DeCrypto Pro: Deep Learning Based Cryptomining Malware Detection Using Performance Counters

Ganapathy Mani
Purdue University
West Lafayette, IN USA
manig@purdue.edu

Vikram Pasumarti
Purdue University
West Lafayette, IN USA
vpasuma@purdue.edu

Bharat Bhargava
Purdue University
West Lafayette, IN USA
bbshail@purdue.edu

Faisal Tariq Vora
Purdue University
West Lafayette, IN USA
voraf@purdue.edu

James MacDonald
Northrop Grumman Corporation
McLean, VA USA
jim.macdonald@ngc.com

Justin King
Northrop Grumman Corporation
McLean, VA USA
Justin.King@ngc.com

Jason Kobes
Northrop Grumman Corporation
Nashville, TN USA
jason.kobes@ngc.com

***Abstract*—Autonomy in cybersystems depends on their ability to be self-aware by understanding the intent of services and applications that are running on those systems. In case of mission-critical cybersystems that are deployed in dynamic and unpredictable environments, the newly integrated unknown applications or services can either be benign and essential for the mission or they can be cyberattacks. In some cases, these cyberattacks are evasive Advanced Persistent Threats (APTs) where the attackers remain undetected for reconnaissance in order to ascertain system features for an attack e.g. Trojan Laziok. In other cases, the attackers can use the system only for computing e.g. cryptomining malware. APTs such as cryptomining malware neither disrupt normal system functionalities nor trigger any warning signs because they simply perform bitwise and cryptographic operations as any other benign compression or encoding application. Thus, it is difficult for defense mechanisms such as antivirus applications to detect these attacks. In this paper, we propose an Operating Context profiling system based on deep neural networks—Long Short-Term Memory (LSTM) networks—using Windows Performance Counters data for detecting these evasive cryptomining applications. In addition, we propose Deep Cryptomining Profiler (DeCrypto Pro), a detection system with a novel model selection framework containing a utility function that can select a classification model for behavior profiling from both the light-weight machine learning models (Random Forest and k-Nearest Neighbors) and a deep learning model (LSTM), depending on available computing resources. Given data from performance counters, we show that individual models perform with high accuracy and can be trained with limited training data. We also show that the DeCrypto Profiler framework reduces the use of computational resources and accurately detects cryptomining applications by selecting an appropriate model, given the constraints such as data sample size and system configuration.**

***Index Terms*—deep learning, LSTM, machine learning, malware, cryptojacking, cryptomining, ransomware, performance counters, Advanced Persistent Threat, collaborative attacks**

I. Introduction

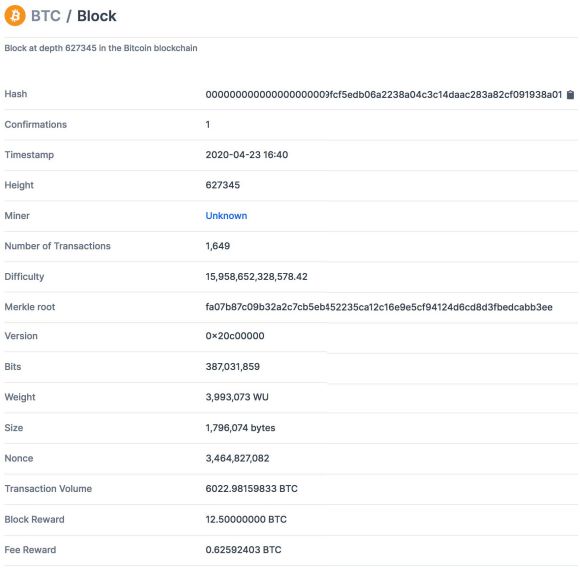
Enhancing self-autonomy in cybersystems, especially, systems with limited computing capability is a non-trivial research problem. These systems should be able to recognize their own actions or applications that are benign and normal as well as detect or predict the actions of applications or services that are malicious. In particular, mission-critical autonomous cybersystems, deployed in unpredictable and dynamic environments, face the problem of limited training data [1] [2]. These systems need to train faster and make decisions quicker. The need for continuous availability [3] of mission-critical systems provides evasive APTs with numerous opportunities. APTs are a class of cyberattacks where the attackers will stay in the system for extended periods of time to observe and execute their malicious activities. They can execute those actions step-by-step by waiting for a long time between each step, making it difficult for detection by antivirus or learning and prediction models [4]. APTs can be categorized into two major types: (1) attacks that try to slowly change or corrupt the fundamental operations of the systems [5] and (2) attacks that only use the infected systems for computing or reconnaissance missions [6]. Both of these attacks aggressively avoid detection since they employ techniques such as file type polymorphism (e.g. file-less attacks) [7] [8], static code polymorphism (e.g. packing, binary obfuscation, or encryption) [9], dynamic behavior polymorphism (understand the system defenses and change the behavior accordingly to evade detection) [10], or simply mimic benign applications [11]. Before explaining our contributions, we provide background and challanges of cryptojacking below.

A. Background: Cryptojacking

Cryptojacking—where collaborative attackers run cryptocurrency miners on victims' systems without their authorization and utilize their Central Processing Units' (CPUs) computational power to mine cryptocurrencies—is a combination of the two APT types: the attack uses only computing power of the victim and they have a potential to increase CPU usages as well as stop or slowdown other vital processes in the system. Cryptojacking is becoming pervasive: Symantec detected 8 million cryptojacking attacks in just three months (from December 2017 to February 2018) [12]. The demand for cryptojacking stems from the need for aggressive cryptomining. Cryptomining is the process of generating wealth by creating and verifying new blockchain-based cryptocurren-

978-1-7281-7277-4/20/$31.00 ©2020 IEEE
DOI 10.1109/ACSOS49614.2020.00032

cies. Cryptocurrencies are digital assets designed for secure and transparent transactions facilitated by Blockchain public ledgers.



BTC / Block

Block at depth 627345 in the Bitcoin blockchain

Hash	00000000000000000009fcf5edb06a2238a04c3c14daac283a82cf091938a01
Confirmations	1
Timestamp	2020-04-23 16:40
Height	627345
Miner	Unknown
Number of Transactions	1,649
Difficulty	15,958,652,328,578.42
Merkle root	fa07b87c09b32a2c7cb5eb452235ca12c16e9e5cf94124d6cd8d3fbedcabb3ee
Version	0x20c00000
Bits	387,031,859
Weight	3,993,073 WU
Size	1,796,074 bytes
Nonce	3,464,827,082
Transaction Volume	6022.98159833 BTC
Block Reward	12.50000000 BTC
Fee Reward	0.62592403 BTC

Fig. 1. A cryptocurrency (Bitcoin) blockchain's block [13]

Each block in a cryptocurrency blockchain has three elements: (1) data (transactions and their relevant information such as sender, receiver, and reward), (2) unique hash signature (generated using *SHA-256* hash function with data plus a random number as input), and (3) unique hash signature of previous block. Fig. 1 shows a sample block of Bitcoin cryptocurrency. The fundamental mechanism of the blockchain ledger is that it is public and open to everyone in the network. When there is a new transaction, it is bundled into a block, everyone in the network must verify the block's information before it is linked as another block to the major blockchain. In order for the new block to be linked to the major blockchain, the right hash value must be calculated i.e. the right random number should be decided and added to the data element in the block to generate the correct hash that fits the previous block. This process is known as Proof-of-Work (PoW) [14]. Finding the right hash given a large set of random numbers is arbitrary and a brute-force mechanism. Solving this blockchain puzzle is a computationally intensive process that requires extensive computing architecture and powerful machines such as GPUs, Application-Specific Integrated Circuits (ASICs), and advance FPGAs for increasing the probability of finding the right PoW [15] [16]. As the cryptocurrency blockchain grows longer, the difficulty of finding a new block goes higher. In Fig. 1, the Difficulty property shows the number of estimated hash computations required to find a new block, which is approximately 16 trillion hash computations. It is also economically prohibitive. Hence, there is a high demand for more and more computing power.

Cryptomining pools can help in sharing the burden of that computationally intensive process. Mining pools provide democratized access to users where users can pool their computing resources together in finding the hash for each new block of a particular currency's blockchain. Computing systems in the pool, trying to solve a new block are essentially digital workers and each one is rewarded by equally splitting the currency for every mined block. Each cryptocurrency has its own crypto-algorithm that sets the mathematics behind the PoW e.g. CryptoNight supports the digital currency Monero [17]. Equipped with static algorithms such as CryptoNight, users can mine specific currencies by joining the cryptomining pools.

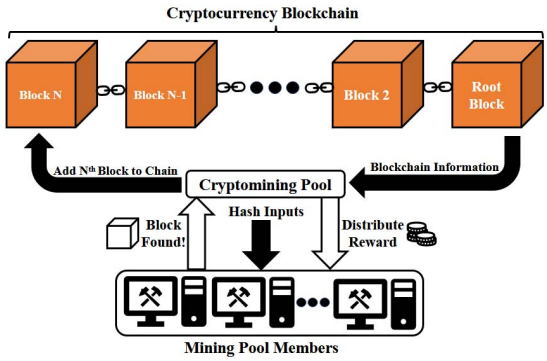


Fig. 2. Cryptomining workflow

Fig. 2 shows the workflow of mining cryptocurrencies using cryptomining pools. The cryptomining pools will have the whole distributed blockchain ledger and act as intermediaries with pool members. A mining pool service issues the hash inputs that are needed to be computed for finding the best PoW to all the systems that have signed up with the mining pool service. If an appropriate block is found, it is added to the major blockchain of the cryptocurrency. In return for providing the computing resources, mining users get the equally distributed reward and pooling service gets its service fee. Pooling services are easy to access and this creates a significant potential for deploying these miners in a large number of systems. Attackers can deploy these mining services through native applications or browser-based web applications [18] where they can mine any particular cryptocurrency in the background without authorization from the victim.

1) Challenges in Detecting Cryptojacking: Many detection techniques use cryptomining's high CPU usage and overheating of the system as triggers to alert the antivirus of potential mining activity [19]. But cryptominers can evade detection through the following techniques and their properties:

- Native mining pool applications such as XMRig [20] have minute control over execution and operating environment parameters where they can set the threshold for CPU usage (in percent).
- Some of the cryptominers employ a drive-by mining technique where they stay on the system for a very short time and use the CPU, before moving on to the next victim [21]. They will come back again and repeat the

process. This short time availability makes it difficult for antivirus software to detect mining activities.

- Most cryptomining algorithms such as CryptoNight, only perform cryptographic calculations, bitwise operations, and encryption operations. They neither change nor disrupt the system's behavior in any significant manner. This makes detection difficult since benign applications such as compression or encoding applications perform similar cryptographic, bitwise, and encryption operations. Due to this issue, the system's defense mechanisms can generate significantly high false negatives, which makes the system oblivious to the threat. Alternatively, they can generate too many false positives, which reduces the usability of the system.
- In both false negatives and false positive scenarios, the system will have to allocate a significantly large portion of computing resources for security operations, which may slow the vital processes in mission-critical cyber-systems.

B. Our Contributions

Given the challenges posed by cryptojacking (or evasive malware, in general) and its detection evasion, we propose a robust solution for cryptojacking detection and prediction by operating context profiling—a type of behavior profiling where applications are classified based on their holistic effect of their operating context.

1) We propose DeCrypto profiler, an operating context profiler that is equipped with both light-weight machine learning models—k-Nearest Neighbors (k-NN) and Random Forest (RF)—as well as a deep learning model (LSTM). Light-weight machine learning models are guided by feature selection and LSTM selects features automatically.
2) We leverage the performance counters data [22] for the training of machine learning and deep learning models. Computing systems that use the Windows operating system have 68 unique performance counters and 245 counter values. Contrary to Hardware Performance Counters (HPCs), the performance counters provide us the status of components from the application-level to the hardware-level i.e. holistic status of the system. This provides a significantly large feature space that can help the model to solidify its training. These counters also provide imperturbable effects of the operational context i.e. cryptomining malware are not equipped to change the outcome of the effects displayed by the performance counters.
3) In order to select the best prediction and detection model bespoke to the computational resources, a model selection framework with utility function is provided. Given newly collected performance counter data, the utility function considers the F1 scores provided by the models during retraining and employs the one with the best F1 for cryptojacking detection. This reduces the computational resource usage.
4) DeCrypto Pro provides both trigger-based and non-trigger-based cryptojacking detection by altering the sampling techniques. If it is a trigger-based detection (example trigger: high CPU usages threshold set by the user) then the profiler employs fixed interval sampling of performance counters whereas non-trigger-based detection employs sampling at random intervals to check the system status.
5) DeCrypto Pro can be periodically retrained for identifying new novel cryptojacking attacks. Given the high dimensional feature space, DeCrypto Pro needs considerably less data for initial training as well as retraining.
6) Since performance counters data is composed of the values from a number of system properties, they provide holistic status of the system. Thus DeCrypto Pro can be trained to identify a wide-range of evasive APT classes.

Through experiments, we show that learning models of DeCrypto Pro provide high accuracy with significantly less false positive and false negative rates, and they need considerably less data for training and retraining. We also show that the model selection framework reduces the computational resource usage by estimating training times and comparing them between utility function and random model selection.

C. Paper Organization

The rest of the paper is organized as follows: Section II provides the related work with respect to cryptojacking and APTs as well as their detection mechanisms, Section III provides the threat model for cryptojacking attacks with specific assumptions of the operational environment and its settings, Section IV provides the details of performance counters and their properties, Section V explains the DeCrypto Profiler and the model selection framework, Section VI details the experimental setup, data collection, data processing, feature selection, results and findings of Decrypto profiler performance, and finally, we conclude our work.

II. Related Work

Cryptojacking is a fairly new class of cyberattack, however it is growing in sophistication, especially in detection evasion. There are a number of techniques proposed in literature using Hardware Performance Counters (HPCs) and machine learning for detecting evasive cyberattacks similar to cryptojacking. Using HPCs for detecting kernel-level rootkits is proposed in [23]. Rootkits are a specific class of malware that modify some parts of the operating system kernel for hiding the presence of malicious attackers in the system. The detection framework uses three learning models (1) One-Class Support Vector Machine (OC-SVM)—an unsupervised learning algorithm, (2) Decision Trees (DTs), and (3) Naive Bayes classifier. Training data consists of HPC traces of synthetic rootkits and benign applications collected through Intel VTune [24] High impacting features are selected through WEKA framework [25]. The solution heavily relies on proprietary tools such as Intel VTune. The learning models may not work on APTs as the data only captures the hardware-level traces. In

addition, long-term dependencies in traces are not considered where rootkits can evade detection by not manifesting for long periods of time. A similar approach is employed in [26] using Linux HPCs and learning models: RF, k-NN, and DTs. HPCs are collected through fixed interval sampling and features are manually extracted from raw data. The solution does not consider APTs since the long-term dependencies in data are not taken into consideration. Without a model selection framework, using three learning models can be computationally expensive for retraining and prediction. A supervised learning-based approach using HPC traces and SVM has been detailed in [27]. The solution cannot handle high dimensional data and requires manual verification of Recursive Feature Elimination (RFE) technique employed for feature selection. RFE does not consider weak features even though they may be useful in contextual behavior profiling of malware. Signature-based dynamic and static analysis as well as detection of evasive malware are proposed in [5], [28], [29], and [30]. But APTs are constantly evolving and adapting to defensive mechanisms. Those solutions are not adaptable to new APTs such as evasive cryptomining.

Recent discovery of covert cryptomining operations has paved the way for several approaches for detection. Detection by dynamic opcode analysis is proposed in [18]. The authors capture HTML files that contain cryptomining JavaScript code. They create benign training data synthetically by removing `start()` function from the HTML files. The solution employs feature selection using WEKA framework and learning using RF. This approach is not adaptive to dynamically changing mining algorithms and it cannot be applied for detecting native unauthorized cryptomining applications. A model is proposed that leverages magnetic side-channel signature and k-NN classifier in [31]. But magnetic side-channel signatures can be easily influenced by load modulations that can be executed by evasive malware applications [32]. In addition, using this one feature can omit contextual factors that can provide a holistic signature of the malware. Similarly, another signature-based approach is proposed in [33]. The authors use fingerprinting through a score computed from user processor time, memory usage (in %), number of running threads, number of active socket connections, and usage of CPU (in %). But cryptomining applications can set the memory and CPU usage and mimic the benign application such as compression or encoding applications. Deep learning-based detection mechanism for cryptojacking is detailed in [34]. The static and dynamic analysis involves capturing system calls of Portable Executable (PE) samples of cryptomining applications. The collected sequential system call data are then sent into the deep learning models—LSTM, Action-based LSTM, and Convolutional Neural Networks (CNNs). The proposed solution does not deal with browser-based cryptomining operations. Evasive malware can hide their behavior by system call and code obfuscation.

We propose a deep learning-based approach using holistic system telemetry that is representative of signatures from the application-level to the hardware-level. We also propose a model selection framework that selects a detection model based on the computational resources available, which makes the system more efficient in terms of security operations. Our solutions is applied to both native and browser-based cryptomining detection. DeCrypto Pro can also be extended to detect new class of evasive malware.

III. Threat Model

Cryptomining malware attacks are carried out by collaborating mining pools that mine the same currencies as any other similar type of cryptocurrencies using the same mining (PoW) algorithm e.g. Monero and Zcash [16]. These cryptominers are deployed as native or browser-based applications. In native applications, the mining process gets carried on without authorization even after stopping the mining process. Figure 3 shows the attack workflow. These mining applications can be transferred to other systems in network with the victim's credentials. In addition, cryptomining malware comes in the form of web applications e.g. Coinhive miner [35]. Browser-based malware are most evasive as they employ drive-by mining [36] when the victim visits the malicious website. The attack can be short-lived where mining is accelerated for a short period of time, which increases the CPU usage. Since this high CPU usage can be used as a trigger to alert the antivirus defenses of the system, both browser-based and native-based mining applications can set the threshold for CPU usage.

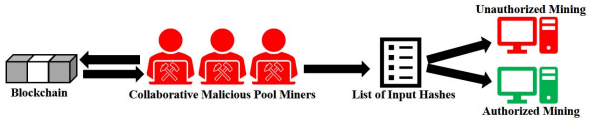


Fig. 3. Threat model for collaborative cryptomining malware (Cryptojacking)

```
"cpu": {
    "enabled": true,
    "huge-pages": true,
    "hw-aes": null,
    "priority": null,
    "memory-pool": false,
    "yield": true,
    "max-threads-hint": 100,
    "asm": true,
    "argon2-impl": null,
    "astrobwt-max-size": 550,
    "cn/0": false,
    "cn-lite/0": false
}
```

This `config.json` file from XMRig miner contains the threshold level setup for maximum CPU usage (`max-thread-hint`) [37]. The high CPU usage trigger can be evaded by setting the minimum CPU usage and slowly mine the currencies like APTs. We assume that the attacker is fully aware of the system defenses and employ more evasive techniques. We also assume that the attackers can collaborate in order to maximize the attack effectiveness. For example, attackers can take turns on drive-by mining with different application instances to hide or neutralize the behavior. When the mining is taking place, the miners execute

the PoW algorithm such as CryptoNight over and over again. This involves a large number of bitwise, cryptographic, and encryption operations. Hence the threat model consists of the attackers' evasive tactics as well as the algorithmic footprints. Using performance counters, we fingerprint the holistic effect of these algorithms, which must be executed in system for mining, on the system.

IV. Performance Counters

Performance counters are small programs that count, monitor, and measure events in the system. They provide information on how a particular operating system or a service, an application or a driver is performing. Thus the information provided by the counters involve status of the system from the application-level to the hardware-level. If there are multiple services or applications running simultaneously, then the counters provide operating context information i.e. how did the counter information change after new applications start to run or old applications get terminated. These counters can help the system to identify bottlenecks and optimize the application as well as system performance. Windows provides a unique set of performance counters that contain 68 properties. Some of the counters for example:

```
logicaldisk(harddiskvolume3)\diskreadtime
logicaldisk(harddiskvolume3)\diskreads/second
processor(3)\c3transitions/second
processor(1)\processortime
processor(0)\dpcsqueued/second
physicaldisk(0c)\avg.diskqueuelength
physicaldisk(_total)\avg.diskbytes/read
memory\systemcodetotalbytes
memory\poolpagedresidentbytes
```

These 68 counters can produce several hundred values depends on the number of processor cores in the CPU. In this research, each sample in the data set contains 245 values. The data contains information from four major system entities (each has several performance counters): Logical disks, Processors, Physical Disks, and Memory. The performance counters are similar to a template [38] for each operating context data item, where values for template item are filled for every sample.

A. Advantages

- Influencing these individual counters is costly for an attacker, especially for a drive-by mining malware attack.
- It is also difficult for APTs to manage and influence these counters over long periods of time.
- Performance counters provide operating context data, which means they are independent of any particular processes or application services.
- Due to the high dimensionality of data, the learning models are provided with feature-rich data for accurate prediction and detection of APTs.

V. Deep Cryptomining Profiler (DeCrypto Pro)

DeCrypto Pro provides a streamlined research approach for detecting as well as predicting evasive cryptomining actions. Compared to the existing techniques, which use fingerprinting based on specific artifacts such as network traffic, specific background processes and registry keys, function hooks, or IP address, DeCrypto Pro uses resilient performance counters data for effective prediction with efficient model selection, which reduces the computational resource usage. This will help mission-critical systems to focus on important tasks rather than spending resources on security operations.

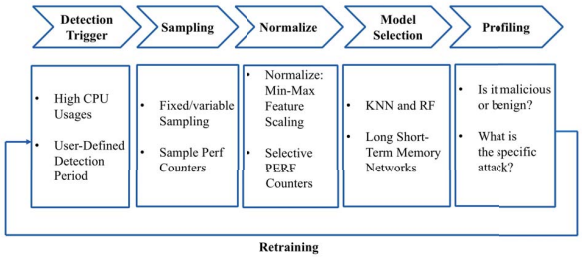


Fig. 4. Framework of Deep Cryptomining Profiler

DeCrypto Pro uses both environmental detection triggers such as high CPU usages (Note: The threshold for CPU usage can be set by the user) or period detection (random or fixed intervals depends on the user's preference) of cryptomining malware. As shown in Figure 4, the sampling of performance counter data also follows the similar method of fixed sampling (every *n* instructions or seconds) or variable random sampling. For this study, we use variable random sampling. Once the sampling of data is completed, it is normalized through Min-Max feature scaling and important features will be selected (Feature selection is discussed at length in Section VI). A model selection utility function considers the computational resources and previous 10-fold cross validation training accuracy and F1 Scores to determine the best model for profiling. Based on the results, users or the antivirus defenses can be alerted. The profiler can also be periodically retrained with newly discovered cryptomining malware or other APTs.

A. Model Selection

In order to make an efficient detection of cryptomining malware, we employ a model selection utility function that can select an optimal model given computational resources such as processor frequency and memory as well as accuracy and F1 scores of the model training.

$$M_i = S_{pc} + S_{pf} + S_m + \frac{1}{k}\sum_{v=1}^{k}\left(\frac{P_v}{2}\right) \tag{1}$$

Here, M_i is the value for model selection where i = k-NN, LSTM, and RF, S_{pc} is number of processor cores, S_{pf} is processor frequency with just the decimal value (e.g. 2.9 GHz = 2.9), similar to processor frequency, S_m represents the memory capacity of the system (e.g. 16 GB = 16), k is number of cross validations, and P is the prediction value obtained by the sum of accuracy and F1 score.

$$\forall i \; M_p < M_c \tag{2}$$

Here, M_p is previous model selection value and M_c is current model selection value. Each one of the models are compared

against the other two. If the condition in equation 2 is satisfied then the current model is selected i.e. best out of three models. Otherwise, the previous model is still selected for profiling.

B. Random Forest and k-Nearest Neighbor Models

Random Forest can detect a compromised system with high accuracy using less computational resources. Training and testing a Random Forest have time complexities of $O(n^2ct)$ and $O(ct)$ respectfully for *n* training examples, *c* features, and *t* trees. In other words, computational resources used for Random Forest are dependent on the number of examples in the Performance Counter data set, the number of performance counters we are using, and the number of trees in the forest. In our case, we are using a Random Forest of 100 trees with a maximum depth of 10 levels and with this model training and testing takes only a few seconds.

Even more computationally simple model is k-NN, whose training time is negligible and whose testing time complexity is $O(nc)$ for *n* training examples and *c* features. In other words, computational resources used for k-NN are dependent on the number of examples in the performance counter data set and the number of performance counters themselves. In our case, we are using k-NN with a *K* of 20 and with this model training and testing again takes seconds. Thus it is computationally advantageous to use k-NN or Random Forest.

C. Long Short-Term Memory Model (LSTM)

Deep learning techniques such as LSTM are used when selecting features manually or based on a static heuristic becomes cumbersome. LSTM networks select the important features automatically based on value maximization of a feature over time. In this paper, cryptomining malware depends on the operating context performance counter data. Deep learning models are ideal for considering the large feature space of performance counters since features are automatically selected by the LSTM model to make an accurate classification.

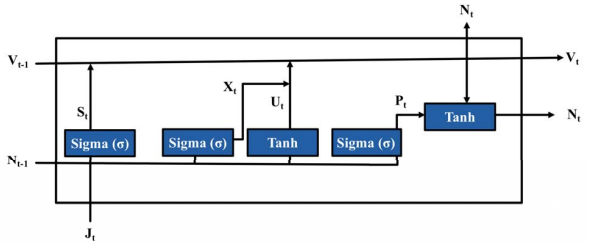


Fig. 5. An example cell of LSTM

As shown in Figure 5, LSTM contains a cell, input/output (I/O) gates, and forget gate constitute LSTM. The cell remembers important values and I/O gate and forget gate regulate information flow from one cell to another. LSTM cell operations can be described step-by-step [39].

1) Identify which input to keep and which one to discard.

$$S_t = \sigma(weight_S \cdot (N_{t-1}, J_t) + Bias_S) \quad (3)$$

2) Identify the new information to be stored.

$$X_t = \sigma(weight_X \cdot (N_{t-1}, J_t) + Bias_X) \quad (4)$$

$$U_t = \tanh(weight_U \cdot (N_{t-1}, J_t) + Bias_U) \quad (5)$$

3) Update the state of the cell with the new information.

$$V_t = S_t * V_{t-1} + X_t * U_t \quad (6)$$

4) Output of the LSTM cell.

$$P_t = \sigma(weight_P \cdot (N_{t-1}, J_t) + Bias_P) \quad (7)$$

$$N_t = P_t + \tanh(V_t) \quad (8)$$

LSTM is a variant of Recurrent Neural Network (RNN), which is widely used for sequence-to-sequence classification. RNN has a Vanishing Gradient (VG) problem that causes bad model training. VG can be addressed based on Gated Recurrent Unit (GRU) or LSTM. LSTM has been shown to have higher accuracy [40].

VI. Experimental Results

Experiments were set up on 3 Windows machines with various configurations on processing frequency (2.40, 2.90, 2.30 GHz), memory size (16, 8, 8 GB), and number of processor cores (2, 4, 5). Each machine provided a unique signature of operating context since all of them had different applications and services installed, including various versions of drivers. Since we aim to capture the system status signature of PoW algorithm such as CryptoNight's signature, we mainly focus on bitwise, cryptographic, and processor-specific encryption operations. Thus we consider compression software (7-Zip, SecureZip, PeaZip, WinRAR, WinZip, and Freemake) as our benign examples and cryptomining applications (XMRig, XMR-Stak, Coinhive, Computta, and GUIminer) as malicious example. Here, Coinhive is a browser-based miner and XMRig covers the most widely used PoW algorithms—RandomX, CryptoNight, AstroBWT and Argon2.

Our experiments are designed to answer the following Research Questions (RQs):

1) Can DeCrypto Pro deal with multiple cryptomining applications at the same time?
   Any defense mechanism must be able to classify (through operating context) whether there is malicious activity going on in the system and what is the specific type of application that is causing the operating context to behave in such a way. To answer this RQ, we will conduct both binary as well as multi-class classification using all the models.
2) Can human-expert-in-the-loop make DeCrypto Pro effective?
   LSTM can select features automatically. But for k-NN and RF, the features have to be manually selected by human experts or by predefined heuristics. In order to answer this question, we will test the performance of k-NN and RF without feature selection i.e. giving all features as input as well as with manual feature selection (Discussed in Section VI-D).

3) What is the amount of training data needed for an accurate classification by DeCrypto Pro?
   By answering this question, we test if the research approach and learning models can be deployed in mission-critical systems. For mission-critical autonomous systems, there is a demand for training and retraining to be quick when the new data arrives. To answer this question, we change the training and testing split of the data for all the models and measure the performance.
4) Can selecting the best model based on retraining performance and computational resources reduce the system overhead?
   By answering this question, we can test if employing several learning models can be useful without adding additional computational resource overhead. For answering this research question, we measure the time for each training and retraining with model selection based on utility function and random model selection, and compare the performance.

A. Performance Metrics for Malware Detection Systems

We use the fundamental metrics used for evaluating malware detection systems: *F1-measure (F1)*, *Precision (P)*, *True Positive Rate (TPR) or Recall (R)*, *False Positive Rate (FPR)*, and *False Negative Rate (FNR)* [41]. Given True Positives (TP), False Positives (FP), False Negatives (FN), and True Negatives (TP), F1 is calculated with $\frac{2*Precision*Recall}{Precision+Recall}$. Similarly, FPR is computed by $\frac{FP}{FP+TN}$ and FNR is nothing but $\frac{FN}{TP+FN}$. Recall and Precision are computed by $1-FNR$ and $\frac{TP}{TP+FP}$, respectively.

For any malware detection system, it is ideal if it does not miss the detection of any malware (i.e. Precision $\approx$ 1 and Recall $\approx$ 1). The detection system should also not trigger too many false alarms (i.e. Precision $\approx$ 1 and FPR $\approx$ 0).

B. Data Collection

We used Windows PowerShell to obtain the performance counter data [42]. There are three settings for collecting operating context performance counter samples for training: (1) compression software is for benign examples, (2) cryptomining software running for malicious examples, and (3) both compression and cryptomining software running for malicious examples. Several samples were collected in random sampling intervals. First, we collected the list of all the counters available in the system using the command line command `TypePerf.exe -q > counters.txt`. We then use this counters list to capture all the counters per sampling operation. At random intervals, we obtain all the 245 performance counter values for each sample.

Performance counter data was collected while these compression and mining applications were running. We collected data by running different applications and then running the code to collect the performance counters for these applications. As Table I shows, we collected $\approx$1200 samples for each application and the whole data set represents almost an equal split of benign and malicious examples.

TABLE I
PERFORMANCE COUNTER DATA PER APPLICATION

Application	Number of Samples
7-Zip & PeaZip & Win (Zip/RAR)	1398 & 1200 & 2400
SecureZip & Freemake	1398 & 1320
Coinhive & Computta	1200 & 1398
GUIMiner & XMR (ig, Stak)	1398 & 2598
Compression + Mining Applications	1200

1) Bias Mitigation: Performance counters data was collected under various settings with several applications running on the side including computationally heavy graphics rendering games. Cryptomining applications were ran with default settings as well as trigger-avoiding settings such as less CPU usages. In addition, we employed the following bias mitigation techniques:

1) **Avoiding Contamination**: System settings are restored and cache is cleared between every data capture.
2) **Environmental noise and Input bias**: Both factors are not controlled as they provide contextual data capture of malware operating context.
3) **Network**: Data is collected with internet connection as well as without or semi-internet connection.
4) **User bias**: We do not control this bias as the data collection is automated, sampled in random intervals.
5) **Successful implementation**: We ensured that cryptominers are properly communicating with both online and local pool of miners.

C. Data Processing

Once the data is collected, we performed Min-Max feature scaling to normalize the data. We label each data row of performance counters data as 0 or 1 (benign or malicious) for binary classification as well as numbered classes (1-12) for multi-class classification. All of the null values are removed from the data set. In order to enhance the models' learning and avoid bias towards any particular class(es), number of data rows for each class is equally balanced through oversampling of under-sampled classes [43] i.e. if a class has fewer data rows than the other classes, more (duplicated) rows will be randomly selected from the under-sampled classes. This will also provide $\approx$ 50%-50% balance for benign and malicious labels. From the balanced data set, we randomly select 70% of the samples for training and 30% of the remaining samples for testing.

D. Feature Selection

For light-weight machine learning models—RF and k-NN —we perform feature selection on normalized data.

Figure 6 shows the average values of randomly selected $\approx$ 3000 samples of all 245 features. Some of the counters' rate of change is minimal, providing us with a clear and unique operating context signature.

32 features were manually selected by setting the threshold normalized value of a particular counter to be greater than 0.01 (Figure 7).

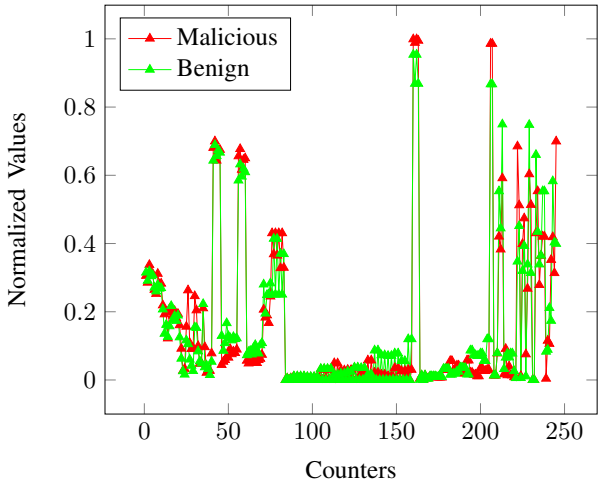


Fig. 6. Average values of a sample set of performance counters data

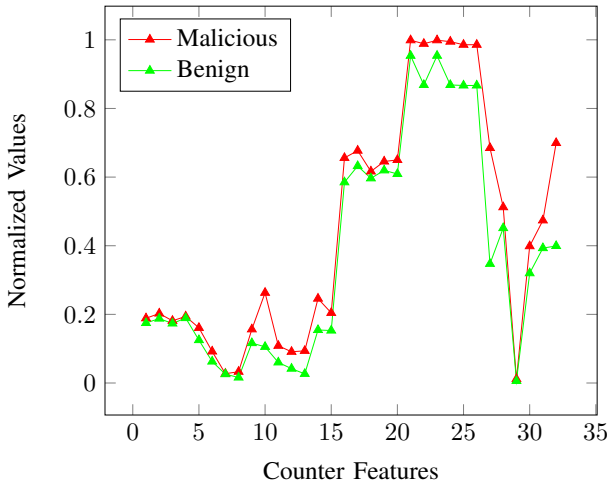


Fig. 7. Average values of selected 32 performance counter features

Note: It can be observed that the counter values for both benign and malicious applications are almost similar with minute variations. It is clearly indicative that the algorithmic operations that are performed by both application types are bitwise, encryption, and cryptographic operations. Thus, we tested RF and k-NN with both feature selection and without feature selection to measure the impact of human-expert-in-the-loop.

E. Implementation

We implemented k-NN and RF using the Scikit-learn version 0.21.2 Python library [44]. For model selection framework, we simulated random selection of models as well as selection based on utility function's output. Each model selection is performed with retraining of the models in random intervals with newly obtained data. Model selection is simulated with different settings of processor cores, processor frequency, and memory capacity. New performance counter data for retraining is obtained by varying system's operating context by running several applications simultaneously. LSTM is implemented through Tensorflow and TensorFlow Core v2.1.0 Python library is used [45]. Each model is executed with 10-fold cross validation.

1) Hyperparameter Optimization for LSTM Model: Given the performance counters data, we use grid search [46] to find the right hyperparameters to fit the LSTM model. We select the following values for LSTM Hyperparameters: Batch Size: 48, Epochs: 64, Kernel Initializer: `uniform`, Dropout rate: 0.4, Learning rate: 0.003, Number of hidden nodes: 200, Number of dense layers: 4, Activation Function: `sigmoid`, and Optimization Function: `Adam`.

F. Findings and Implications

1) Experiments for Answering RQ1: In order to test that DeCrypto Pro can detect and classify benign and malicious operating context through performance counters as well as classify individual malware, we perform both binary and multi-class classification.

TABLE II
PERFORMANCE METRICS FOR BINARY CLASSIFICATION

Model	P (%)	R (%)	F1 (%)	FNR (%)	FPR (%)
k-NN	95.71	92.27	93.64	7.73	3.82
RF	96.86	97.43	96.84	2.57	3.5
LSTM	99.45	99	99.9	0.01	0.15

TABLE III
PERFORMANCE METRICS FOR MULTI-CLASS CLASSIFICATION

Model	P (%)	R (%)	F1 (%)	FNR (%)	FPR (%)
k-NN	93	90.97	89.99	9.03	1.08
RF	97.62	97.82	97.62	2.18	0.263
LSTM	96	95	95.5	2.2	1.8

Insight 1: From Tables II and III, we observe that all three models provide consistently high performance for both binary classification and multi-class classification with less FNR and FPR. The F1 score is consistantly high for randomized data obtained with multiple applications running on the side. We theorize that cryptomining malware trigger specific performance counters. Thus DeCrypto Pro can detect multiple cryptominers.

2) Experiment for Answering RQ2: We manually selected features as specified in Section VI-D and identified 32 important performance counters. We implemented k-NN and RF models with 32-features.

Insight 2: From Tables IV and V, we observe that all the light-weight machine learning models of DeCrypto Pro provide consistently high performance for both binary classification and multi-class classification with human-exert-in-the-loop feature selection. Siimilar to insight 1, DeCrypto Pro feature selection provides the flexibility to reduce the number of features, if needed by experts.

3) Experiments for Answering RQ3: : In order to answer RQ3, we changed the testing and training data sizes for all the three models to test the percent of training data required to perform accurate classification.

TABLE IV
PERFORMANCE METRICS FOR MULTI-CLASS CLASSIFICATION

Model	P (%)	R (%)	F1 (%)	FNR (%)	FPR (%)
k-NN	89.85	87.91	86.64	12.09	1.43
RF	96.32	96.58	96.42	3.42	0.41

TABLE V
PERFORMANCE METRICS FOR BINARY CLASSIFICATION

Model	P (%)	R (%)	F1 (%)	FNR (%)	FPR (%)
k-NN	93.3	88.36	90.06	11.64	5.4
RF	97.4	86.57	89.85	13.43	2.07

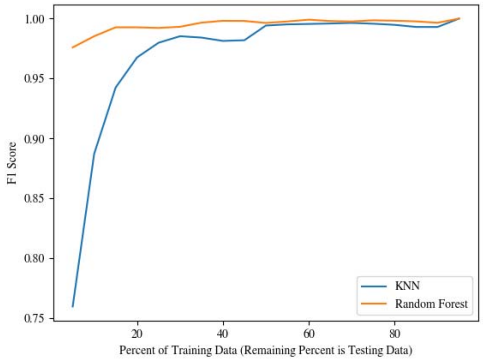


Fig. 8. Training-testing split for binary classification

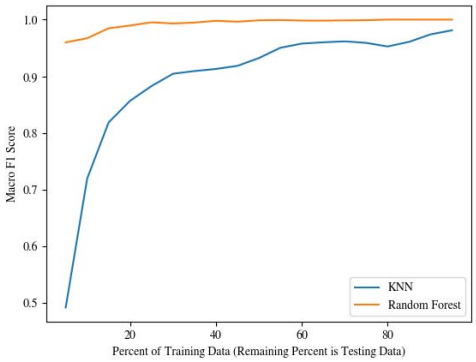


Fig. 9. Training-testing split for multi-class classification

Insight 3: From Figure 8 and 9, we observe that the light-weight machine learning models of DeCrypto Pro provide consistently high F1 scores with just around 30-40% of training data. Similar behavior is observed with LSTM model as well. We can infer that DeCrypto Pro can be successful in mission-critical autonomous systems since it can retrain itself with limited amount of data.

4) Experiment for Answering RQ4: : We wanted to test both random model selection and model selection based on utility function to test the hypothesis that retraining and inference takes considerably less time. We only consider training that provided high F1 score. *Insight 4:* From Figure 10,

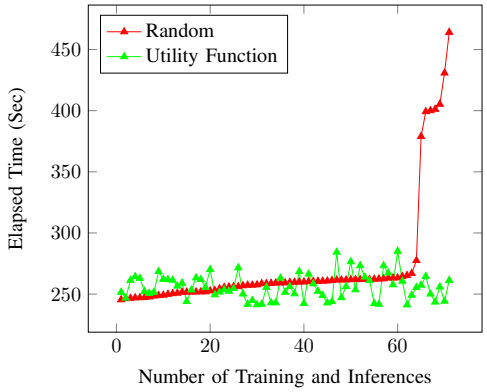


Fig. 10. Performance of Model Utility Function

initial results suggest that random model selection produces increasing training and inference times whereas utility function based selection provides relatively stable training times. These initial results indicate that given the large feature space, the light-weight models can perform equally well with LSTM. But manual feature selection might miss minute changes in performance counters.

VII. CONCLUSION

We presented DeCrypto Pro, a deep learning based cryptomining malware detection model that uses performance counter data, which aims to relieve human experts from difficult manual feature selection and reduce false negatives and false positives that are usually incurred by evasive APT detection systems. We showed that DeCrypto Pro can classify both benign and malicious operating contexts through binary and multi-class classification. Due to large feature space of performance counters, we observed that DeCrypto Pro models only require limited amount of training data for accurate classification of cryptomining malware. Classification results also showed that DeCrypto Pro incurs only minimal amount of false positives and false negatives. In addition, we presented the results on the efficiency of model selection utility function that can select the best model given the computational resources, making DeCrypto Pro to be best equipped for mission-critical systems. DeCrypto Pro can also be extended to include specific types of APTs for profiling their algorithms. As a future study, in addition to training DeCrypto Pro with more APT classes, we plan to expand on model selection framework to investigate more system setting use cases and integrate explainable AI framework with LSTM model to provide explainability with feature selection, training, and inference.

ACKNOWLEDGMENT

This research is funded by Northrop Grumman Corporation Research Consortium.

REFERENCES

[1] D. Danks and A. J. London, "Algorithmic bias in autonomous systems." in *IJCAI*, 2017, pp. 4691–4697.

[2] G. Mani, B. Bhargava, and B. Shivakumar, "Incremental learning through graceful degradations in autonomous systems," in *2018 IEEE International Conference on Cognitive Computing (ICCC)*. IEEE, 2018, pp. 25–32.

[3] J. Bosch and H. H. Olsson, "Data-driven continuous evolution of smart systems," in *2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2016, pp. 28–34.

[4] E. Vorontsov, C. Trabelsi, S. Kadoury, and C. Pal, "On orthogonality and learning recurrent networks with long term dependencies," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 3570–3578.

[5] D. Kirat, G. Vigna, and C. Kruegel, "Barecloud: bare-metal analysis-based evasive malware detection," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 2014, pp. 287–301.

[6] R. Chatterjee, P. Doerfler, H. Orgad, S. Havron, J. Palmer, D. Freed, K. Levy, N. Dell, D. McCoy, and T. Ristenpart, "The spyware used in intimate partner violence," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 441–458.

[7] J. B. Fraley and M. Figueroa, "Polymorphic malware detection using topological feature extraction with data mining," in *SoutheastCon 2016*. IEEE, 2016, pp. 1–7.

[8] F. Dang, Z. Li, Y. Liu, E. Zhai, Q. A. Chen, T. Xu, Y. Chen, and J. Yang, "Understanding fileless attacks on linux-based iot devices with honeycloud," in *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, 2019, pp. 482–493.

[9] D. Couroussé, T. Barry, B. Robisson, P. Jaillon, O. Potin, and J.-L. Lanet, "Runtime code polymorphism as a protection against side channel attacks," in *IFIP International Conference on Information Security Theory and Practice*. Springer, 2016, pp. 136–152.

[10] A. Cabrera and R. A. Calix, "On the anatomy of the dynamic behavior of polymorphic viruses," in *2016 International Conference on Collaboration Technologies and Systems (CTS)*. IEEE, 2016, pp. 424–429.

[11] M. Rigaki and S. Garcia, "Bringing a gan to a knife-fight: Adapting malware communication to avoid detection," in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 70–75.

[12] C. Symantec, "Internet security threat report 2019," 2019. [Online]. Available: https://docs.broadcom.com/doc/istr-24-2019-en

[13] www.blockchain.com, "Blockchain explorer - search the blockchain of BTC, ETH, BCH," 2020. [Online]. Available: https://www.blockchain.com/explorer

[14] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Manubot, Tech. Rep., 2019.

[15] J. Rüth, T. Zimmermann, K. Wolsing, and O. Hohlfeld, "Digging into browser-based crypto mining," in *Proceedings of the Internet Measurement Conference 2018*, 2018, pp. 70–76.

[16] Z. Feng and Q. Luo, "Evaluating memory-hard proof-of-work algorithms on three processors," *Proceedings of the VLDB Endowment*, vol. 13, no. 6, pp. 898–911, 2020.

[17] M. Documentation, "Cryptonight: Monero documentation," Tech. Rep., 2020. [Online]. Available: https://monerodocs.org/proof-of-work/cryptonight/

[18] D. Carlin, P. O'kane, S. Sezer, and J. Burgess, "Detecting cryptomining using dynamic analysis," in *2018 16th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 2018, pp. 1–6.

[19] R. K. Konoth, E. Vineti, V. Moonsamy, M. Lindorfer, C. Kruegel, H. Bos, and G. Vigna, "Minesweeper: An in-depth look into drive-by cryptocurrency mining and its defense," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1714–1730.

[20] S. G. Iyer and A. D. Pawar, "Gpu and cpu accelerated mining of cryptocurrencies and their financial analysis," in *2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC) I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC), 2018 2nd International Conference on*. IEEE, 2018, pp. 599–604.

[21] A. Zimba, Z. Wang, M. Mulenga, and N. H. Odongo, "Crypto mining attacks in information systems: An emerging threat to cyber security," *Journal of Computer Information Systems*, pp. 1–12, 2018.

[22] Microsoft, "Performance counters documentation," Tech. Rep., 2020. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/perfctrs/performance-counters-portal

[23] B. Singh, D. Evtyushkin, J. Elwell, R. Riley, and I. Cervesato, "On the detection of kernel-level rootkits using hardware performance counters," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017, pp. 483–493.

[24] J. Reinders, "Vtune performance analyzer essentials," *Intel Press*, 2005.

[25] Z. Markov and I. Russell, "An introduction to the weka data mining system," *ACM SIGCSE Bulletin*, vol. 38, no. 3, pp. 367–368, 2006.

[26] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, "On the feasibility of online malware detection with performance counters," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 559–570, 2013.

[27] V. Jyothi, X. Wang, S. K. Addepalli, and R. Karri, "Brain: Behavior based adaptive intrusion detection in networks: Using hardware performance counters to detect ddos attacks," in *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*. IEEE, 2016, pp. 587–588.

[28] C. Kruegel, "Full system emulation: Achieving successful automated dynamic analysis of evasive malware," in *Proc. BlackHat USA Security Conference*, 2014, pp. 1–7.

[29] D. Kirat and G. Vigna, "Malgene: Automatic extraction of malware analysis evasion signature," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 769–780.

[30] K. N. Khasawneh, N. Abu-Ghazaleh, D. Ponomarev, and L. Yu, "Rhmd: evasion-resilient hardware malware detectors," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 315–327.

[31] A. Gangwal and M. Conti, "Cryptomining cannot change its spots: Detecting covert cryptomining using magnetic side-channel," *IEEE Transactions on Information Forensics and Security*, 2019.

[32] J. Heyszl, D. Merli, B. Heinz, F. De Santis, and G. Sigl, "Strengths and limitations of high-resolution electromagnetic field measurements for side-channel analysis," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2012, pp. 248–262.

[33] D. Draghicescu, A. Caranica, A. Vulpe, and O. Fratu, "Crypto-mining application fingerprinting method," in *2018 International Conference on Communications (COMM)*. IEEE, 2018, pp. 543–546.

[34] H. Darabian, S. Homayounoot, A. Dehghantanha, S. Hashemi, H. Karimipour, R. M. Parizi, and K.-K. R. Choo, "Detecting cryptomining malware: a deep learning approach for static and dynamic analysis," *Journal of Grid Computing*, pp. 1–11, 2020.

[35] R. Tahir, S. Durrani, F. Ahmed, H. Saeed, F. Zaffar, and S. Ilyas, "The browsers strike back: countering cryptojacking and parasitic miners on the web," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 703–711.

[36] J. Segura, "A look into the global 'drive-by cryptocurrency mining'phenomenon," *Malwarebytes Technical Report*, 2017.

[37] XMRig, "Xmrig maximum cpu usage documentation," 2020. [Online]. Available: https://github.com/xmrig/xmrig/blob/master/doc/CPU_MAX_USAGE.md

[38] G. Mani, N. Bari, D. Liao, and S. Berkovich, "Organization of knowledge extraction from big data systems," in *2014 Fifth International Conference on Computing for Geospatial Research and Application*. IEEE, 2014, pp. 63–69.

[39] C. Olah, "Understanding lstm networks," 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[40] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[41] M. Pendleton, R. Garcia-Lebron, J.-H. Cho, and S. Xu, "A survey on systems security metrics," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, pp. 1–35, 2016.

[42] Microsoft, "Microsoft powershell diagnostics," 2020. [Online]. Available: https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.diagnostics/get-counter

[43] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[45] TensorFlow, "Tensorflow api documentation," 2020. [Online]. Available: https://www.tensorflow.org/api_docs

[46] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.