer, M., Gander, M., King, J., Markkula, J., Oivo, M., Sauerwein, C., Williams, L.: Establishing a baseline for measuring advancement in the science of security: An analysis of the 2015 ieee security & privacy proceedings. In: Proceedings of the Symposium and Bootcamp on the Science of Security. p. 38–51. HotSos '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2898375.2898380, `https://doi.org/10.1145/2898375.2898380`
13. Chalmers, A.: What Is This Thing Called Science? University of Queensland Press (2013)
14. Chiappetta, M., Savas, E., Yilmaz, C.: Real time detection of cache-based side-channel attacks using hardware performance counters. Appl. Soft Comput. **49**(C), 1162–1174 (Dec 2016). https://doi.org/10.1016/j.asoc.2016.09.014, `https://doi.org/10.1016/j.asoc.2016.09.014`
15. Cohen, F.: Computer viruses: Theory and experiments (1987). https://doi.org/https://doi.org/10.1016/0167-4048(87)90122-2, `https://www.sciencedirect.com/science/article/pii/0167404887901222`
16. 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 Symposium on Security and Privacy (SP). pp. 20–38 (2019). https://doi.org/10.1109/SP.2019.00021
17. Demetrio, L., Biggio, B., Lagorio, G., Roli, F., Armando, A.: Functionality-preserving black-box optimization of adversarial windows malware. IEEE Transactions on Information Forensics and Security **16**, 3469–3478 (2021). https://doi.org/10.1109/TIFS.2021.3082330
18. 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)
19. DeRose, L.A.: The hardware performance monitor toolkit. In: Sakellariou, R., Gurd, J., Freeman, L., Keane, J. (eds.) Euro-Par 2001 Parallel Processing. pp. 122–132. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
20. ExploitDB: Rowhammer. `https://www.exploit-db.com/exploits/36310` (2015)
21. ExploitDB: Rowhammer. `https://www.exploit-db.com/exploits/36311` (2015)
22. 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)
23. ExploitDB: Linux kernel 2.6.22 ¡ 3.9 - 'dirty cow ptrace_pokedata' race condition (write access method). `https://www.exploit-db.com/exploits/40838` (2016)
24. ExploitDB: Crashmail 1.6 - stack-based buffer overflow (rop). `https://www.exploit-db.com/exploits/44331` (2017)
25. ExploitDB: Pms 0.42 - local stack-based overflow (rop). `https://www.exploit-db.com/exploits/44426` (2017)

26. Fleshman, W., Raff, E., Sylvester, J., Forsyth, S., McLean, M.: Non-negative networks against adversarial attacks (2019)
27. Galante, L., Botacin, M., Grégio, A., de Geus, P.: Machine learning for malware detection: Beyond accuracy rates. In: Anais Estendidos do XIX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais. pp. 47–56. SBC, Porto Alegre, RS, Brasil (2019). https://doi.org/10.5753/sbseg_estendido.2019.14005, https://sol.sbc.org.br/index.php/sbseg_estendido/article/view/14005
28. Gates, C., Taylor, C.: Challenging the anomaly detection paradigm: A provocative discussion. In: Proceedings of the 2006 Workshop on New Security Paradigms. p. 21–29. NSPW '06, Association for Computing Machinery, New York, NY, USA (2006). https://doi.org/10.1145/1278940.1278945, https://doi.org/10.1145/1278940.1278945
29. Grégio, A.R.A., Afonso, V.M., Filho, D.S.F., Geus, P.L.d., Jino, M.: Toward a Taxonomy of Malware Behaviors. The Computer Journal **58**(10), 2758–2777 (07 2015). https://doi.org/10.1093/comjnl/bxv047, https://doi.org/10.1093/comjnl/bxv047
30. Herley, C., Van Oorschot, P.: Sok: Science, security and the elusive goal of security as a scientific pursuit. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 99–120 (2017). https://doi.org/10.1109/SP.2017.38
31. Hutchins, E.M., Cloppert, M.J., Amin, R.M.: Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. In: White Paper (2010)
32. Intel: Intel® 64 and IA-32 Architectures Software Developer's Manual. Intel (2013)
33. 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)
34. Kazdagli, M., Reddi, V.J., Tiwari, M.: Quantifying and improving the efficiency of hardware-based mobile malware detectors. In: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). pp. 1–13 (2016). https://doi.org/10.1109/MICRO.2016.7783740
35. Kramer, S., Bradfield, J.C.: A general definition of malware. Journal in Computer Virology **6**(2), 105–114 (May 2010). https://doi.org/10.1007/s11416-009-0137-1, https://doi.org/10.1007/s11416-009-0137-1
36. 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, Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/1972551.1972557, https://doi.org/10.1145/1972551.1972557
37. KVM: Perf events. https://www.linux-kvm.org/page/Perf_events (2020)
38. Li, C., Gaudiot, J.L.: Detecting malicious attacks exploiting hardware vulnerabilities using performance counters. In: 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC). vol. 1, pp. 588–597 (2019). https://doi.org/10.1109/COMPSAC.2019.00090
39. 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
40. Meng, G., Xue, Y., Xu, Z., Liu, Y., Zhang, J., Narayanan, A.: Semantic modelling of android malware for effective malware comprehension, detection, and classification. In: Proceedings of the 25th International Symposium on Software Testing and Analysis. p. 306–317. ISSTA 2016, Association for Computing Machinery, New

York, NY, USA (2016). https://doi.org/10.1145/2931037.2931043, `https://doi.org/10.1145/2931037.2931043`
41. Mirbagher-Ajorpaz, S., Pokam, G., Mohammadian-Koruyeh, E., Garza, E., Abu-Ghazaleh, N., Jiménez, D.A.: Perspectron: Detecting invariant footprints of microarchitectural attacks with perceptron. In: 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). pp. 1124–1137 (2020). https://doi.org/10.1109/MICRO50266.2020.00093
42. Moseley, T., Vachharajani, N., Jalby, W.: Hardware performance monitoring for the rest of us: A position and survey. In: Altman, E., Shi, W. (eds.) Network and Parallel Computing. pp. 293–312. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
43. NIST: Glossary. `https://csrc.nist.gov/glossary/term/malware` (2021)
44. Íncer Romeo, I.n., Theodorides, M., Afroz, S., Wagner, D.: Adversarially robust malware detection using monotonic classification. In: Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics. p. 54–63. IWSPA '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3180445.3180449, `https://doi.org/10.1145/3180445.3180449`
45. Schumilo, S., Aschermann, C., Gawlik, R., Schinzel, S., Holz, T.: kafl: Hardware-assisted feedback fuzzing for OS kernels. In: 26th USENIX Security Symposium (USENIX Security 17). pp. 167–182. USENIX Association, Vancouver, BC (Aug 2017), `https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/schumilo`
46. Seaborn, M.: Exploiting the dram rowhammer bug to gain kernel privileges. `https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html` (2015)
47. 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. pp. 230–253. Springer International Publishing, Cham (2016)
48. Singh, J., Singh, J.: A survey on machine learning-based malware detection in executable files. Journal of Systems Architecture **112**, 101861 (2021). https://doi.org/https://doi.org/10.1016/j.sysarc.2020.101861, `https://www.sciencedirect.com/science/article/pii/S1383762120301442`
49. Spring, J.: An analysis of how many undiscovered vulnerabilities remain in information systems. `https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=875310` (2022)
50. Spring, J.M., Moore, T., Pym, D.: Practicing a science of security: A philosophy of science perspective. In: Proceedings of the 2017 New Security Paradigms Workshop. p. 1–18. NSPW 2017, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3171533.3171540, `https://doi.org/10.1145/3171533.3171540`
51. Tasiopoulos, V.G., Katsikas, S.K.: Bypassing antivirus detection with encryption. In: Proceedings of the 18th Panhellenic Conference on Informatics. p. 1–2. PCI '14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2645791.2645857, `https://doi.org/10.1145/2645791.2645857`
52. (jvoisin) Voisin, J.: Spectre exploits in the "wild". `https://dustri.org/b/spectre-exploits-in-the-wild.html` (2021)
53. Wang, X., Backer, J.: Sigdrop: Signature-based rop detection using hardware performance counters (2016)

54. Wressnegger, C., Freeman, K., Yamaguchi, F., Rieck, K.: Automatically inferring malware signatures for anti-virus assisted attacks. In: AsiaCCS. ACM (2017)
55. Zhou, B., Gupta, A., Jahanshahi, R., Egele, M., Joshi, A.: Hardware performance counters can detect malware: Myth or fact? In: AsiaCCS. ACM (2018)
56. 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)