Mining on Someone Else's Dime: Mitigating Covert Mining Operations in Clouds and Enterprises

Rashid Tahir[1(✉)], Muhammad Huzaifa[1], Anupam Das[2], Mohammad Ahmad[1], Carl Gunter[1], Fareed Zaffar[3], Matthew Caesar[1], and Nikita Borisov[1]

[1] University of Illinois Urbana-Champaign, Urbana, USA
{tahir2,huzaifa2,mahmad11,cgunter,caesar,nikita}@illinois.edu
[2] Carnegie Mellon University, Pittsburgh, USA
anupamd@cs.cmu.edu
[3] Lahore University of Management Sciences, Lahore, Pakistan
fareed.zaffar@lums.edu.pk

Abstract. Covert cryptocurrency mining operations are causing notable losses to both cloud providers and enterprises. Increased power consumption resulting from constant CPU and GPU usage from mining, inflated cooling and electricity costs, and wastage of resources that could otherwise benefit legitimate users are some of the factors that contribute to these incurred losses. Affected organizations currently have no way of detecting these covert, and at times illegal miners and often discover the abuse when attackers have already fled and the damage is done.

In this paper, we present *MineGuard*, a tool that can detect mining behavior in *real-time* across pools of mining VMs or processes, and prevent abuse despite an active adversary trying to bypass the defenses. Our system employs hardware-assisted profiling to create discernible signatures for various mining algorithms and can accurately detect these, with negligible overhead (<0.01%), for both CPU and GPU-based miners. We empirically demonstrate the uniqueness of mining behavior and show the effectiveness of our mitigation approach(≈99.7% detection rate). Furthermore, we characterize the noise introduced by virtualization and incorporate it into our detection mechanism making it highly robust. The design of MineGuard is both practical and usable and requires no modification to the core infrastructure of commercial clouds or enterprises.

Keywords: Cryptocurrency · Cloud abuse · Hardware Performance Counters

Electronic supplementary material The online version of this chapter (doi:10.1007/978-3-319-66332-6_13) contains supplementary material, which is available to authorized users.

M. Dacier et al. (Eds.): RAID 2017, LNCS 10453, pp. 287–310, 2017.
DOI: 10.1007/978-3-319-66332-6_13

1 Introduction

For most popular cryptocurrencies, such as Bitcoin and Litecoin, it is not profitable to mine using one's own resources unless the mining is carried out using specialized hardware [17]. However, the exercise can be of value if carried out on "stolen" resources, such as pools of hijacked VM instances or resources acquired under false pretexts (e.g., for research). This has incentivized both hackers [8,11,18] and unethical employees, such as professors [15], academic researchers and students mining on university-owned resources [10,26]. Even IT admins [7] have been found doing covert cryptomining. One researcher, for instance, misused NSF-funded supercomputers to mine for Bitcoins costing the university upwards of $150,000 [44]. On two other noteworthy occasions, NAS device botnets secretly mined for DogeCoin and Monero amounting to $600,000 and $82,000 respectively, before the covert operations were eventually discovered and shut down [14,29]. There are several other instances of employees and hackers secretly mining for coins in both the corporate [5] and government sectors [4].

This covert abuse of "borrowed" resources is not limited to enterprises and has also been observed in commercial clouds and datacenters [11]. The sheer amount of resources needed for a covert cryptomining operation are readily available in a cloud setting. Furthermore, since mined coins can easily be transferred to the attacker using a simple anonymized wallet address, it makes the "get away" scheme straightforward [1]. As a result, numerous instances of this targeted cloud abuse have already been uncovered, whereby attackers successfully *broke* into clouds and deployed cryptominers at a massive scale by spawning numerous VM instances dedicated exclusively to mining [9,11,18]. The advent of GPU clouds, such as those operated by Amazon and Microsoft, have further incentivized attackers to transfer their operations onto clouds and leverage the power of parallel computing, as GPUs often have higher hash rates and perform better for certain mining algorithms.

In this paper we present MineGuard, a simple hypervisor tool based on hardware-assisted behavioral monitoring, which accurately detects the signature of a miner. Specifically, our system uses Hardware Performance Counters (HPCs), a set of special-purpose registers built into modern processors, to accurately track low-level mining operations or events within the CPU and GPU with minimal overhead. This gives MineGuard the ability to accurately detect, in real-time, if a VM is trying to mine for cryptocurrency, without incurring any substantial slowdown ($<0.01\%$). MineGuard is built on the observation that for attackers to mine for any cryptocurrency, they will have to repeatedly run the core Proof-of-Work (PoW) algorithm that the currency is based on (such as Scrypt [32] for Litecoin) millions of times at the very least. Such repeated runs would substantially influence the count of certain HPCs in a particular way, which we can detect using a runtime checker. We empirically demonstrate very high detection rates ($\approx 99.7\%$), low false positives ($<0.25\%$) and false negatives ($<0.30\%$). Furthermore, our system does not modify any hypervisor code and leverages commonly available tools such as *perf* [19], thus making it easy to deploy and use in cloud and enterprise environments. We believe that attackers

cannot deceive MineGuard as (1) it attempts to catch the inherent mining behavior essential for mining and (2) it is more privileged than a VM and hence difficult to bypass. We make the following contributions:

Behavioral Analysis of Cryptomining: We perform a first-of-its-kind comprehensive study to explore the behavior of cryptocurrency mining focusing on micro-architectural execution patterns. Specifically, (1) we show that CPU/GPU signatures of mining and non-mining applications differ substantially; (2) different implementations of the same coin exhibit similar signatures due to the same underlying PoW algorithm, meaning that mining should be detectable by profiling an algorithm instead of the executing binaries (to overcome polymorphic malware) and (3) surprisingly, profiles of various coins exhibit overlapping signatures, despite having different PoW algorithms.

HPC Monitoring in Virtual Environments: While prior work has demonstrated the use of HPCs for malware detection, their utility and feasibility in a *virtualized* context has largely been ignored. We characterize the noise that is introduced into each HPC value individually due to virtualization, and show the best-fit distribution for this noise in each case. Our findings indicate that certain counters have a very pronounced noise-distribution, which can be used to error-correct the signatures. In contrast, some HPCs show negligible effects of noise. To incorporate this noise into our behavior profiles we develop a step-by-step signature creation process that captures an evolving profile of mining malware in increasingly noisier environments making our detection robust under different virtualized environments.

Userspace Detection Tool: We build a user space tool, MineGuard, that can run on top of any hypervisor or host OS and perform real-time detection. MineGuard has a negligible overhead, a small size footprint, is hard to evade, and cannot be compromised by malicious VMs. We believe MineGuard can be extended for other resource-intensive malware with minor modifications and serves as a valuable addition to the cloud security toolbox.

Paper Organization: We discuss the cost of covert cryptomining in Sect. 2 and how HPCs can be used to detect such miners in Sect. 3; followed by our system design in Sect. 4, methodology in Sect. 5 and evaluation in Sect. 6. Limitations are presented in Sect. 7 and related work in Sect. 8. Finally, we conclude in Sect. 9.

2 Understanding the Cost of Covert Cryptomining

Apart from using compromised accounts and hijacked VM instances for mining, hackers can also exploit the freemium business model of clouds. They can amass the complimentary resources allocated to individual accounts and build a large valuable pool [48,51], e.g., building an unlimited "slack space" on top of small free storage shares in Dropbox [43]. This issue has recently gained more traction amongst cloud providers with Google expressly forbidding any mining-related activity in its free tier resources [27]. Furthermore, providers also offer free resources under other specialized programs, such as to app developers and

students. These resources can also be abused in the aforementioned manner. As evidence to these freeloading issues, researchers recently constructed a mining botnet on Amazon entirely out of free resources [11]. The mining botnet was capable of generating cryptocurrency worth thousands of dollars and went completely undetected, despite its large footprint and conspicuous behavior.

These covert and cleverly concealed mining operations are a serious financial concern for admins. First, they waste valuable resources. Second, to maximize the hash rates hackers push CPUs/GPUs to full compute capacity for extended periods of time. This increases power consumption and generates heat, both of which impact operating costs [6]. Hence, it is imperative that mining deployments be thwarted before different losses stack up.

Users can't prevent this abuse as attackers can easily get root access and bypass security mechanisms or simply spawn their own VMs using stolen accounts. Similarly, providers and admins also struggle to mitigate these mining rigs [18], as they cannot distinguish mining from other types of workloads from outside the VM. Traditional VM instrospection techniques, such as analyzing memory dumps [41] or virtual disk monitoring [45], could be used but they have a large overhead and do not scale well. Also, if vendors start "peeking" into customers' VMs (e.g., by analyzing memory dumps), they run the risk of compromising the confidentiality and privacy of sensitive user data and computations.

Hence, a tool like MineGuard that proactively detects mining-related abuse (on free and stolen/compromised instances) and does not directly look at user data or code, is needed as a part of the provider's security toolbox.

3 Using Hardware Performance Counters

Past work has shown the effectiveness of hardware-based monitoring for malware detection [34,35,53–55] using architectural and microarchitectural execution patterns. The approach is predominantly characterized by an extremely low performance overhead making it ideal for real-time monitoring on latency sensitive systems. We build upon these past works and present the design of a system based on Hardware Performance Counters (HPCs) for detecting mining behavior on clouds/enterprises. HPCs, outlined in Table 2 later on, are a set of special purpose registers internal to the processor that record and represent the runtime behavior and characteristics of the programs being executed. Common examples include counts of page faults, executed instructions, cache misses etc. Though developed to aid application developers in fine-tuning their code, HPCs can also be used for behavior profiling without directly looking at code and data. Other than the fact that HPCs are extremely fast, their choice as the main detection metric is based on the following insights.

First, miners need to run the core PoW algorithm of a coin repeatedly, millions of times. If an algorithm A alters a few specific HPCs, say counters X, Y and Z, as part of the main hashing operations, then the values for these three counters should dwarf counts of all other (relatively under utilized) HPCs given

that algorithm A has to run millions of times. This implies that a very strong signature can be constructed based on the relevant counters of a particular algorithm, such as Scrypt [32] or CryptoNight [2]. If an adversary tries to stay under the radar by mining conservatively, then the hash rates will take a hit and profits will decline correspondingly making the exercise less lucrative. Also, since the processor will remain relatively under utilized, power and cooling costs will stay at manageable levels, making mining less of a nuisance for cloud vendors.

Second, any computation can only ever add to the values of HPCs and has no way of reducing counter values, as opposed to software-based defenses, which the attacker can subvert and alter. Hence, if an adversary mines for a coin, they will have no way of reducing counter values to avoid detection, and will be flagged with high likelihood. An adversary however, can try and neutralize the signature by increasing the values of other HPCs not associated with the PoW algorithm. But to do so successfully, the adversary has to overcome two *hard* challenges. First and foremost, they have to figure out a computation that **only** affects HPCs other than the ones related to the mining algorithm. In other words, there can be no overlap in the counters altered by the miner and the computation in question. Otherwise, the signature of the miner will only be bolstered further. Second, and more importantly, they have to run the secondary computation millions of times so that counter values are approximately equalized. However, the extra load on the system would greatly diminish the hash rate of the miner, reducing their profits.

Finally, HPCs are low-level registers and can be directly accessed by the hypervisor, requiring no modifications to the guest OS or applications. Furthermore, an adversary that manages to compromise a VM, even with root access, will not be able to falsify the values of the HPCs as the hardware does not allow this.

4 Design and Signature

The design of MineGuard was influenced by the following observations: First, unlike generic malware that can exploit users in novel ways, miners have to stick to the core PoW algorithm on which a cryptocurrency is based. This means that if a signature is built specifically for the algorithm, various implementations, even polymorphic and metamorphic ones, would be detectable. Second, detection has to be performed in a manner oblivious to the VM so that a malicious party cannot identify if they are being profiled or not, lest they start behaving differently. In addition, if a malicious entity does start behaving differently to cover up its tracks, it should incur a massive penalty, thereby defeating the whole purpose of the attack. Third, the detection mechanism has to be more privileged than the mining entity for obvious reasons. Finally, given the massive scale of clouds, the mechanism needs to be highly scalable with low performance overhead.

Given these stringent requirements, a hardware-assisted mechanism that can be executed on the host OS or the hypervisor emerged as the only logical candidate. As shown in Fig. 1A, MineGuard comprises of three components: A Profiler, a Detection Agent, and a Mitigation Agent. These three components run on each server in the cloud on top of the host or the hypervisor.

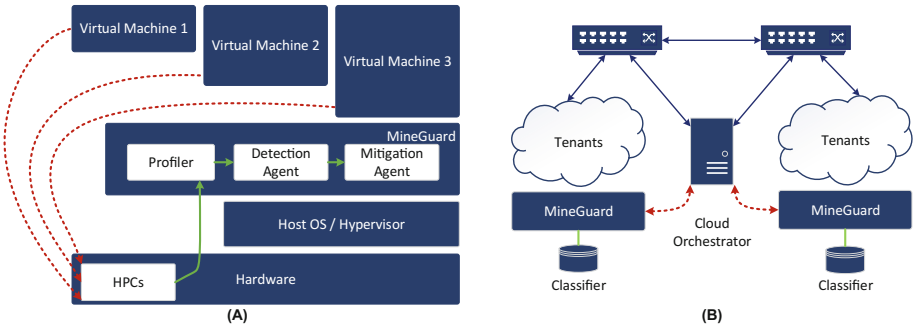

Fig. 1. (A) Inner components of a MineGuard instance. **(B)** Overview of MineGuard. Sequentially: MineGuard checks for current HPC values against the classifier. If a match occurs, it discovers all other VMs of the tenant and shuts down/suspends these VMs if they are also found mining.

The Profiler instruments each VM in real-time by polling the HPCs with a 2 s interval. The interval length is an adjustable metric, as MineGuard can use any user-defined sampling frequency to increase the accuracy even further. However, since mining is a long-term activity usually carried out for several hours at the very least (as opposed to short-term malware) we can easily afford to utilize large sampling intervals. This has the benefit of minimizing MineGuard's resource usage and does not effect the quality of the signature giving highly accurate detection rates as shown in Sect. 6. Furthermore, long intervals before repolling for HPCs, minimizes the overhead experienced by legitimate users as their VMs are profiled less often.

The Detection Agent runs the current HPC values against a classifier trained to detect mining behavior. If the classifier outputs a positive match, the Detection Agent flags the VM. Once a VM is flagged, the Mitigation Agent suspends it temporarily and determines the location of all VMs belonging to that tenant by contacting the cloud orchestrator as shown in Fig. 1B. All of the tenant's VM are then put to further screening by the Detection Agents on their corresponding servers. If more matches occur in this phase, the Mitigation Agents shut down those suspicious VMs as well.

Signature Creation: To incorporate the noise introduced by virtualization, we use a three-phased approach to creating accurate and precise mining signatures for both CPUs and GPUs. For our purposes, a signature is a time series of performance counter values of an application over a specified interval of time. To generate such time series, in the first phase, we run miners for various coins in a native environment and profile only the mining processes using *perf* [19] with a sampling rate of 2 s (empirically chosen for ease and accuracy). This gave us noise-free *process-level* HPC-based signatures for numerous coins. For GPUs we used *nvprof* [3]. The signature that we obtain during this phase is cleaned so that the bootstrapping code of the miner is not considered and only the signature

for the core PoW algorithm is captured. We call this signature the OS-level signature. In the second phase, we run miners inside VMs to cater to *noise* that is induced by executing in a virtualized environment. No additional processes are run on the VMs other than the default OS ones and our cryptominers, giving us pure *VM-level* signatures. This phase corresponds to a scenario in which an attacker uses dedicated VM instances for mining coins. Finally, in the last phase, we perform mining inside VMs that are already running other jobs and processes. This allows us to capture signatures in the presence of maximum noise. We repeat our experiments for various popular and common cloud workloads running in parallel with a cryptocurrency miner. Signatures generated during this phase are called *VM-Interference* signatures. The aforementioned scheme, explicitly captures the effects of virtualization-induced noise and workload-induced noise both of which are a must for efficient detection of mining activity.

Signature Database: MineGuard's signature database, which we use to train the classifier, is very small in size for numerous reasons. First, unlike generic malware, miners have to stick to a core PoW algorithm. Whether the miner is polymorphic, metamorphic or heavily obfuscated, the core algorithm, which we profile in our system, remains the same. Since there are a finite number of coins, and consequently a limited number of proof-of-work algorithms, added to the fact that there is no such thing as a zero-day coin, our resulting signatures are few in number (<100). This makes our signature database small. And, since each signature in the database is distinct and prominent compared to other common cloud workloads, as shown in Sect. 6, the classifier is able to build its inner models successfully.

5 Methodology

Before we jump into the results, we explain our prototype implementation and test environment, and present details of the cryptocurrencies, miners and benchmarks we used for testing and evaluation.

MineGuard Implementation: We implemented MineGuard in userspace using a combination of C++, Python and Bash. We used C++ for the signature creation and detection modules, and Bash and Python scripts for profiling VMs and collecting and organizing data. We used an open source random forest library [25] for the bagged decision tree implementation, and *perf/perf-KVM* [19] and *nvprof* [3] for CPU and GPU profiling, respectively. Upon deployment, a driver script samples any given process (application/miner/VM) for 2 s (equivalent to one test vector), formats the test vector and passes it to the predict module to classify the process. Excluding the random forest library, the entire MineGuard infrastructure only requires 282 lines of code. We have also made the source code and training/test data publicly available at the following URL: https://bitbucket.org/mhuzai/mineguard/overview.

Testbed: All experiments were performed on a machine with an Intel Core-i7 2600 K processor (Sandy Bridge), an NVIDIA GTX 960 GPU (Maxwell) and

Table 1. Cryptocoins we used along with their PoW algorithms and CPU/GPU miners.

Cryptocurrency	Proof-of-work algorithm	CPU miner	GPU miner
Bitcoin	SHA256	cpuminer-multi-windows, bfgminer-5.1.0, cgminer-2.11.4	–
Bytecoin	CryptoNight	cpuminer-multi-windows	ccMiner-cryptonight-0.17
Dash	X11	cpuminer-multi-windows	ccMiner-1.6.6-tpruvot
Litecoin	Scrypt	cpuminer-multi-windows	cudaminer-2014-02-28
Quarkcoin	BLAKE, Blue Midnight Wish, Grøstl, JH, SHA-3 and Skein	cpuminer-multi-windows	ccMiner-1.6.6-tpruvot
Vertcoin	Lyra2RE	cpuminer-multi-windows	ccMiner-1.6.6-tpruvot
Ethereum	Ethash (Modified Dagger-Hashimoto)	ethminer-1.3.0	ethminer-1.3.0
Zcash	Equihash	nheqminer-0.5c	nheqminer-0.5c

8 GB of DDR3 RAM. We ran Linux 3.16.0-44 for both desktop (native) and server (virtualized) environments. For collecting CPU-based training data, each application was profiled for 20 s, with one sample being collected every 2 s, for a total of 10 samples per application and miner. This provided ample data for high accuracy classification with negligible overhead (discussed in Sect. 6). For GPU-based training data, samples were only collected once at the end of a 120 s execution window - unlike *perf*, *nvprof* does not allow live periodic sampling of running applications.

Cryptocurrencies and Miners: Other than Bitcoin, the seven additional cryptocurrencies listed in Table 1 are still actively mined using CPUs and GPUs, and hence together comprise a realistic group for mining in the cloud. Furthermore, the currencies were chosen to evaluate a variety of mining algorithms and provide maximum coverage across the entire algorithm-space for mining-related PoW algorithms. The coins were also chosen to represent a large fraction of the market cap for mine-able coins (excluding Ripple, which cannot be mined, or coins with similar PoW algorithms, like Monero which is based on the same algorithm as Bytecoin). To mine these coins, we used cryptominers that were open-source and readily available online. Table 1 lists the cryptominers and mining algorithms for each of the cryptocurrencies used. Each miner was run using as

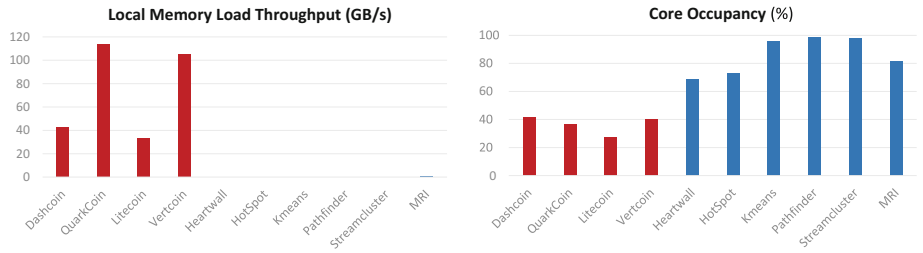

Fig. 2. Difference in behavior of GPU miners and GPU applications. Miners are shown in red; applications are shown in blue. (Color figure online)

many cores as available on the test system (8 cores for both the non-virtualized and virtualized environment) and public mining pools were used to mine coins. Using public pools ensured that our signature also incorporated the I/O aspects of miners, in addition to the dominant compute aspects. Finally, each miner was profiled in three different operating environments; **OS** (running standalone in a host OS), **VM** (running standalone in a guest OS) and **VM+Int** (running simultaneously with interfering applications in a guest OS).

Benchmarks and Cloud Workloads: To obtain signatures for non-mining applications, we chose various workloads from representative benchmark suites like CloudSuite (v3.0) [37], SPEC 2006 [20], Rodinia [33] and Parboil [49]. The benchmarks were chosen to cover a wide variety of domains, such as Hadoop workloads, scientific computing, AI simulations, data mining, graph analytics, web searching etc.; and a wide variety of workload characteristics such as compute and memory intensity, branch and cache behavior, and latency vs. throughput sensitivity. Furthermore, our mix of benchmarks consisted of both single-threaded and multi-threaded applications. We tested a total of 39 applications which we feel are representative of a real-world cloud setting.

Classification Algorithm and Evaluation Metrics: For evaluating our multi-class classification problem, we resorted to standard metrics like—*precision*, *recall*, and *F-score* [47] which is the harmonic mean between precision and recall. Since we do not know the underlying distribution of the different features for miners, we tried out different non-parametric classifiers like k-Nearest Neighbor (k-NN), Multiclass Decision Tree and Random Forest. We found that in general, ensemble-based approaches like Random Forest outperformed the other classifiers. During training, features from all applications (i.e., both miners and non-miners) were used to train the classifier. We used a random forest with 50 decision trees. In the test phase, the classifier predicted the most probable class for an unseen feature vector.[1]

[1] Unless otherwise stated, all experiments perform binary classification.

6 Evaluation

In this section we show empirical results from MineGuard, and present a discussion on various aspects and limitations of our system. Before moving onto the first set of results, we discuss the empirical overhead of our HPC-based approach. Prior work has shown in detail that the overhead of sampling counters, even in microsecond intervals (much more fine-grained compared to our approach), is *negligible* [35,46]. We observed very similar results with small values (<0.01%) for various polling intervals, hence, we do not present results for the overhead incurred due to space limitations and instead focus on other details surrounding MineGuard. Additionally, we found that the average time required to match a new sample against the classifier was 8 ms, bulk of which was spent in file I/O such as formatting the profiling data and reading the signature from disk. However, unnecessary I/O can be eliminated by keeping the signature in main memory. Finally, actual classification only took *32* μs, showcasing the low overhead nature of our design.

Uniqueness of GPU Mining Signatures: As explained above, MineGuard uses HPC-based signatures to detect miners in real time. We justify our choice of HPCs by demonstrating the uniqueness of mining behavior on GPU instances compared to other common and popular GPU-based workloads. Figure 2 presents this comparison between mining software and some popular and common GPU workloads taken from the Rodinia [33] and Parboil [49] GPU-benchmark suites. The figure shows the behavior of two different profiling metrics, out of a total of 28 GPU metrics, across four miners and six applications. We ran these experiments for several other benchmarks from the aforementioned benchmark suites and found consistent results. However, those results have been omitted for brevity. Some observations from our GPU results are discussed below.

Miners have significantly less core occupancy (number of actual threads out of maximum possible threads) than non-mining applications. This is due to the fact that, in general, it is a good practice to run as many threads as optimally possible on a GPU core, and therefore non-mining applications tend to have high core occupancy. Miners, on the other hand, also optimize for memory per warp (the basic unit of execution in NVIDIA GPUs), and aim to avoid creating bottlenecks in the memory system. Consequently, they usually exhibit low core occupancies.

Another noticeable difference between miners and non-mining applications is their usage of local memory. Local memory in NVIDIA GPUs is used for register spilling and per-thread local data. However, despite its name, local memory physically resides in the main memory of the GPU and as such it is not as fast as scratchpads or texture caches. As a result, GPU application programmers tune their code to minimize local memory usage as much as possible. As can be seen in Fig. 2, the six different non-mining applications have in fact no local memory usage (an exception is MRI, which does use local memory but does so minimally). Miners, in stark contrast, exhibit high usage of local memory. This is a consequence of the fact that mining algorithms require a significant

number of registers and this in turn results in a significant number of register spills (note: the high register usage of these algorithms also contributes to the low core occupancy).

As evident, there is a marked difference between the performance counter profiles of GPU miners and typical GPU applications. It is precisely these differences that our classification algorithm relies upon to detect miners with high accuracy.

Uniqueness of CPU Mining Signatures: As with GPU-based miners, we collected HPC-based signatures for CPU-based miners as well. These signatures were then compared to common CPU-based workloads from CloudSuite and the SPEC2006 benchmark suite to distinguish CPU miners from non-mining applications. The unique and distinct characteristics of CPU-based miners, similar to their GPU counterparts, can be seen in Fig. 3. The figure shows subgraphs for two different HPCs, out of a total of 26 CPU HPCs shown later in Table 2. Both subgraphs show a live-trace of a HPC's value during the execution of a CPU-based miner mining Litecoin and four non-mining applications; namely data caching (memcached server), AI (game of Go), H264 (hardware video encoding) and NAMD (molecular dynamics). The results from other benchmarks have been omitted for clarity.

In both graphs, the mining signature stands out. Since miners repeatedly run a small set of computations over and over again for millions of times, their resource usage is consistent throughout their execution. In other words, miners generally do not exhibit irregular *phases* as most common applications do. Rather, miners possess regular and structured phases. This consistency in signature is represented by a step function like recurring pattern in both graphs (red line).

On the other hand, non-mining applications and workloads have phases that are noticeably different. While the phases are, like miners, repeated in regular intervals, the behavior of each phase is much more irregular and possesses a high degree of variance (a finding consistent with prior research [35]). These patterns are particularly visible for H264 (black line). For example, the L1 Store curve of H264 is rhythmic but irregular, and, in fact, we found that troughs in load count

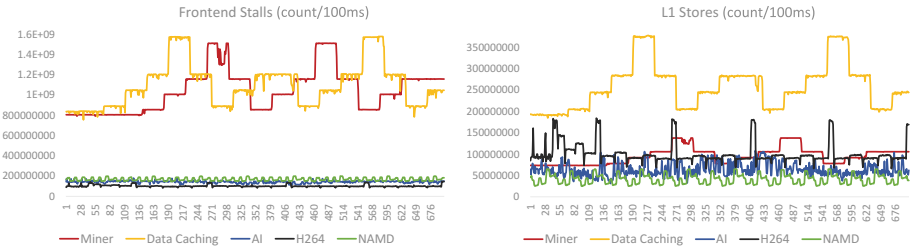

Fig. 3. Difference in behavior of a Litecoin CPU miner and four representative CPU applications. The x axis shows time in units of 100 ms (miner in red). (Color figure online)

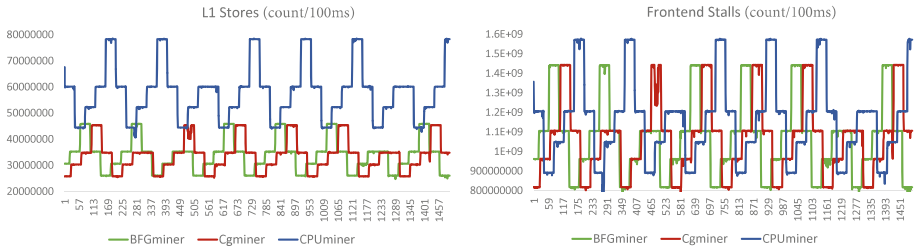

Fig. 4. Similarity in behavior of three different Bitcoin mining softwares. The x axis shows time in units of 100 ms.

correspond to peaks in store count. Similarly, another interesting observation is that for the Litecoin miner, the curves for the HPCs closely follow each other - an increase in one is accompanied by an increase in the other, which is generally not the case in the other workloads. Finally, even though data caching exhibits a slight similarity to Litecoin for these two HPCs, it is quite different for all metrics taken together. We take away the following insight from these results: CPU-miners exhibit a unique HPC-based signature and this signature can be effectively leveraged to detect virtual machines that are performing cryptocurrency mining.

Signature Homogeneity Within a Coin: Hackers usually employ various techniques and mechanisms to bypass detection mechanisms. They use polymorphic, metamorphic and obfuscated malware to fool anti-virus software and runtime checkers by completely overhauling their codebase. To show MineGuard's resilience against these techniques, we demonstrate how three completely different miner implementations that are mining the same coin still exhibit the same HPC-based signature.

Figure 4 shows two graphs, one per HPC, for three different miners all mining for Bitcoin. The implementations of these miners are quite different from one another, however, the graphs all show similar HPC patterns, thereby backing our claim that the mining signature is consistent across different implementations. The reason behind this, as mentioned previously, is that at their core, all miners have to abide by a fixed PoW algorithm. Not only does this limit the amount of variability that can be afforded by different implementations, but since the algorithm is run millions of times, it dwarfs any differences that are present in polymorphic or metamorphic versions of the mining malware. Consequently, the resulting signatures only have minor variations from miner to miner. These variations are broadly manifested across three categories. Phase shifts (where the patterns are offset from each other by a small time delta), differences in magnitude and occasionally in curve shape. We found that these changes are rare and usually impact one or two HPCs largely keeping the signature similar across implementations. MineGuard exploits this uniformity during its detection phase allowing it to catch altered versions of a mining malware.

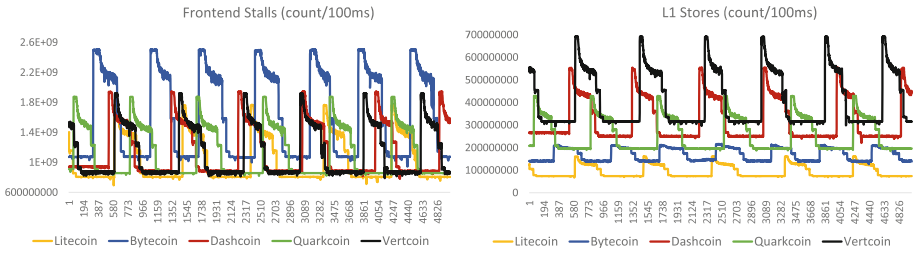

Fig. 5. Similarity in behavior of various cryptocurrencies (algorithms). The x axis shows time in units of 100 ms.

Signature Homogeneity Across Coins: We also claim that different cryptocurrencies have similar signatures due to the nature of cryptomining. As evidence, we present the signatures of five different cryptocurrencies in Fig. 5. The figure shows a subset of the signatures of cryptominers mining Litecoin, Bytecoin, Dashcoin, Quarkcoin and Vertcoin. It is immediately obvious that all five signatures follow the same pattern - periods of constant computation (the flat part of the curves, corresponding to hashing) punctuated by phases of exponentially decaying irregular code that executes when new blocks are found, the mining difficulty is changed, I/O is performed, etc. The only differences are in the magnitudes of the various HPC values, which can be attributed to different PoW algorithms having higher or lower operation counts. However, when looking at the combined signature of all HPCs, the similarities dwarf the differences, as shown in Fig. 5.

Effects and Characterization of Noise: So far, we have discussed the signatures of miners and various other applications that were obtained in a non-virtualized environment (OS). Although these signatures aptly present the similarities and differences between various miners and non-mining applications, they do not account for VM noise that would naturally be added when the aforementioned software are executed in a virtualized environment (guest OS) and profiled from the hypervisor. Since monitoring virtual machines is the primary role of MineGuard, we characterize this noise and study its effects on mining signatures.

By performing per feature noise profiling (on both OS and VM environments using all miner and cloud workloads) for all 26 HPCs (see Table 2), we found that roughly one-fourth of the counters show variation in values due to noise e.g., cycles, instructions, stalled-cycles-fronted, context switches etc. Figure 6A shows the process via which we arrived at the best fit, which was determined using the Akaike Information Criterion (AIC) [28]. The empirical data points (blue bars) represent the noise added as we moved from native to in-VM execution. The colored curves represent various distributions superimposed on top. As evident, a vivid pattern can be extracted based on the distribution and later used for error correction. Similarly, Fig. 6B shows the probability density functions for a few HPC counters. The best fit distributions in the depicted cases were *Nakagami* (cycles, stalled-cycles-frontend) and *Burr* (instructions, stalled-cycles-backend)

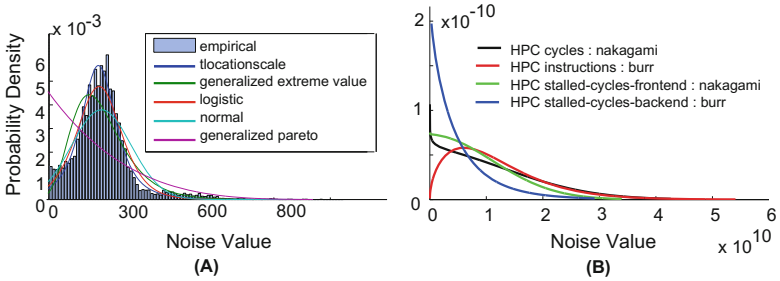

Fig. 6. (A) The fitting process via which we arrive at the final best fit distribution (*t*Location-Scale) for the context switch counter (ID 10). **(B)** Noise distribution for number of instructions counter (ID 2). The best fit distribution in this case is Burr.

distributions. Other HPCs, such as context switches, followed the *t*Location-Scale distribution. We found that three-fourths of the counters have negligible change to their values or patterns when we move from OS to VM. This fact justifies our choice of HPCs for MineGuard given that virtualization has limited impact on HPCs. Furthermore, if necessary, the discovered distributions can be factored into the signatures to develop error-corrected profiles for even more robust and accurate signatures.

To visually demonstrate how this noise distorts signatures, we present graphs for native vs in-VM execution of miners in Fig. 7A. The graph depicts the values of the L1 Loads counter. The curves have become more jagged and noisy as the system processes of the guest OS influence counter values, but their involvement results in a *minimal degradation* of the signature. For example, the peaks and troughs can still be clearly seen. Similarly, the slopes are unchanged and the noisy plateaus are still flat, preserving the consistent behavior of miners. All this follows from the fact that most HPCs do not suffer from virtualization-induced noise as shown above and maintain unique patterns and characteristics associated with mining.

MineGuard Under an Active Adversary: In an attempt to throw off MineGuard, a clever attacker could selectively create noise in the HPCs by running computations in parallel that influence counters not involved in mining. This would artificially inflate the values and modify patterns of certain HPCs that are irrelevant to the mining signature and appear as a benign workload to the classifier. To check the effectiveness of this scheme we performed an experiment with Litecoin where we modified the miner's code to run a computation in parallel that predominantly affects the set of mining-orthogonal counters (HPCs not showing significance use during mining). We measured how increasing the number of threads for the extra computation negatively impacts the total hash rate along with the corresponding reduction in MineGuard's detection accuracy. Figure 7B captures this relationship for 100 different runs of the aforementioned experiment. As expected, increasing the number of threads for the camouflaging computation severely degrades the hash rate (base hash rate is approximately

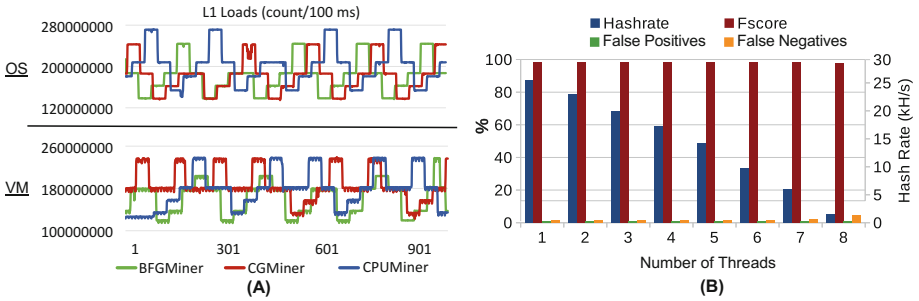

Fig. 7. (A) Effect of virtualization-induced noise on L1 Loads for various miners all mining for Bitcoin. The x axis shows time in increments of 100 ms. **(B)** Degradation of mining hash rate as the number of masking threads is increased. The hash rate falls consistently while the detection rate remains constant throughout, with only a slight increase in false negatives as 8 threads are used.

30 kH/s). However, it has very little impact on the detection rate meaning that the exercise would not be of benefit to the attacker. Granted, the experiment covers only a small subset of the overall computation space available to the attacker, we still feel that the impact suffered by the hash rate will be much more severe compared to the hit taken by the classifier in nearly all cases.

Feature (Counter) Selection: We now present a formal approach to feature (counter) selection to determine the importance of each counter, both by itself and in relation to other counters. When looking at each counter individually, we use mutual information to determine its importance. The mutual information (MI) of two random variables is a measure of the mutual dependence between the two variables. More specifically, it quantifies the "amount of information" (in units such as bits or entropy) one random variable contributes to generating a unique signature for the miner. When looking at multiple features together, their importance as a whole is represented by joint mutual information (JMI), a measure of the features' combined entropy. JMI can then be used to rank features from most important to least important. In turn, the ranking can be used to choose the minimum number of features that provide the best classification accuracy.

Table 2 lists the 26 different counters that were available on our system. To obtain MI and JMI for each counter, we used FEAST, an open source toolbox for feature selection algorithms [31]. The entropy (MI) of all 26 counters, both in an OS setting and in a VM setting, is shown in Fig. 8. It can be seen that features can be broadly divided into three categories. First, certain features like feature ID 1 (clock cycles), 5 (bus cycles) and 8 (task clock) hold a significant amount of information in both OS and VM environments. Second, features like feature ID 9 (page faults) and 10 (context switches) contribute negligibly to the classification process in both environments. Finally, the remaining features provide varying amounts of information depending upon the environment. While

Table 2. HPCs used for CPU-based signatures along with their JMI rank and explanation.

Name of counter	Counter ID	OS rank	VM rank	Explanation
cycles	1	4	4	# of CPU clock cycles
instructions	2	6	6	# of executed instructions
branches	3	2	19	# of branch instructions
branch-misses	4	16	15	# of mispredicted branches
bus-cycles	5	8	1	# of useful bus cycles
stalled-cycles-frontend	6	1	5	# of stalled cycles in frontend of pipeline
stalled-cycles-backend	7	11	16	# of stalled cycles in backend of pipeline
task-clock	8	3	3	CPU time in milliseconds
page-faults	9	26	26	# of page faults
context-switches	10	24	24	# of context switches
cpu-migrations	11	25	25	# of migrations of profiled app
L1-dcache-loads	12	13	14	# of loads at L1 data cache
L1-dcache-load-misses	13	21	13	# of load misses at L1 data cache
L1-dcache-stores	14	7	7	# of stores at L1 data cache
L1-dcache-store-misses	15	14	8	# of store misses at L1 data cache
L1-dcache-prefetch-misses	16	18	17	# of misses at L1 cache that did not benefit from prefetching
L1-icache-load-misses	17	15	10	# of instruction fetches missed in the L1 instruction cache
LLC-loads	18	12	11	# of loads at the Last Level Cache
LLC-stores	19	20	2	# of loads that missed in the data TLB
LLC-prefetches	20	9	20	# of stores that queried the data TLB
dTLB-loads	21	5	12	# of stores at the Last Level Cache
dTLB-load-misses	22	17	21	# of prefetches at the Last Level Cache
dTLB-stores	23	10	9	# of loads that queried the data TLB
dTLB-store-misses	24	23	22	# of stores that missed in the data TLB
iTLB-loads	25	19	23	# of instruction fetches that queried the instruction TLB
iTLB-load-misses	26	22	18	# of instruction fetches that missed in the instruction TLB

the general trends are the same in both environments, the differences between the two graphs present the importance of performing feature selection for each environment.

We present feature ranking results for both OS and VM environments based on JMI in Table 2. Feature rankings mimic the patterns observed in MI - certain features like 2 (instructions) do not change rank while others like 3 (branches) change rank significantly. Another interesting observation is that system level events like page faults and context switches have a low rank while purely hardware-based events like loads and stores are ranked highly in both scenarios.

Classification Accuracy: We now present results for MineGuard's miner detection performance in both closed and open world setting. A **closed world setting** is a scenario in which every cryptocurrency that MineGuard can be

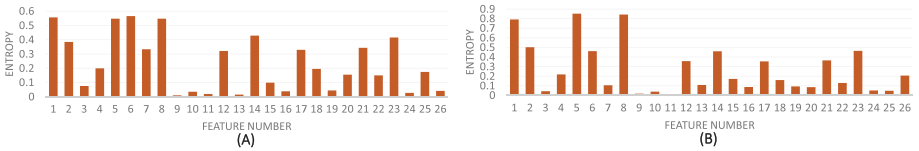

Fig. 8. The mutual information (entropy) contained within each hardware performance counter for **(A)** an OS environment, **(B)** a VM environment.

requested to detect is a part of the signature database. The test sample may vary from the signatures stored in the database but as we have previously shown, miners have unique and consistent signatures, increasing the likelihood if the test sample is from a miner, it will be matched to a miner in the signature database.

Table 3. Classification results for three different operating environments in a closed world setting. Each result's 95% confidence interval is written in brackets.

Closed world scenario	F-score (CI)	False positives (CI)	False negatives (CI)
OS-Level	99.69% (0.13%)	0.22% (0.11%)	0.29% (0.25%)
VM-Level	99.69% (0.17%)	0.27% (0.14%)	0.26% (0.18%)
VM-Interference	99.15% (0.11%)	2.12% (0.29%)	0.04% (0.03%)
Open world scenario	F-score (CI)	False positives (CI)	False negatives (CI)
OS-Level	94.91% (1.02%)	4.50% (1.13%)	2.58% (1.64%)
VM-Level	93.58% (1.33%)	5.63% (1.61%)	4.52% (2.41%)
VM-Interference	95.82% (0.86%)	6.77% (1.45%)	2.53% (1.54%)

Table 3 shows our results for this scenario where all values are reported after 100 runs. Since MineGuard has been trained on every cryptocurrency in the test set, it achieves an exceptionally high miner detection accuracy. It achieves ≈99.5% accuracy with a false positive rate (FPR) of 0.22% and false negative rate (FNR) of 0.29% when classifying miners running solely in either the OS and VM setting. *This equates to near-perfect miner detection and implies that if a known cryptocurrency is being mined in an OS or a VM, MineGuard will detect it almost every time.* When classifying miners running with other applications, the average *F-score* drops to 99.15% and FPR increases to 2.12%, while FNR remains at ≈0%. Even in an **open world setting**, where all test signatures are unseen (i.e., miners in the test set are unknown to the classifiers), MineGuard still achieves accuracy ≈95% for all three cases. Though the results are slightly worse than a closed world setting, they are still satisfactory overall. Furthermore, as we explain in Sect. 7, unseen signatures are rare as zero-day coins are an unlikely scenario.

The results shown in Table 3, have been computed on a per sample basis. This means that the classifier treats each 2 s sample of HPC values as an independent

test vector rather than labeling all samples collectively as miner/non-miner. An alternate way is to use per application classification and treat all samples collected from a running process as a single test vector. This approach has the advantage that given the number of samples for a particular application, the classification can be done using various ratios. For example, if 5 samples for an application are available, if one is categorized as miner the entire application is labeled as a miner. Similarly, we can use a scheme where all samples need to be classified as a miner or use a simple majority rule (3 out of 5 classified as miner then app is miner). In each case, the corresponding F-score, FPR and FNR would be different. In Table 4, we present open world results for a simple majority scheme (though other settings can also be used such as classification based on 33% match or 75% match etc.) using per application testing.

Table 4. Classification results for three different operating environments in a open world setting when all samples are treated collectively (per application processing).

Open world scenario	F-score (CI)	False positives (CI)	False negatives (CI)
OS-Level	93.85% (2.68%)	0.0% (0.0%)	9.70% (3.77%)
VM-Level	91.67% (3.16%)	0.0% (0.0%)	16.33% (5.83%)
VM-Interference	96.32% (1.75%)	0.0% (0.0%)	7.99% (4.10%)

The results show that the F-score is still high. The corresponding FPRs for our simple majority scheme are zero in all cases, which eliminates the primary concern in our setting as legitimate tenants would rarely be flagged or shut down. The reason for the 0% FPR is that previously, we were classifying each 2 s HPC sample individually. In such a scenario there is a possibility that a particular sample belonging to a non-miner exhibits HPC values matching those of a miner (perhaps due to a hashing intensive phase in the execution). However, since now we're looking at all samples of a single application collectively, the chances of all samples being hashing intensive (or even a majority of them) for a non-miner app are rare and hence the 0% FPR. The corresponding FNRs are a bit high, however this is less of a concern for the following reasons. First, since mining is commonly a long-term activity the attacker will eventually get flagged in a subsequent scan even if he evades the classifier once or twice. Second, if the attacker uses multiple VMs to form a personal mining pool, then with high likelihood one of their VMs will get flagged (even if other VMs successfully evade the classifier), which would trigger MineGuard to immediately scan all other VMs that are part of the same deployment again and if more VMs are caught, the cloud provider can do a more comprehensive check of the entire deployment using VMI or other more invasive tools.

Taken collectively, these results indicate that MineGuard is extremely adept at identifying miners running in any operating environment. Even in the worse case of detecting miners running in noisy environments, it achieves very high accuracy.

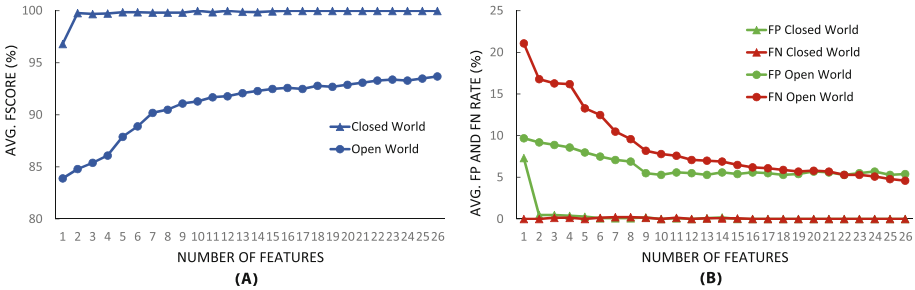

Fig. 9. Accuracy of miner classification in a VM environment, in terms of **(A)** average *F-score*, **(B)** average false positive rate and average false negative rate, as the number of features is increased.

Effect of Signature Size on Accuracy: Figure 9 captures the relationship between the size of the signature (number of top counters used) and the accuracy of detection in a VM environment for both open and closed world settings. As shown in Fig. 9A, for the closed world scenario (triangles) even when only 2 counters are used, we achieve an average *F-score* above 99.5%, an average false positive rate (FPR) below 0.5% and an average false negative rate (FNR) of approximately 0, shown in Fig. 9B. This implies that MineGuard can actually work with very small signature footprints speeding up all processes from profiling to matching. Similarly, in the open world case (circles) with only 3 counters the average F-score is around 85% and jumps to 90% if we consider the top 7 counters. Increasing the size further brings marginal increases that ultimately take the detection rate close to 95% for all 26 counters. An opposite downward trend is observed in the average values of FP and FN for the open world case as shown in Fig. 9B, with the rates declining all the way to roughly 5% when the entire vector of HPCs is used. These numbers might appear a bit high, but as we argue in Sect. 7 the open world case is highly unlikely as unseen mining algorithms are an extremely rare possibility.

7 Discussion

We discuss a few aspects of our work in this section and explore potential limitations.

Custom or Zero-Day Cryptocurrency: Is MineGuard vulnerable to zero-day or custom coins? We believe it is not. By definition, zero-day or custom coins do not exist because for a coin to have any dollar value, it first needs to have a known PoW algorithm, needs to be recognized by the cryptocommunity as mathematically sound and has to be adopted at a moderate-to-large scale for the core network to exist. Therefore, the first time a new piece of malware for some new coin makes an appearance in the wild, it would already be well-known in the cryptocurrency community, giving cloud vendors enough time to

train MineGuard on the new coin's signature, as its algorithm would be public knowledge as mentioned above.

Evasion: An attacker can employ several techniques to try and evade MineGuard's detection scheme. First, they could employ known techniques of software obfuscation. However, since we target the algorithm and not the implementation, we believe that the attacker would have limited success (as shown in Sect. 6). Second, the attacker could artificially manipulate the counters by performing alternate computations that modify a distinct set of counters orthogonal to the ones used in mining. Again, as we have shown in Sect. 6, this would severely penalize the hash rate of the attacker while having very limited impact on his chances of evading MineGuard. Thirdly, the attacker could attempt to stay under the radar and mine at an extremely low hash rate. Theoretically, this is a limitation since the attacker can evade MineGuard by sticking to low hash rates. However, we argue that the whole exercise becomes non-profitable for the attacker and nullifies the whole point of mining on clouds. Furthermore, low hash rates eliminate the original problem of resource abuse making it less of a nuisance. Finally, the attacker could try to determine when the VM is being profiled by MineGuard and stop mining temporarily. However, there are numerous issues with this evasion scheme. First, since there is no measurable profiling overhead, it is hard for an adversary to tell if their VM is being profiled. Second, instead of monitoring the VMs in a round-robin fashion, the hypervisor can monitor the VMs randomly, making it impossible to predict when a VM would be profiled.

Coin Diversity: We could not perform analysis on all cryptocurrencies available in the market and chose to work with a popular subset (choosing coins with distinct algorithms and ignoring those which were forks of popular coins) as shown in Table 1. Additionally, with the above restriction in mind we selected coins that collectively comprise the largest share of market cap. Also, we justify our choice by highlighting that most cryptocurrency exchanges, like Kraken [23], only deal with the top 25–30 cryptocurrencies, as other altcoins have exceptionally low dollar value and profit margins from transactions are very low [22]. Moreover, documented cases of cryptocurrency Trojans have been mostly limited to the top 10–15 coins [12,13,16,21,24]. Hence, attackers avoid wasting precious hashes on less valuable coins, which is why we chose our subset of popularly used coins. Nevertheless, we list this as a limitation, since the possibility, however minute, of an excluded coin's signature matching a cloud app still remains.

8 Related Work

Cloud abuse has become a hot topic of research. Recent efforts [40,52] have been geared towards developing a sound understanding of the problems and vulnerabilities inherent to clouds. Others have demonstrated novel ways of exploiting these vulnerabilities by building practical systems that are of value to attackers, such as file sharing applications [51], unlimited storage banks [43] and email-based storage overlays [48]. To mitigate these concerns, researchers have

proposed various Virtual Machine Introspection (VMI) approaches [36,42,45]. However, some of these are voluntary and require user participation [30], which of course the attacker wants no part of, and others have a large overhead [41]. Furthermore, these VMI-based approaches are designed to observe the memory, disk and processor state of customers' VMs, which is a serious privacy concern given the sensitive nature of customer data.

A different yet related line of work attempts to describe the infrastructure and mechanism of mining botnets. Huang et al. [39] present a thorough investigation of mining ecosystems in the wild. They claim that mining is less profitable than other malicious activities, such as spamming or booter-renting (DDoS for hire), and should be used as a secondary monetizing scheme. However, we believe that it is unfair to compare mining profits with other monetizing activities as the price of coins varies substantially over time and as of this writing, the value of one Bitcoin is a $1000 (and rising) as opposed to $100 in 2013, which demonstrates that mining can generate an order of magnitude more revenue now. Furthermore, as mining uses an orthogonal set of resources (CPU/GPU and memory) compared to DDoS attacks (network), we postulate that botnet-herders should maximize their profits by running various resource-disjoint monetizing activities in parallel making a strong case for covert cryptomining. Indeed, Sophos Security presented evidence that mining botnets could potentially generate around $100,000 per day of profits for herders [8].

Finally, there has been much research on detecting generic malware using architectural and microarchitectural execution patterns, such as HPCs, with differing results. Demme et al. [35] built a system for detection of generic malware and demonstrate the feasibility of the design based on ARM (Android) and Intel (Linux) platforms. Other researchers [38,50,55] have also used low-level hardware features to promising success, furthering the work of Demme et al. In addition to generic malware, HPCs have also been successfully used to detect kernel-level rootkits [53], side-channel attacks [34], firmware modifications [54] etc. However, none of these previous works try to accommodate the noise introduced by virtualization, as we do in this work.

9 Conclusion

We present MineGuard, a userspace tool that prevents abuse of resources at the hands of hackers interested in mining cryptocurrencies on others' resources. Whether the mining operation is local (restricted to one VM) or being conducted in a pool of participating VMs, MineGuard can successfully detect and shutdown the illegitimate mining "ring". We empirically demonstrate that our design imposes negligible overhead to legitimate tenants and can detect mining in real-time with high precision. If multiple VMs are involved in mining, MineGuard can collaborate with other MineGuard instances to expose the entire footprint of the mining deployment. For detection, MineGuard uses signatures based on Hardware Performance Counters for both CPU and GPU-based miners. The fact that MineGuard runs on top of the hypervisor or the host OS

prevents miners running inside the VMs from subverting it despite root access on the guest. We also account for the noise generated as a result of virtualization to provide error correction for our detection mechanisms. In the future, we plan to extend MineGuard to accurately detect other types of malwares in highly multiplexed and virtualized environments.

References

1. Bitcoin Anonymizer TOR Wallet. https://torwallet.com/
2. CryptoNight. https://en.bitcoin.it/wiki/CryptoNight
3. CUDA Toolkit Documentation. https://tinyurl.com/z7bx3b3
4. Government employee caught mining using work supercomputer. https://tinyurl.com/mrpqffd
5. ABC employee caught mining for Bitcoins on company servers (2011). https://tinyurl.com/lxcujtx
6. Data Center Power and Cooling. CISCO White Paper (2011)
7. How to Get Rich on Bitcoin, By a System Administrator Who's Secretly Growing Them On His School's Computers (2011). https://tinyurl.com/lwx8rup
8. The ZeroAccess Botnet - Mining and Fraud for Massive Financial Gain (2012). https://tinyurl.com/ldgcfao
9. Online Thief Steals Amazon Account to Mine Litecoins in the Cloud (2013). https://tinyurl.com/mzpbype
10. Harvard Research Computing Resources Misused for Dogecoin Mining Operation (2014). https://tinyurl.com/n8pzvt6
11. How Hackers Hid a Money-Mining Botnet in the Clouds of Amazon and Others (2014). https://tinyurl.com/mowzx73
12. List of Major Bitcoin Heists, Thefts, Hacks, Scams, and Losses (2014). https://bitcointalk.org/index.php?topic=576337
13. Mobile Malware Mines Dogecoins and Litecoins for Bitcoin Payout (2014). https://tinyurl.com/q828blg
14. NAS device botnet mined $600,000 in Dogecoin over two months (2014). https://tinyurl.com/myglgoa
15. US Government Bans Professor for Mining Bitcoin with A Supercomputer (2014). https://tinyurl.com/k3ww4rp
16. Adobe Flash Player Exploit Could Be Used to Install BitCoinMiner Trojan (2015). https://tinyurl.com/lhxzloa
17. Cloud Mining Put to the Test- Is It Worth Your Money? (2015). https://tinyurl.com/zquylbo
18. Developer Hit with $6,500 AWS Bill from Visual Studio Bug (2015). https://tinyurl.com/zm3pzjq
19. Perf Tool Wiki (2015). https://tinyurl.com/2enxbko
20. Standard Performance Evaluation Corporation (2015). https://www.spec.org/benchmarks.html
21. Trojan, C.: A Grave Threat to BitCoin Wallets (2016). https://tinyurl.com/k73wdaq
22. Crypto-Currency Market Capitalizations (2016). https://coinmarketcap.com/
23. Kraken Bitcoin Exchange (2016). https://www.kraken.com/
24. Linux. Lady. 1 Trojan Infects Redis Servers and Mines for Cryptocurrency (2016). urlhttps://tinyurl.com/ka9ae4c

25. Randomized Decision Trees: A Fast C++ Implementation of Random Forests (2016). https://github.com/bjoern-andres/random-forest
26. Student uses university computers to mine Dogecoin (2016). https://tinyurl.com/lubeqct
27. Supplemental Terms and Conditions For Google Cloud Platform Free Trial (2017). https://tinyurl.com/ke5vs49
28. Akaike, H.: A new look at the statistical model identification. IEEE TAC 19 (1974)
29. Marosi, A.: Cryptomining malware on NAS servers (2016)
30. Baek, H.W., Srivastava, A., van der Merwe, J.E.: Cloudvmi: virtual machine introspection as a cloud service. In: 2014 IEEE International Conference on Cloud Engineering (2014)
31. Brown, G., Pocock, A.C., Zhao, M., Luján, M.: Conditional likelihood maximisation: a unifying framework for information theoretic feature selection. In: JMLR (2012)
32. Percival, C., Josefsson, S.: The Scrypt Password-Based Key Derivation Function. IETF (2012)
33. Che, S., et al.: Rodinia: A benchmark suite for heterogeneous computing. In: Proceedings of the 2009 IEEE International Symposium on Workload Characterization (2009)
34. Chiappetta, M., Savas, E., Yilmaz, C.: Real time detection of cache-based side-channel attacks using hardware performance counters. IACR Cryptol. ePrint Archive **2015**, 1034 (2015)
35. Demme, J., Maycock, M., Schmitz, J., Tang, A., Waksman, A., Sethumadhavan, S., Stolfo, S.J.: On the feasibility of online malware detection with performance counters. In: The 40th Annual ISCA (2013)
36. Dinaburg, A., Royal, P., Sharif, M.I., Lee, W.: Ether: malware analysis via hardware virtualization extensions. In: ACM CCS (2008)
37. Ferdman, M., Adileh, A., Koçberber, Y.O., Volos, S., Alisafaee, M., Jevdjic, D., Kaynak, C., Popescu, A.D., Ailamaki, A., Falsafi, B.: Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In: ASPLOS (2012)
38. Garcia-Serrano, A.: Anomaly detection for malware identification using hardware performance counters. CoRR (2015)
39. Huang, D.Y., Dharmdasani, H., Meiklejohn, S., Dave, V., Grier, C., McCoy, D., Savage, S., Weaver, N., Snoeren, A.C., Levchenko, K.: Botcoin: monetizing stolen cycles. In: NDSS (2014)
40. Idziorek, J., Tannian, M.: Exploiting cloud utility models for profit and ruin. In: IEEE CLOUD (2011)
41. Jiang, X., Wang, X., Xu, D.: Stealthy malware detection and monitoring through VMM-based "out-of-the-box" semantic view reconstruction. ACM Trans. Inf. Syst. Secur. **13**(2), 12:1–12:28 (2010)
42. Lengyel, T.K., Neumann, J., Maresca, S., Payne, B.D., Kiayias, A.: Virtual machine introspection in a hybrid honeypot architecture. In: CSET (2012)
43. Mulazzani, M., Schrittwieser, S., Leithner, M., Huber, M., Weippl, E.R.: Dark clouds on the horizon: using cloud storage as attack vector and online slack space. In: 20th USENIX Security Symposium (2011)
44. National Science Foundation Office of Inspector General: SEMIANNUAL REPORT TO CONGRESS (2014)
45. Payne, B.D., Lee, W.: Secure and flexible monitoring of virtual machines. In: ACSAC (2007)
46. Sembrant, A.: Low Overhead Online Phase Predictor and Classifier. Master's thesis, UPPSALA UNIVERSITET (2011)

47. Sokolova, M., Lapalme, G.: A systematic analysis of performance measures for classification tasks. Inf. Process. Manage. **45**, 427–437 (2009)
48. Srinivasan, J., Wei, W., Ma, X., Yu, T.: EMFS: email-based personal cloud storage. In: NAS (2011)
49. Stratton, J.A., et al.: Parboil: A revised benchmark suite for scientific and commercial throughput computing. In: IMPACT Technical report (2012)
50. Tang, A., Sethumadhavan, S., Stolfo, S.J.: Unsupervised anomaly-based malware detection using hardware features. In: Stavrou, A., Bos, H., Portokalidis, G. (eds.) RAID 2014. LNCS, vol. 8688, pp. 109–129. Springer, Cham (2014). doi:10.1007/978-3-319-11379-1_6
51. Tinedo, R.G., Artigas, M.S., López, P.G.: Cloud-as-a-gift: effectively exploiting personal cloud free accounts via REST apis. In: IEEE CLOUD (2013)
52. Vaquero, L.M., Rodero-Merino, L., Morán, D.: Locking the sky: a survey on IaaS cloud security. Computing **91**(1), 93–118 (2011)
53. Wang, X., Karri, R.: Numchecker: detecting kernel control-flow modifying rootkits by using hardware performance counters. In: The 50th Annual DAC (2013)
54. Wang, X., Konstantinou, C., Maniatakos, M., Karri, R.: Confirm: Detecting firmware modifications in embedded systems using hardware performance counters. In: ICCAD (2015)
55. Yuan, L., Xing, W., Chen, H., Zang, B.: Security breaches as PMU deviation: detecting and identifying security attacks using performance counters. In: APSys (2011)