Detecting Covert Cryptomining using HPC

Gangwal, Ankit; Piazzetta, Samuele Giuliano; Lain, Gianluca; Conti, Mauro

DOI
10.1007/978-3-030-65411-5_17

Publication date
2020

Document Version
Final published version

Published in
Cryptology and Network Security

Citation (APA)
Gangwal, A., Piazzetta, S. G., Lain, G., & Conti, M. (2020). Detecting Covert Cryptomining using HPC. In S. Krenn, H. Shulman, & S. Vaudenay (Eds.), *Cryptology and Network Security: 19th International Conference, CANS 202, Proceedings* (pp. 344–364). (Part of the Lecture Notes in Computer Science book series (LNCS, volume 12579); Vol. 12579). Springer. https://doi.org/10.1007/978-3-030-65411-5_17

Detecting Covert Cryptomining Using HPC

Ankit Gangwal[1](✉), Samuele Giuliano Piazzetta[2], Gianluca Lain[2], and Mauro Conti[3]

[1] TU Delft, Delft, The Netherlands
a.gangwal@tudelft.nl
[2] ETH Zürich, Zürich, Switzerland
{spiazzetta,gilain}@student.ethz.ch
[3] University of Padua, Padua, Italy
conti@math.unipd.it

Abstract. Cybercriminals have been exploiting cryptocurrencies to commit various unique financial frauds. Covert cryptomining - which is defined as an unauthorized harnessing of victims' computational resources to mine cryptocurrencies - is one of the prevalent ways nowadays used by cybercriminals to earn financial benefits. Such exploitation of resources causes financial losses to the victims.

In this paper, we present our efficient approach to detect covert cryptomining on users' machine. Our solution is a generic solution that, unlike currently available solutions to detect covert cryptomining, is not tailored to a specific cryptocurrency or a particular form of cryptomining. In particular, we focus on the core mining algorithms and utilize Hardware Performance Counters (HPC) to create clean signatures that grasp the execution pattern of these algorithms on a processor. We built a complete implementation of our solution employing advanced machine learning techniques. We evaluated our methodology on two different processors through an exhaustive set of experiments. In our experiments, we considered all the cryptocurrencies mined by the top-10 mining pools, which collectively represent the largest share of the cryptomining market. Our results show that our classifier can achieve a near-perfect classification with samples of length as low as five seconds. Due to its robust and practical design, our solution can even adapt to zero-day cryptocurrencies. Finally, we believe our solution is scalable and can be deployed to tackle the uprising problem of covert cryptomining.

Keywords: Cryptocurrency · Machine learning · Mining · Profiling

1 Introduction

Cryptomining, or simply mining, is a process of validating and adding new transaction in the blockchain digital ledger for various cryptocurrency. It is an essential process to keep most of the cryptocurrencies running. Typically, mining is a

S. Krenn et al. (Eds.): CANS 2020, LNCS 12579, pp. 344–364, 2020.
https://doi.org/10.1007/978-3-030-65411-5_17

resource-intensive process that continuously performs heavy computations. Upon successful mining, miners receive newly generated cryptocoins as their remuneration. Usually, newer cryptocurrencies tend to pay a higher reward. Some cryptocurrencies, such as Monero, make mining feasible on the web-browsers that enable even layman users to participate in mining.

After the success of Bitcoin [40], many alternative cryptocurrencies (altcoins) have been introduced to the market. At the time of writing, there are over 2000 active cryptocurrencies [2]. The massive number of cryptocurrencies raises an enormous demand for mining. This demand continues to remain huge because mining, as mentioned before, is an inevitable operation to keep these virtual currency systems running. Such an immense demand for mining has attracted cybercriminals [7,18] to earn financial gains, who have already been exploiting cryptocurrencies to perform several types of financial crimes, e.g., ransomware [29].

Motivation: A genuine miner has to make an investment in hardware and bear the significant cost of electricity to run the mining hardware as well as cooling facilities [14]. Nevertheless, mining is not beneficial on personal expenditure (mainly, on electricity) unless mining is performed with specialized hardware [16]. However, mining can be very profitable if it is performed with "stolen" resources, e.g., through covert cryptomining, or simply cryptojacking. Cryptojacking is defined as an unauthorized use of the computing resources on a computer, tablet, mobile phone, or connected home device to mine cryptocurrencies.

Cybercriminals have made several ingenious attempts to spread cryptojackers in the form of malware [20], malicious browser extensions [12], etc.. by exploiting vulnerability [17], compromising third-party plug-ins [19], maneuvering misconfigurations [11], taking advantage of web-based hosting service [13], and so on. To evade intrinsic detection techniques (e.g., processor's usage), some cryptojackers suspend their execution when the victim is using the computer [31], use "pop-under" windows to keep mining for a comparatively longer duration [8], and utilize legitimate processes of the operating system to mine [28]. Moreover, merely monitoring CPU load, etc.. is an ineffective strategy because of both false positives and false negatives [37].

To further aggravate the situation, cryptocurrency mining service (e.g., Coinhive [1], Crypto-Loot [3]) easily integrate into websites to monetize the computational power of their visitors. In fact, cryptojacking attacks exceeded ransomware attacks in 2018 and affected five times more systems as compared to ransomware [25]. According to Symantec's report [10], almost double cryptominers were detected on consumer machines as compared to enterprise machines between October 2017 and February 2018 while the same volume of cryptominers was detected on consumer and enterprise machines between March 2018 and July 2018. Kaspersky's report [15] shows that the total number of internet users who encountered cryptominers rose from 1.9 million in 2016–2017 to 2.7 million in 2017–2018. IBM X-Force Threat Intelligence Index 2019 [23] estimates that cryptojacking attacks increased by more than 4-times (∼450%) from Q1 2018 to Q4 2018. SonicWall researchers [24] reported that cryptojacking attackers made 52.7 million cryptojacking hits during the first half of 2019. Such exploitation of the computational resources causes financial damage - primarily in the form of increased[1] electricity bills - to the victims, who often discover the misuse when the damage has already been done.

On another side, the current state of cryptomining has been consuming a vast amount of energy. As a representative example, Bitcoin Energy Consumption Index was created to provide insight into this amount with respect to Bitcoin, Bitcoin network consumes electricity close to the total demand by Iraq, and a single Bitcoin transaction requires nearly 2.7 times the electrical energy consumed by 100,000 transactions on the VISA network [9]. Moreover, a recent study [39] has suggested that "Bitcoin usage could alone produce enough CO_2 emissions to push warming above 2 °C within less than three decades." The current situation would further worsen with illegal/unauthorized/covert cryptomining. Finally, the abundance of the active cryptocurrencies raises the demand for a generic solution to detect covert cryptomining that does not focus on a particular cryptocurrency.

Contribution: In this paper, we focus on detecting covert cryptomining on users' machine. The major contributions of this paper are as follows:

1. We propose an efficient approach to detect covert cryptomining. In particular, our approach uses HPC to profile the core of the mining process, i.e., the mining algorithms, on a given processor to accurately identify cryptomining in real-time. We designed our solution to be a generic one, i.e., it is not tailored to a particular cryptocurrency or a specific form of cryptomining.
2. We exhaustively assess the quality of our proposed approach. To this end, we designed six different experiments: (1) *binary* classification; (2) *currency* classification; (3) *nested* classification; (4) *sample length*; (5) *feature relevance*; and (6) *unseen miner programs.* For a thorough evaluation, we considered eleven distinct cryptocurrencies in our experiments. Our results show that our classifier can accurately classify cryptomining activities.
3. In the spirit of reproducible research, we make our collected datasets and the code publicly available[2].

Organization: The remainder of this paper is organized as follows. Section 2 presents a summary of the related works. We explain our system's architecture in Sect. 3 and discuss its evaluation in Sect. 4. Section 5 addresses the potential limitations of our solution. Finally, Sect. 6 concludes the paper.

2 Related Works

HPC are special-purpose registers in modern microprocessors that count and store hardware-related activities. These activities are commonly referred to as

[1] A machine consistently performs heavy computations while it does cryptomining, which, in turn, continuously draws electricity.

[2] spritz.math.unipd.it/projects/cryptojackers/.

hardware *events*[3]. HPC are often used to conduct low-level performance analysis and tuning. HPC-based monitoring has very low-performance overhead, which makes it suitable even for latency-sensitive systems. Several works have shown the effectiveness of using HPC to detect generic malware [32,46,48], kernel-level rootkits [47], side-channel attacks [27], unauthorized firmware modifications [45], etc.

A general-purpose process classification may distinguish a browser application from a media player or one browser application from another browser application. In the former case, the nature of the applications is different while both the applications in the latter case have the same nature and perform the same operation of rendering pages. Cryptominers have the same nature (of mining), but they essentially perform very different underlying operations due to different proof-of-works, and they also require different compute resources (e.g., BTC[4] mining is processor-oriented while XMR mining is memory-oriented). Hence, a comparison of our work with the general-purpose process classification methods falls out of the scope of this paper.

On another side, there are limited number of works on detecting cryptomining. Bonneau et al. [26] discuss open research challenges of various cryptocurrencies and their mining. Huang et al. [36] present a systematic study of Bitcoin mining malware and have shown that modern botnets tend to do illegal cryptomining. Gangwal et al. [33] use magnetic side-channel to detect cryptomining. Other works [37,38,41,42,44] focus particularly on browser-based mining. However, only a limited number of cryptocurrencies can be mined in the web-browsers. MineGuard [43] focuses on detecting cryptomining operations in the cloud infrastructure.

Our work is different from the state-of-the-art on the following dimensions: (1) our proposed solution is a generic solution that is not tailored to a particular cryptocurrency or a specific form (e.g., browser-based) of cryptomining on computers; and (2) we tested our solution against all the cryptocurrencies mined by the top-10 mining pools, which collectively represent the largest portion of the cryptomining business.

3 System Architecture

We elucidate the key concept behind our approach in Sect. 3.1, our data collection phase in Sect. 3.2, selection of cryptocurrencies in Sect. 3.3, and our classifier's design in Sect. 3.4.

3.1 Fundamental Intuition of Our Approach

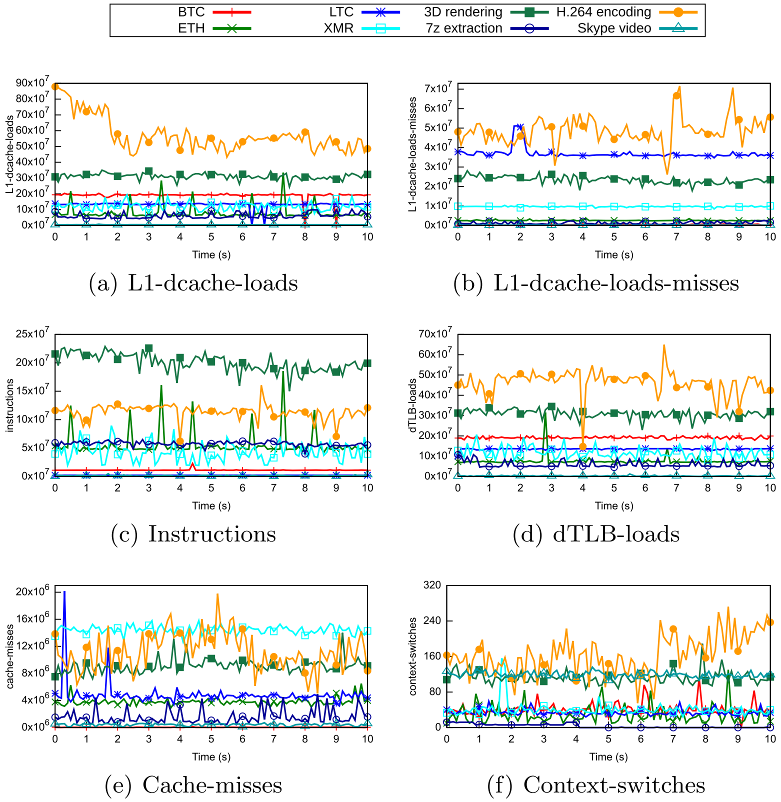
The task of cryptomining requires a miner to run the core Proof-of-Work (PoW[5]) algorithm repetitively to solve the cryptographic puzzle. At a coarse-grained

[3] An *event* is defined as a countable activity, action, or occurrence on a device.

[4] To refer to different cryptocurrencies, we use their standard ticker symbol. See Table 3 for acronyms and their corresponding cryptocurrencies.

[5] We use the term "PoW" to represent different consensus algorithms.

level, some PoW algorithms are processor-oriented (e.g., BTC) while some are memory-oriented (e.g., XMR) due to their underlying design. At a fine-grained level, each PoW algorithm has its own unique mathematical/logical computations (or, in other words, the sequence of operations). Thus, each algorithm upon execution affects some specific *events* more as compared to other *events* on the processor. Consequently, when an algorithm is executed several times repetitively, the "more" affected *events* outnumber the other - relatively under affected - *events*. It means that a discernible signature can be built using the relevant *events* for a PoW algorithm. As a representative example, Fig. 1 depicts the variation in *events* while mining different cryptocurrencies and performing some common user-tasks. LTC, for instance, shows a more erratic pattern in cache-misses as compared to the other *events* that are affected during LTC mining. On the other hand, a Skype video call has more disparity in context-switches.



(a) L1-dcache-loads

(b) L1-dcache-loads-misses

(c) Instructions

(d) dTLB-loads

(e) Cache-misses

(f) Context-switches

Fig. 1. A representative example of variation in *events* while mining different cryptocurrencies and performing some common user-tasks. HPC were polled every 100 ms. The line-points in the graphs do not represent data points and are merely used to make lines distinguishable.

In practice, there is a finite number of PoW algorithms upon which cryptocurrencies are established. So, we concentrate on the mining algorithms instead of individual currency in our solution. To this end, we use supervised machine learning (cf. Sect. 3.4) to construct signatures and build our classifier.

On another side, an adversary may attempt to circumvent such signature-based detection in the following ways: (1) by controlling/limiting the mining; or (2) by neutralizing the signatures. Limiting the mining would reduce the hashing rate, which would indeed make the mining less profitable. Whereas, to neutralize the signatures, the adversary has to succeed in two main hurdles. First, the adversary must have to find those computation(s) that only changes those *events* that are unrelated to the PoW algorithm. Second, the adversary must have to run these computation(s) in parallel to the PoW algorithm, which would again hamper the hashing rate, and thus the profit. In this work, we make a practical assumption that the attacker wants to maximize the profit and does not want to lose the computation cycles (hashing rate).

3.2 Data Collection

To better explain our work, we first describe what data we collect and how we collect it. We used the *perf* [5] tool to profile the processor's *events* using HPC. In particular, we focus on hardware[6] *events* (e.g., branch-misses), software[7] *events* (e.g., page-faults), and hardware cache[8] *events* (e.g., cache-misses) on CPU as the mining processes - depending on their design - require different type of resources. We profiled each program of both positive (mining) and negative (non-mining) class individually and collected a total 50 samples per program. Each sample consists of recordings of 28 *events* (described in Table 1) for 30 s with a sampling rate 10 Hz, which means that each sample comprises 300 readings of 28 *events*, i.e., 8400 readings. To obtain clean signatures: (1) we profiled each program in its stable stage, i.e., omitting the bootstrapping phase; and (2) restarted the system to remove any trace of the previous sample.

For the positive class, we profiled a total of 11 cryptocurrencies discussed in Sect. 3.3. As the representatives of negative class, we chose: 3D rendering; *7z* archive extraction of *tar.gz* files; H.264 video encoding of raw video; solving *mqueens* problem; Nanoscale Molecular Dynamics (NAMD) simulation; *Netflix* movie playback; execution of Random Forest (RF) machine learning algorithm; *Skype* video calls; *stress-ng* [6] stress test with CPU, memory, I/O, and disk workers together; playing *Team Fortress 2* game; and Visual Molecular Dynamics (VMD) modeling and visualization. It is worth mentioning that these user-tasks represent medium to high resource-intensive tasks.

We used two different systems to build our dataset for the experiments. The configuration of these systems are as follows: (1) *S1*, a laptop with an Intel Core i7-7500U @ 2.70 GHz (1 socket × 2 cores × 2 threads = 4 logical compute

[6] Basic events, measured by Performance Monitoring Units (PMU).

[7] Measurable by kernel counters.

[8] Data- and instruction-cache hardware events.

Table 1. The *events* that we monitor using HPC. Here, HW = hardware, SW = software, and HC = hardware cache *event*.

Event	**Type**	**Description**	**Event**	**Type**	**Description**
branch-instructions	HW	N. of retired branch instructions.	iTLB-load-misses	HC	N. of instruction fetches that missed instruction TLB.
branch-load-misses	HW	N. of Branch load misses.	iTLB-loads	HC	N. of instruction fetches that queried instruction TLB.
branch-loads	HW	N. of Branch load accesses.	L1-dcache-load-misses	HC	N. of load misses at L1 data cache.
branch-misses	HW	N. of mispredicted branch instructions.	L1-dcache-loads	HC	N. of loads at L1 data cache.
bus-cycles	HW	N. of bus cycles, which can be different from total cycles.	L1-dcache-stores	HC	N. of stores at L1 data cache.
cache-misses	HC	N. of cache misses.	LLC-load-misses	HC	N. of load misses at the last level cache.
cache-references	HC	N. of cache accesses.	LLC-loads	HC	N. of loads at the last level cache.
context-switches	SW	N. of context switches.	LLC-store-misses	HC	N. of store misses at the last level cache.
cpu-migrations	SW	N. of times the process has migrated.	LLC-stores	HC	N. of stores at the last level cache.
dTLB-load-misses	HC	N. of load misses at data TLB.	mem-loads	HC	N. of memory loads.
dTLB-loads	HC	N. of load hits at data TLB.	mem-stores	HC	N. of memory stores.
dTLB-store-misses	HC	N. of store misses at data TLB.	page-faults	SW	N. of page faults.
dTLB-stores	HC	N. of store hits at data TLB.	ref-cycles	HW	N. of total cycles; not affected by CPU frequency scaling.
instructions	HW	N. of retired instructions.	task-clock	SW	The clock count specific to the task that is running.

resources) processor, 8 GB memory, 512 GB SSD storage, NVIDIA GeForce 940MX 2 GB dedicated graphic card, Linux kernel 4.14 and (2) *S2*, a laptop with an Intel Core i7-8550U @ 1.80 GHz (1 socket × 2 cores × 4 threads = 8 logical compute resources) processor, 16 GB memory, 512 GB SSD storage, Linux kernel 4.14.

All miner programs and the *perf* tool were launched in *user*-mode. Even though we did not use any system-level privileges, we believe that using *root* permissions for defense against cryptojacking is reasonable. The *perf* tool allows us to create per-process profile using *PID*. It is worth emphasizing that even though the dataset has been accumulated in a controlled setup, our experiments (discussed in Sect. 4) well simulate real-world scenario, where samples are collected in the real-time.

3.3 Cryptocurrencies and Miners

The probability of solving the cryptographic puzzle during mining is directly proportional to the miner's computational power/resources. Consequently, the miners pool their resources to combine their hashing power with an aim to consistently earn a portion of the block reward by solving blocks quickly. Typically, the mining pools are characterized by their hashing power. Table 2 shows the top-10 mining pools [21] and the cryptocurrencies mined by them. These ten mining pools collectively constitutes the biggest share (84% during Q1 2019) of the cryptomining business.

Table 2. Cryptocurrencies mined by the top-10 mining pools

N.	Mining pool	Cryptocurrency															
		BCD	BCH	BTC	BTM	DASH	DCR	ETC	ETH	LTC	SBTC	SC	UBTC	XMC	XMR	XZC	ZEC
1	BTC.com	✗	✓	✓	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗	✗	✗
2	AntPool	✗	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✗	✓	✗	✗	✓
3	ViaBTC	✗	✓	✓	✗	✓	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✓
4	SlushPool	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
5	F2Pool	✗	✗	✓	✗	✓	✓	✓	✓	✓	✗	✓	✗	✓	✓	✓	✓
6	BTC.top	✗	✓	✓	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗
7	Bitclub.network	✗	✗	✓	✗	✓	✗	✓	✓	✗	✗	✗	✗	✗	✓	✗	✓
8	BTCC	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗
9	BitFury	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
10	BW.com	✗	✗	✓	✗	✗	✗	✓	✓	✓	✗	✓	✓	✗	✗	✗	✗

We considered all the cryptocurrencies mentioned in the Table 2 in our experiments. We used open-source miner programs to mine these cryptocurrencies. Each miner program was configured to mine with public mining pools and to utilize all available the CPUs present on the system. At the time of our experiments, the miner program for SC was not able to mine using only the CPU. Hence, we excluded SC from our experiments. To compensate SC, we included QRK whose mining algorithm - in contrast to other cryptocurrencies - uses multiple hashing algorithms. Table 3 shows the mining algorithm of different cryptocurrencies and the CPU miners that we used.

Table 3. Mining algorithm and CPU miner for different cryptocurrencies

Cryptocurrency	Mining algorithm	CPU miner
Bitcoin Diamond (BCD)	X13	cpuminer-opt 3.8.8.1
Bitcoin Cash (BCH), Bitcoin (BTC), SuperBitcoin (SBTC), UnitedBitcoin (UBTC)	SHA-256	cpuminer-multi 1.3.4
Bytom (BTM)	Tensority	bytom-wallet-desktop 1.0.2
Dash (DASH)	X11	cpuminer-multi 1.3.4
Decred (DCR)	Blake256-r14	cpuminer-multi 1.3.4
Ethereum Classic (ETC), Ethereum (ETH)	Ethash (Modified Dagger-Hashimoto)	geth 1.7.3
Litecoin (LTC)	scrypt	cpuminer-multi 1.3.4
Quark (QRK)	BLAKE + Grϕstl + Blue Midnight Wish + JH + Keccak (SHA-3) + Skein	cpuminer-multi 1.3.4
Siacoin (SC)	BLAKE2b	gominer 0.6
Monero-Classic (XMC), Monero (XMR)	CryptoNight	cpuminer-multi 1.3.4
Zcoin (XZC)	Lyra2z	cpuminer-opt 3.8.8.1
Zcash (ZEC)	Equihash	Nicehash nheqminer 0.3a

Since our approach focuses on the underlying core PoW algorithm, we considered one currency for every mining algorithm mentioned in Table 3 and excluded BCH, SBTC, UBTC, ETC, and XMC in our study. As the proof-of-concept implementation, we considered only CPU-based miner programs because each computer has at least one CPU, which cryptojackers can harness to mine.

3.4 Classifier Design

In this section, we elucidate the design of our classification methodology. Algorithm 1 describes the pipeline of our classifier. Our supervised classification algorithm begins with splitting the base-dataset of 1100 samples (2 classes × 11 instances × 50 samples) into 90–10% stratified train-test sets, denoted as *raw_train_set* and *raw_test_set*. Then, these subsets are processed as follows:

Algorithm 1. Pseudo code for our supervised classification.

```
for each run i from 1 to 10 do
    Create raw_train_set and raw_test_set by 90–10% stratified partitioning.
    Data preprocessing
    • Replace NaN values from raw_train_set and raw_test_set with arithmetic mean
    of the considered event.
    Feature engineering
    • train_set := Extract_feature(raw_train_set)
    • test_set := Extract_feature(raw_test_set)
    Feature scaling
    • scaler := StandardScaler()
    • scaler.fit(train_set)                                ▷Fit scaler on train_set
    • scaler.transform(train_set)
    • scaler.transform(test_set)
    Feature selection
    • Compute features' importance with forests of trees on train_set and select the
    most relevant features.
    Training
    • Learn the model parameters for the given classifier (RF/SVM) on the training
    set using grid search with 5-fold stratified CV.
    Predict/classify the test_set.
end for
```

1. *Data preprocessing:* The first step of any machine learning-based classification is to process the raw datasets to fix any missing value. Since each event we monitor returns a numerical value, we replace the missing values, if any, with the arithmetic mean of the respective event.
2. *Feature engineering:* In this step, we obtain features that can be used to train a machine learning model for our prediction problem. Here, we compute 12 statistical functions (listed in Table 4) for every event. This step converts each sample consisting of 300 readings (rows) × 28 *events* (columns) to a single

row of 336 (28 *events* × 12 features) data-points. The features extracted in this phase, hereinafter referred to as *train_set* and *test_set*, are used for the subsequent stages.

Table 4. The statistical functions that we used for our feature engineering phase

0.2, 0.4, 0.6, and 0.8 quantile	1, 2, and 3 sigma	Skewness
Arithmetic and geometric mean	Kurtosis	Variance

3. *Feature scaling:* It is an essential step to eliminate the influence of large-valued features because features with larger magnitude can dominate the objective function, and thus, deterring an estimator to learn from other features correctly. Hence, we standardize features using standard scaler, which removes the mean and scale the features to unit variance.
4. *Feature selection:* In machine learning, feature selection or dimensionality reduction is the process of selecting a subset of relevant features that are used in model construction. It aims to improve estimators' accuracy as well as to boost their performance on high-dimensional datasets. To do so, we calculate the importance of features using *forests of trees* [22] and select the most relevant features.
5. *Training:* The training phase consists of learning the model parameters for the given classifier on the training set, i.e., *train_set*. Given the nature of the problem, we resort to supervised machine learning procedures. In particular, we employed two of the most successful machine learning methods for classification, namely Random Forest (RF) [34] and Support Vector Machine (SVM) [30].
   For model selection, we use grid search with 5-fold Cross Validation (CV). The validated hyper-parameters for RF and SVM are shown in Table 10 and Table 11, respectively in Appendix A. We chose standard range of values for the hyper-parameters [35].
6. *Prediction:* Finally, prediction is made on *test_set*.

The process is repeated ten times for a given experiment and the final results are computed over these ten runs.

4 Evaluation

We throughly evaluated our approach by performing an exhaustive set of experiments. We performed the following six different experiments: (1) *binary* classification (Sect. 4.1); (2) *currency* classification (Sect. 4.2); (3) *nested* classification (Sect. 4.3); (4) *sample length* (Sect. 4.4); (5) *feature relevance* (Sect. 4.5); and (6) *unseen miner programs* (Sect. 4.6). Table 5 describes the sample distribution in our base-dataset for each system, i.e., *S1* and *S2*. Here, sub-classes of the mining task refer to the cryptocurrencies (discussed in Sect. 3.3) while sub-classes of the non-mining task refer to the actual user-tasks that belong to the negative class (mentioned in Sect. 3.2). We use the entire base-dataset (1100 samples per system) for each experiment, unless otherwise stated in an experiment.

Table 5. Dataset: name of the task, sub-classes per task, samples per sub-class, and total samples per task for each system

Task	Sub-classes per task	Samples per sub-class	Total samples per task
Mining	11	50	550
Non-mining	11	50	550

We evaluated our classifier using standard classification metrics: Accuracy, Precision, Recall, and F_1 score. To increase the confidence in our results, we report the mean and the margin of error for the results with 95% confidence interval from ten runs of each experiment for each of the evaluation metric. We use $(\cdot)$ to indicate the best result for the metric and report the results as $mean \pm margin\ of\ error$.

4.1 Binary Classification

Our main goal is to identify whether a given instance represents the mining task or not. Hence, in this experiment, the label of each sample was defined as the positive or negative class, accordingly. Table 6 presents the results of the *binary* classification using both RF and SVM.

Table 6. Results for binary classification

System	Method	Accuracy	Precision	Recall	F1
S1	RF	$1.000 \pm 0.000\cdot$	$1.000 \pm 0.000\cdot$	$1.000 \pm 0.000\cdot$	$1.000 \pm 0.000\cdot$
	SVM	0.999 ± 0.002	0.999 ± 0.002	0.999 ± 0.002	0.999 ± 0.002
S2	RF	$0.999 \pm 0.002\cdot$	$0.999 \pm 0.002\cdot$	$0.999 \pm 0.002\cdot$	$0.999 \pm 0.002\cdot$
	SVM	0.990 ± 0.018	0.991 ± 0.016	0.990 ± 0.018	0.990 ± 0.018

Both the RF and SVM yielded superior performance. However, RF performed better than SVM on both the systems; the possible reason for the difference in classifiers' performance is their underlying designs - RF and SVM characterize their decision boundaries differently and also handle the outliers present in the dataset differently. On another side, the minute variations in the performance of a given classifiers across *S1* and *S2* are natural and expected; mainly due to distinct dataset and data stratification. For the sake of brevity, we report the results only for RF for the subsequent experiments. We also present the details of parameters selected by grid search in Appendix B.

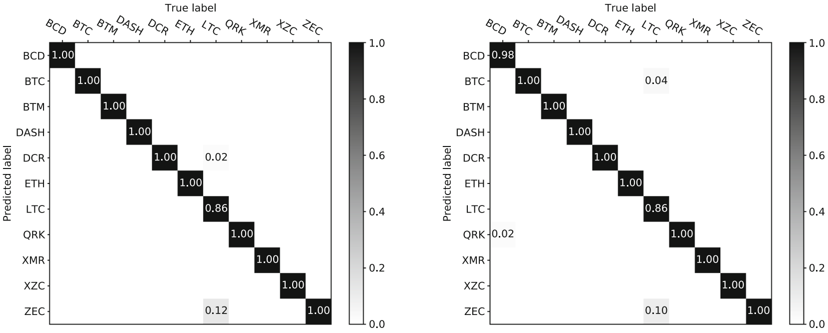
4.2 Currency Classification

The aim of this experiment is to understand the difficulty level of classification among various cryptocurrencies. Therefore, the input dataset for this experiment contained instances only of the cryptocurrencies. Table 7 lists the results of the *currency* classification.

Table 7. Results for currency classification

System	Accuracy	Precision	Recall	F1
S1	0.987 ± 0.017	0.992 ± 0.011	0.988 ± 0.016	0.985 ± 0.020
S2	0.986 ± 0.018	0.981 ± 0.027	0.985 ± 0.018	0.982 ± 0.024

Figure 2 depicts the confusion matrices for the classification among various cryptocurrencies to provide a better perception of the results. Here, Fig. 2(a) and Fig. 2(b) correspond to *S1* and *S2*, respectively. The confusion matrices are drawn using the aggregate results from all the ten runs. *Currency* classification is a multi-class classification problem, and some cryptocurrencies were misclassified among each other (see Fig. 2). Hence, the results are slightly lower than that of the *binary* classification.



(a) *S1* (b) *S2*

Fig. 2. Confusion matrix for classification among various cryptocurrencies

4.3 Nested Classification

This experiment represents a simulation of a real-world scenario. Here, we first classify whether a given instance belongs to the positive class. If so, we identify the cryptocurrency it belongs to. Essentially, *nested* classification is equivalent to performing *currency* classification on the instances classified as positive in the *binary* classification.

Table 8 shows the results of the *nested* classification. In the worst case, we expect the outcome of this experiment to be lower than that of the *binary* classification and *currency* classification together because a crucial aspect of such staged classification is that an error made in the prediction during the primary stage influences the subsequent stage; the results for *S1* shows this phenomenon. However, in a common scenario, the expected outcome of this experiment would be between the results for the *binary* classification and *currency* classification; the results for *S2* shows this effect.

Table 8. Results for nested classification

System	Accuracy	Precision	Recall	F1
S1	0.973 ± 0.020	0.972 ± 0.026	0.972 ± 0.020	0.967 ± 0.026
S2	0.996 ± 0.007	0.997 ± 0.006	0.996 ± 0.008	0.996 ± 0.008

4.4 Sample Length

The objective of this experiment is to understand the effect of length of the samples. For deployment in the real-world scenario, any solution - apart from being accurate - must be able to detect cryptojackers rapidly. To this end, we performed the *binary* classification of samples of a length of 5, 10, 15, 20, 25, and 30 s, each in separate experiments. It is worth mentioning that we used samples of identical length for both the training and testing. Figure 3 shows the F_1 score when using samples of different length.

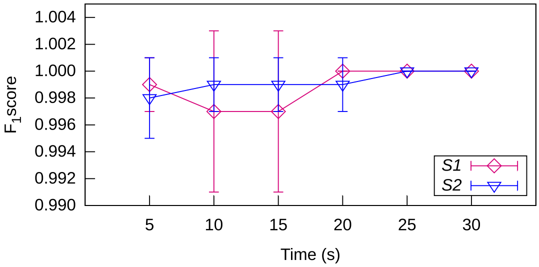


Fig. 3. F_1 score for different sample lengths (whiskers represent margin of error)

As explained in Sect. 3.1, the task of mining is to repeatedly execute the core PoW algorithm. Hence, even samples of shorter length can grasp the signature. As shown in Fig. 3, our system can achieve high performance with samples of 5 s. The dip in the curve for *S1* corresponds to the thousandths digit of the F_1 score. For the sake of brevity, we omitted the results for sample shorter than five seconds and only focus on the required minimum sample length to attain high performance with our solution.

4.5 Feature Relevance

Next, we focus on our feature selection process (mentioned in Sect. 3.4). After calculating the importance of features, we sorted them in ascending order of their importance and selected the first-Ψ% features to do the *binary* classification. The key idea here is to identify the lower-limit of (even less important) features required to obtain the best performance. Figure 4 depicts the F_1 score when using first-Ψ% features.

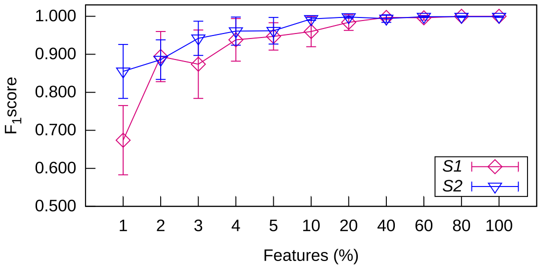


Fig. 4. F_1 score for first-Ψ% features (whiskers represent margin of error)

Since the features are sorted in the ascending order of their importance, we begin with the feature with lowest significance. Intuitively, including important features further improves the classification process. As shown in the Fig. 4, our classifier attains high performance on both the systems using only the first-40% (less relevant) features, which verifies/approves our feature engineering and selection process.

4.6 Unseen Miner Programs

There can be several different miner-programs available to mine a given cryptocurrency. These programs come from different developers/sources. Consequently, there can be some variations in the behavior of the miner-program itself, e.g., in the code section before/after the PoW function or handling (on the programming-side) a correct nonce found while mining. The reason is that they are developed by different developers, which intuitively will cause variations.

Training the model for each program may not be feasible for a variety of reasons. Hence, to investigate the effectiveness of our approach in such a situation, we set up this experiment. Here, we selected the *binary* classification as the target where the samples from all the mining and non-mining tasks were labeled as the positive or negative class, respectively. However, we chose two additional miner programs for BTC, namely, BFGMiner 5.5 and cgminer 4.10. We collected additional 50 samples each for BFGMiner 5.5 and cgminer 4.10 on both *S1* and *S2* separately. In the training phase, we used samples from one of the three miner programs for BTC. On the contrary, we used samples from one of the other two miner programs for BTC during the testing phase. Table 9 presents the results of classifying samples from the miner programs that were unseen in the training phase.

Table 9. Results for unseen miner programs

System	Task	Accuracy	Precision	Recall	F1
S1	α_β	0.997 ± 0.006	0.997 ± 0.006	0.997 ± 0.006	0.997 ± 0.006
	α_γ	0.998 ± 0.005	1.000 ± 0.000	0.997 ± 0.006	0.998 ± 0.004
	β_α	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
	β_γ	0.999 ± 0.001	0.999 ± 0.002	0.999 ± 0.002	0.999 ± 0.002
	γ_α	0.999 ± 0.002	0.999 ± 0.002	0.999 ± 0.002	0.999 ± 0.002
	γ_β	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
S2	α_β	0.999 ± 0.001	0.999 ± 0.002	0.999 ± 0.002	0.999 ± 0.002
	α_γ	0.998 ± 0.002	0.997 ± 0.003	0.997 ± 0.003	0.997 ± 0.003
	β_α	0.999 ± 0.002	0.998 ± 0.003	0.998 ± 0.003	0.998 ± 0.003
	β_γ	0.999 ± 0.001	0.999 ± 0.002	0.999 ± 0.002	0.999 ± 0.002
	γ_α	0.999 ± 0.001	0.999 ± 0.002	0.999 ± 0.002	0.999 ± 0.002
	γ_β	0.999 ± 0.001	0.999 ± 0.002	0.999 ± 0.002	0.999 ± 0.002

The notation X_Y means that the training was done with the samples from X while the testing was done with the sample from Y for BTC. Here, α = cpuminer-multi 1.3.4, β = BFGMiner 5.5, γ = cgminer 4.10. It is important to mention that these results are for the classification of all the mining and non-mining tasks with BTC being trained and tested upon samples from different programs. As discussed in Sect. 3.1, the miners have to execute the same core PoW algorithm for a given cryptocurrency. Hence, samples from different miner programs for a cryptocurrency retain the same signatures, which is reflected in our results.

Cross-Platform Classification: Next, we evaluate the transferability of the profiles built by our approach. We perform binary classification with additional samples from *S1'* (a system with the same processor as *S1*) and *S2'* (a system with the same processor as *S2*), and found that: (1) the profile of an algorithm on a given processor can be used with the help of machine learning technique to classify samples from another system with the same processor and (2) on the contrary, the profile of an algorithm on one processor is not useful to perform classification of samples from another processor.

5 Limitations

In this section, we address the potential limitation of our proposed approach.

5.1 Zero-Day Cryptocurrencies

A zero-day cryptocurrency would be a currency that uses a completely new or custom PoW algorithm that was never seen before. As a matter of fact, for a cryptocurrency to obtain market value: (1) its core-network should be supported by miners/pools; and (2) its PoW algorithm must be accepted by the crypto-community and tested mathematically for its robustness. Therefore, the PoW algorithm for a new cryptocurrency would become public by the time it gets ready for mining, which would give us sufficient time to capture this new cryptocurrency's signature and to train our model.

Importantly, miners prefer to mine cryptocurrencies that are more profitable and avoid hashing the less rewarding ones. As it happens to be, more profitable cryptocurrencies are indeed popular and their PoW algorithms are certainly known to the public. In our experiments, we considered all the popular cryptocurrencies, and our results (presented in Sect. 4) demonstrate the high quality of our proposed approach along various dimensions.

5.2 Scalability

The key concept of our approach is to profile the behavior of a processor's *events* for mining algorithms. Since there are only a finite number of CPUs/GPUs, procuring their signature is only a matter of data collection. It might appear as a ponderous job and may be seen as a limitation of our work. But, once it is accomplished for the available CPUs/GPUs, maintaining it is relatively simpler as merely a limited number of CPUs/GPUs are released over a period of time.

5.3 Process Selection

As mentioned in Sect. 3.2, our system requires per program/process-based recording of HPC for different *events* as the input to the classifier. In practice, several processes run in the system. Hence, monitoring each process may consume time and can be seen as a limitation of our work. However, as shown in Fig. 3, our system can achieve high performance even with samples of 5 s. On another side, the miner programs attempt to use all the available resources. Therefore, an initial sorting/filtering of processes based on their resource usage can help to boost the detection process in real-time.

5.4 Restricted Mining

A mining strategy to evade detection from our proposed methodology can be *restricted mining* that aims to change the footprint of the mining process. The essence here is that the miner program/process can be modified to perform arbitrary operations during mining. But, such maneuvers would directly affect the hashing rate and consequently the profits of mining; making the task of mining less appealing. Nevertheless, like any signature-based detection technique, it may be seen as a limitation of our work.

6 Conclusion and Future Works

Cybercriminals have developed several proficient ways to exploit cryptocurrencies with an aim to commit many unconventional financial frauds. Covert cryptomining is one of the most recent means to monetize the computational power of the victims. In this paper, we present our efficient methodology to identify covert cryptomining on users' machine. Our solution has a broader scope - compared to the solution that are tailored to a particular cryptocurrency or a specific form (e.g., browser-based) of cryptomining on computers - as it targets the core PoW algorithms and uses the low-performance overhead HPC that are present in modern processors to create discernible signatures. We tested our generic approach against a set of rigorous experiments that include eleven distinct cryptocurrencies. We found that our classifier attains high performance even with short samples of five seconds.

We believe that our approach is valid to distinguish GPU-based miners because dedicated profiling tools, such as the *nvprof* [4] tool for NVIDIA GPUs, allow us to monitor GPU *events*. Apart from most of the standard *events* found on CPUs, GPUs have several dedicated *events* that can assist in creating unique signatures for GPUs. Nevertheless, we keep such investigation as part of our future work. We will also perform our experiments with a larger set of systems (CPUs) to observe the generalization of our approach. We also hope to release a desktop application for run-time identification of covert cryptomining.

Appendix A Validated Hyper-parameters

The validated hyper-parameters for RF and SVM are shown in Table 10 and Table 11, respectively.

Table 10. Hyper-parameters validated for RF classifier

Parameter	Validated values	Effect on the model
n_estimators	{10, 25, 50, 75, 100, 125, 150}	Number of trees use in the ensemble
max_depth	[2, ∞)	Maximum depth of the trees
max_features	'auto', 'log2'	Number of features to consider when looking for the best split
split_criterion	*gini, entropy*	Criterion used to split a node in a decision tree
bootstrap	*true, false*	Bootstrap Aggregation (a.k.a. bagging) is a technique that reduces model variances (overfitting) and improves the outcome of learning on limited sample or unstable datasets
random_state	10	The seed used by the random number generator

Table 11. Hyper-parameters validated for SVM classifier

Parameter	Validated values	Effect on the model
kernel	'rbf', 'poly', 'sigmoid'	Specifies the kernel type to be used in the algorithm
C	$[10^{-3}, 10^{5}]$	Regularization parameter that controls the trade-off between the achieving a low training error and a low testing error that is the ability to generalize your classifier to unseen data
γ	'auto', $[10^{-7}, 10^{3}]$	Shape parameter of the RBF kernel which defines how an example influence in the final classification
degree	default=3	Degree of the polynomial kernel function ('poly'). Ignored by all other kernels
random_state	10	The seed of the pseudo random number generator used when shuffling the data for probability estimates

Appendix B Parameters selected by grid search

Here, we list the frequency of parameter-values selected by grid search over ten-runs of different experiments. Table 12 corresponds to *binary* classification experiment with SVM while Table 13 corresponds to *binary*, *currency*, and *full* classification experiments with RF for both *S1* and *S2*.

Table 12. *Binary* classification with SVM

Parameter	Value	N. of times selected on *S1*	N. of times selected on *S2*
kernel	'rbf'	7	6
	'poly'	1	0
	'sigmoid'	2	4
C	0.01	1	0
	0.1	0	4
	1	1	1
	10	3	2
	100	2	2
	1000	3	1
γ	0.0001	2	1
	0.001	1	4
	0.01	2	1
	0.1	2	0
	'auto'	3	4

Table 13. Different classifications with RF

		Binary classification		*Currency* classification		*Full* classification	
Parameter	Value	N. of times selected on *S1*	N. of times selected on *S2*	N. of times selected on *S1*	N. of times selected on *S2*	N. of times selected on *S1*	N. of times selected on *S2*
bootstrap	*true*	10	10	10	10	10	10
	false	0	0	0	0	0	0
max_features	‘log2’	3	4	5	3	5	1
	‘auto’	7	6	5	7	5	9
max_depth	2	0	0	4	1	0	0
	3	5	5	5	5	5	1
	4	2	1	0	3	4	7
	5	2	2	1	1	1	2
	6	1	0	0	0	0	0
	7	0	2	0	0	0	0
split_criterion	*gini*	9	9	10	6	10	10
	entropy	1	1	0	4	0	0
n_estimators	10	2	3	0	5	0	0
	25	5	1	1	2	1	0
	50	2	1	4	1	0	1
	75	0	0	2	2	0	0
	100	0	0	2	0	5	5
	125	1	4	0	0	3	1
	150	0	1	1	0	1	3

References

1. Coinhive. https://tinyurl.com/ybsy89k2
2. CoinMarketCap. https://tinyurl.com/o94fhlw
3. Crypto-Loot. https://tinyurl.com/y76ppd5g
4. The *nvprof* Tool. https://tinyurl.com/y8tqxn74
5. The *perf* Tool. https://tinyurl.com/ybpmxw8
6. The *stress-ng* Tool. https://tinyurl.com/my6ehnj
7. An Italian Bank’s Server was Hijacked to Mine Bitcoin (2017). https://tinyurl.com/yac8c8jq
8. Persistent Drive-by Cryptomining Coming to a Browser Near You (2017). https://tinyurl.com/yd5roadb
9. Bitcoin Energy Consumption Index (2018). https://tinyurl.com/y8w5yj9l
10. Cryptojacking: A Modern Cash Cow (2018). https://tinyurl.com/y28eqdav
11. Cryptojacking Attack Found on Los Angeles Times Website (2018). https://tinyurl.com/y8ghcvmd
12. FacexWorm Targets Cryptocurrency Trading Platforms, Abuses Facebook Messenger for Propagation (2018). https://tinyurl.com/yd2zja9q
13. Greedy Cybercriminals Host Malware on GitHub (2018). https://tinyurl.com/y9qon8ch
14. Is Bitcoin Mining Profitable or Worth it in 2018? (2018). https://tinyurl.com/ybnydb8g
15. KSN Report: Ransomware and Malicious Cryptominers 2016–2018 (2018). https://tinyurl.com/y29kybtx
16. Revenues Down, Hashrates Up: 2018 Mining Outlook by the Numbers (2018). https://tinyurl.com/yc586s9v
17. rTorrent Client Exploited in the Wild to Deploy Monero Crypto-miner (2018). https://tinyurl.com/yaqy7u3k
18. Tesla Hackers Hijacked Amazon Cloud Account to Mine Cryptocurrency (2018). https://tinyurl.com/y9epv3do
19. UK ICO, USCourts.gov. Thousands of Websites Hijacked by Hidden Cryptomining Code after Popular Plug-in Pwned (2018). https://tinyurl.com/y7upaxgv
20. WebCobra Malware Uses Victims’ Computers to Mine Cryptocurrency (2018). https://tinyurl.com/ycuhowb3

21. Bitcoin Mining Pools (2019). https://tinyurl.com/y8pdk922
22. Feature Importances with Forests of Trees (2019). https://tinyurl.com/y3nlad2h
23. IBM X-Force Threat Intelligence Index (2019). https://tinyurl.com/y5nbprve
24. SonicWall Cyber Threat Report (2019). https://tinyurl.com/y3wj69s7
25. Under the Hood of Cyber Crime (2019). https://tinyurl.com/ydhauj8x
26. Bonneau, J., et al.: SoK: research perspectives and challenges for bitcoin and cryptocurrencies. In: 36th IEEE S&P, pp. 104–121 (2015)
27. Chiappetta, M., et al.: Real-time detection of cache-based side-channel attacks using hardware performance counters. Appl. Soft Comput. **49**, 1162–1174 (2016)
28. Comodo Cybersecurity: Global Threat Report Q2 2018 Edition (2018). https://tinyurl.com/y8eos9pl
29. Conti, M., et al.: On the economic significance of ransomware campaigns: a bitcoin transactions perspective. Comput. Secur. **79**, 162–189 (2018)
30. Cortes, C., Vapnik, V.: Support vector networks. Mach. Learn. **20**(3), 273–297 (1995)
31. Cyber Threat Alliance (CTA): The illicit cryptocurrency mining threat report (2018). https://tinyurl.com/yco7cykl
32. Demme, J., et al.: On the feasibility of online malware detection with performance counters. In: 40th ISCA, pp. 559–570 (2013)
33. Gangwal, A., Conti, M.: Cryptomining cannot change its spots: detecting covert cryptomining using magnetic side-channel. IEEE Trans. Inf. Forensics Secur. **15**(1), 1630–1639 (2019)
34. Ho, T.K.: Random decision forests. In: 3rd ICDAR, pp. 278–282 (1995)
35. Hsu, C.W., et al.: A practical guide to support vector classification. Tech. rep. (2003)
36. Huang, D.Y., et al.: Botcoin: monetizing stolen cycles. In: 21st NDSS, pp. 1–16 (2014)
37. Konoth, R.K., et al.: MineSweeper: an in-depth look into drive-by cryptocurrency mining and its defense. In: 25th ACM CCS (2018)
38. Liu, J., et al.: A novel approach for detecting browser-based silent miner. In: 3rd IEEE DSC, pp. 490–497 (2018)
39. Mora, C., et al.: Bitcoin emissions alone could push global warming above 2 °C. Nat. Clim. Change **8**(11), 931–933 (2018)
40. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). https://tinyurl.com/3f4a6lr
41. Rauchberger, J., et al.: The other side of the coin: a framework for detecting and analyzing web-based cryptocurrency mining campaigns. In: 13th ARES, pp. 1–10 (2018)
42. Rüth, J., et al.: Digging into browser-based crypto mining. arXiv preprint: 1808.00811 (2018)
43. Tahir, R., et al.: Mining on someone else's dime: mitigating covert mining operations in clouds and enterprises. In: Dacier, M., Bailey, M., Polychronakis, M., Antonakakis, M. (eds.) RAID 2017. LNCS, vol. 10453, pp. 287–310. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66332-6_13
44. Wang, W., Ferrell, B., Xu, X., Hamlen, K.W., Hao, S.: SEISMIC: SEcure In-lined Script Monitors for Interrupting Cryptojacks. In: Lopez, J., Zhou, J., Soriano, M. (eds.) ESORICS 2018. LNCS, vol. 11099, pp. 122–142. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98989-1_7
45. Wang, X., et al.: ConFirm: detecting firmware modifications in embedded systems using hardware performance counters. In: 34th IEEE/ACM ICCAD, pp. 544–551 (2015)

46. Wang, X., et al.: Hardware performance counter-based malware identification and detection with adaptive compressive sensing. ACM TACO **13**(1), 1–23 (2016)
47. Wang, X., Karri, R.: NumChecker: detecting kernel control-flow modifying rootkits by using hardware performance counters. In: 50th DAC, pp. 1–7 (2013)
48. Yuan, L., et al.: Security breaches as PMU deviation: detecting and identifying security attacks using performance counters. In: 2nd ACM SIGOPS APSys, pp. 1–6 (2011)