CapJack: Capture In-Browser Crypto-jacking by Deep Capsule Network through Behavioral Analysis

Rui Ning, Cong Wang, ChunSheng Xin, Jiang Li, Liuwan Zhu, and Hongyi Wu
Center for Cybersecurity Education and Research
Old Dominion University, Norfolk, VA 23529, USA

***Abstract*—This work proposes an innovative approach, named CapJack, to detect in-browser malicious cryptocurrency mining activities by using the latest CapsNet technology. To the best of our knowledge, this is the first work to introduce CapsNet to the field of malware detection through system behavioral analysis. It is particularly effective to detect malicious miners under multitasking environments where multiple applications run simultaneously. Experimental data show appealing performance of CapJack, with a detection rate of as high as 87% instantly and 99% within a window of 11 seconds.**

I. Introduction

Cryptocurrencies have gained global attention since 2017 due to the sharp surge in their exchange prices. Amid a debate on whether it is a "tulip bubble" [1] or "future economy" [2], the price of bitcoin peaked at $20,000 in Jan. 2018 [3], a stunning 20-fold increase within 12 months. The fever also spreads to other alternative coins (i.e., altcoins). According to [3], the market valuation of cryptocurrency hit 1 trillion USD in 2018. As a critical link of the value chain, transactions rely on the underlying blockchain technology called *mining*. It defines a series of processes to add transaction records to the public ledger, confirm transactions in a trustful manner and reward the participants (called miners) some "tips" for their efforts [4]. For example, bitcoin adopts the proof-of-work principle to ensure the information was difficult to make by solving a series of hash functions [4]. Due to the high cost of hardware and maintenance, businesses have been investing in cloud mining to concentrate hashpower (CPU/GPU/ASIC miners) and lease them through contracts [5].

As opposed to those centralized hashpower, if one could distribute the mining computation through hundreds of thousands devices (including datacenters, PCs, laptops, smartphones, and IoTs), it would be a lucrative business opportunity. As nefarious as it sounds, cybercriminals also think along the same line to hijack the victims' devices for mining via crypto-malwares. Different from bitcoin, which requires GPU/ASIC for mining, many altcoins such as Monero can be mined effectively by CPU [3]. The growing number of devices (both computers and embedded devices) connected to the Internet have been turned into their preys. The damage would have significant financial impact on personal and business infrastructure by causing system slowdown, reducing hardware lifespan and driving up the electric bill. Due to the anonymous nature of cryptocurrency, these malicious activities are difficult to trace. As demonstrated in Fig. 1, hackers can implant a segment of *javascript* to use the victim's device to mine cyrptocurrency without being noticed by the victim. Due to their stealthy and immediately lucrative nature, mining malware had spiked by 629% in the first quarter of 2018 as reported by McAfee [6].

Fig. 1. Procedure of crypto-jacking and profit chain.

Malware detection relies on the analysis of static signatures and dynamic behaviors [7]. Static analysis usually reverses the program to discover malicious pieces in the binaries such as API calls, file manifest, domain name and permissions. To block cryptominers, browser extensions like *No Coin* [8] and *MinerBlock* [9] use static method to detect mining scripts and blacklist the malicious sites. However, as those signature scripts can be easily obfuscated or changed, static analysis falls short to detect new/emerging patterns of crypto-malware. Dynamic analysis monitors system behaviors such as network activities because malware tends to use specialized procedures for communication. Since crypto-malware should intermittently connect to the mining pool, security analyst has been trying to find these network signatures through protocols, packets, traffic intervals, domain names, etc. However, it is challenging to differentiate the crypto-malware traffic among other types of communications since the messages are short and the malware writers can adopt a variety of obfuscation methods to blend them into normal traffic. Thus, it is rather difficult to create firewall rules to block those miners.

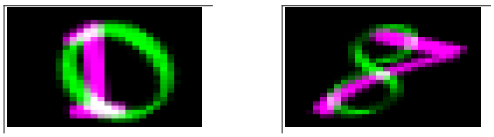
Although scripts and network signatures alone can be obfuscated, mining malware cannot escape from using a combination of computational and communication resources. To this end, this research aims to develop effective solutions to detecting mining malware based on system behaviors. We focus on a popular javascript offered by *Coinhive* [3], which mines a cryptocurrency called Monero using CPU hashpower and can be implanted into any website. Coinhive is the most prevalent malware online today, which holds the 1st place in Check Point's Top 10 Most Wanted Malware Index, with a global reach of 16 percent in April 2018 [3].

Our quest begins at a few naive approaches and ends with a highly efficient and accurate scheme based on the

Authorized licensed use limited to: National University Fast. Downloaded on February 20,2023 at 13:51:22 UTC from IEEE Xplore. Restrictions apply.

latest Capsule Network technology. First, we perform basic static and dynamic analysis to detect crypto mining using online virus/malicious scripts scanner and abnormal resource utilization. However, it turns out that these methods have high miss detection rate. A recent study [10] introduces a method to detect mining malware by analyzing websites' source code and monitoring their behavior concurrently. However, this scheme is for web-miner only and requires user to monitor the behavior of each individual webpage. Our exploration also leads to a more sophisticated mechanism based on Convolutional Neural Network (CNN) [11], which is a state-of-the-art deep learning algorithm to extract features from data. While it yields high detection rate for a single program, the performance deteriorates sharply when multiple programs are mixed – a scenario that is very common in practice since users tend to multitask by launching different programs.

The observations and lessons learnt from the preliminary exploration motivate us to adopt the latest Capsule Network (CapsNet). It is a machine learning system proposed recently by Hinton et. al. [12] to more closely mimic biological neural organization. The design is motivated by the importance of preserving hierarchical pose relationships between object parts in order to achieve correct classification and object recognition. To this end, CapsNet adds structures called capsules to a convolutional neural network and employs dynamic routing to connect capsules such that relative relationships between objects can be represented numerically as a pose matrix. Among other benefits, it can effectively recognize multiple objects even if they overlap. As demonstrated in the seminal work [12], the overlapping digits (see Fig. 2 for example) can now be recognized, which is unattainable by CNN.



(a) (b)

Fig. 2. CapsNet can effectively recognize overlapping digits. (a) 0 overlaps with 1; CapsNet output: $(0,1)$. (b) 7 overlaps with 8; CapsNet output: $(7,8)$.

Thus, we extend the original architecture of Capsule Network to detect crypto-jacking in a multi-task environment. The main contribution of this paper is summarized as follows:

- This research proposes an innovative approach to detect malicious cryptocurrency mining activities by using the latest CapsNet technology. To the best of our knowledge, this is the first work to introduce CapsNet to the field of malware detection. It is particularly effective to detect malicious miners under multitasking environments.
- Built upon the success of the CapsNet-based approach, we further develop a two-layer classification system, named *CapJack*, which can effectively transform a pre-trained model to detect miners on new devices. This is intrinsically important to achieve practical usability given the wide variety of devices used by victims.
- The work delivers a well engineered prototype. The experiments reveal valuable empirical insights into the design space for miner detection and the application of CapsNet for detecting malicious mining activities. Experimental data show the appealing performance of CapJack, with a detection rate of as high as 87% instantly and 99% within a window of 11 seconds.

The rest of the paper is organized as follows. Sec. II summarizes the preliminary explorations. Sec. III introduces the proposed scheme based on CapsNet. Sec. IV presents the experimental results. Finally, Sec. V concludes the paper.

II. Preliminary

A. Threat Model

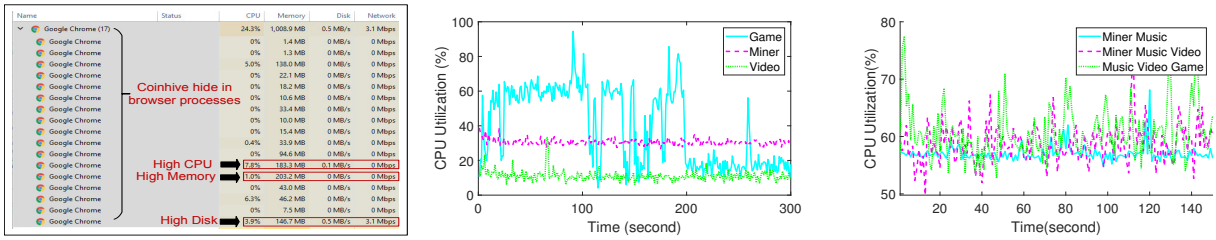
Web browsers are vulnerable to malicious mining scripts and their presence is difficult to detect. A hacker can create a mining instance within 10 lines of javascript with his CoinHive site key. Like many third-party scripts, they perform tasks in the background threads without user knowledge or permission by creating a *Worker* object. The hacker can also define the number of CPU *threads* (number of cores on the victim's machine) and *throttle* (fraction of time that threads are idle), or set threads and throttle to a smaller number to avoid detection of system slowdown. Even after the user close the browser, the attacker can still launch a hidden window under the windows taskbar to continue mining. Miners rely on *WebSockets* to open an interactive communication session between the user's browser and a server. The hacker could set up several WebSocket servers to connect their miners through the standard Stratum protocol [13] or even encrypted traffic with SSL support (from Monero v9.7), that makes the network signature difficult to be detected. In this paper, the threat model assumes the hacker has all these capabilities to achieve stealthy and effective crypto-jacking of the victim's machine.

B. System Features

Feature selection is critical to malware behavioral analysis. Since in-browser mining scripts do not attempt to inject malicious code or infect system files, it would be ineffective to use traditional features such as API calls, DLL access, and file system registry activities. Nevertheless, we discover mining is associated with a few essential features as outlined below.

- *CPU Utilization.* It indicates the sum of work handled by the CPU. Monero mining uses AES-based hash called Cryptonight algorithm [14] that efficiently utilize CPU but not GPU/FPGA/ASIC.
- *Memory.* It adopts a scratchpad with a size of the per-core L3 cache on CPUs. Therefore, the memory consumption is typically the number of threads times the L3 cache (about 2 MB on Intel CPUs).
- *Disk Read/Write.* It may take intensive disk read/write during blockchain synchronization process, which happens periodically during the mining process.
- *Network Interface.* Monero uses the Stratum protocol to communicate with the server for authorization, job submission, transactions, etc. The activities on the network interface result in subtle patterns although they are usually not obvious to be observed and recognized directly.

Authorized licensed use limited to: National University Fast. Downloaded on February 20,2023 at 13:51:22 UTC from IEEE Xplore. Restrictions apply.



(a) (b) (c)

Fig. 3. (a) Miner hiding in browser processes. (b) CPU usage of three individual applications: Game, Miner, and Video. (c) CPU usage of mixed applications.

While more system-level features in a finer granularity could be collected, in this work, we found that accurate detection of crypto-malware can be sufficiently achieved based on the combination of these high-level features that are easily accessible from task managers. This also helps the detection process minimize the input dimension and system complexity.

C. First Attempt: Task Manager and Malware Scanner

The first attempt is to use the *task manager*, hoping to find the miner among the list of running processes. While it would be straightforward for a user to perform such detection, the approach seldom succeeds because the miner hides among the browser processes. For example, we conduct an experiment by building a custom website that integrates the CoinHive scripts. As shown in Fig. 3(a), the task manager does not unveil the existence of the miner, whereas it is actually hiding in one of the 17 browser threads. It is rather difficult to tell which one is the miner since the largest thread only utilizes 7.8% CPU, and at the same time consumes less memory and network I/O than several other threads. Worse yet, on mobile devices, as people tend to leave browsers open in background, more opportunties are exploitable by attackers. Smart malware writers can even use the *navigator* class to monitor the battery charging status to avoid draining the device battery.

The second attempt is to employ malware scanners. Most scanners on the market strive to protect users from crypojacking malware by detecting the program's signatures. Malware scanners also develop their browser extensions to block in-browser mining scripts by monitoring network connections [8, 9]. We use *VirusTotal* [15], which exhaustively scans files and webpages with almost all major antivirus engines and URL blacklisting services. When we directly feed the CoinHive weblink to the scanners, merely 6 out of the total 68 scanners can detect it. When we download the CoinHive Javascript and feed it to VirusTotal, the detection rate is higher, where 17 scanners are able to detect the miner, as shown in Table I. But the detection rate decreases quickly when we apply simple obfuscation mechanisms, e.g., by using code obfuscation [16]. None of the scanners detect the miner after 2 times of obfuscations. In fact, as Coinhive becomes "famous", mining scripts are often specially crafted or reimplemented with new service domain not on the blacklist. Once the source code is obfuscated, it would be extremely difficult for static malware scanners to detect it since deobfuscation requires expertise from experienced security professionals.

TABLE I
SCANNING RESULTS OF VIRUSTOTAL (SCANNED ON 07/28/2018).

Scanners	Raw	1 × Obfuscated	2 × Obfuscated
AegisLab	Hit	Miss	Miss
Comodo	Hit	Miss	Miss
Cyren	Hit	Miss	Miss
DrWeb	Hit	Miss	Miss
ESET-NOD32	Hit	Miss	Miss
GData	Hit	Miss	Miss
Jiangmin	Hit	Miss	Miss
Kaspersky	Hit	Hit	Miss
MAX	Hit	Miss	Miss
Microsoft	Hit	Miss	Miss
Qihoo-360	Hit	Miss	Miss
Rising	Hit	Miss	Miss
Sophos AV	Hit	Miss	Miss
Symantec	Hit	Miss	Miss
TrendMicro-HouseCall	Hit	Miss	Miss
ZoneAlarm	Hit	Hit	Miss
ViRobot	Hit	Miss	Miss

D. A More Serious Approach: CNN-based Miner Detection

As our first attempt using system tools and malware scanners are unsuccessful, we turn to develop new detection techniques. When a miner runs on a device, it consumes computing resources, which is obviously what the attackers want: using the computing resources on victims' devices for mining. To this end, we attempt to achieve effective miner detection by observing and analyzing resource utilization features of CPU, memory, disk read/write and network interface I/O.

For example, Fig. 3(b) shows the CPU utilization while running these applications individually on a workstation with i5 CPU (4 cores) and 16 GB RAM. We can observe noticeable difference between the three applications, in which the miner exhibits stable resource utilization.

While the initial results look promising, further investigation soon dampens our enthusiasm. Discouraging results are observed when we mix those applications. People tend to do multitasking nowadays and operating system is built in such way to support different processes. For instance, many people browse web pages or play games while listening to music, or have the web browser running in the background while watching movie or editing document. It is common for a device to execute some applications (such as games and videos) while the miner is also running. In fact hackers love to exploit these opportunities since users tend to stay on them for long time. Fig. 3(c) illustrates the CPU utilization under three scenarios when two or more applications are running simultaneously. It is visually difficult to single out which curve

Authorized licensed use limited to: National University Fast. Downloaded on February 20,2023 at 13:51:22 UTC from IEEE Xplore. Restrictions apply.

TABLE II
SYSTEM RESOURCE UTILIZATION (WHERE C1, C2, AND C3 ARE THREE POWER SAVE MODES OF CPU).

Processor	Processor Time
	Interrupts/second
	C1 Time
	C2 Time
	C3 Time
Memory	Page Reads/second
	Page Write/second
	Page Fault/second
Network	Packets Reveived/second
	Packets sent/second
Disk	Disk Reads/second
	Disk Writes/second

corresponds to the scenario with miner.

Will machine learning techniques help identify and recognize key features that are not perceivable by human eye? Does it help by considering not only CPU but also other system parameters? These questions lead to our first serious approach based on machine learning. Previous research has considered to use machine learning techniques such as Naive Bayes and Decision Trees [17]. As those non-parametric methods have limited discriminative power, we adopt the state-of-the-art convolutional neural network (CNN) to recognize miners, as to be outlined next.

CNN has demonstrated proven success in computer vision [11, 18]. Compared to traditional learning techniques based on hand-crafted features, CNN can be trained from end-to-end to extract features automatically. Our goal is to train a CNN classifier to detect the mining process based on the runtime system data. We use *performance monitor* to gather runtime system data. We choose 5 applications for the experiment including one Coinhive miner and four common applications: music (Spotify), video (Local Video), game playing (Human: Fall Flat), and web-browsing. We run each application individually and record 12 runtime system data (as summarized in Table. II) for 30 minutes. The sampling rate is 1 Hz. Thus a dataset of 1800×12 is created for each application. CNN requires data augmentation in order to "remember" patterns in the data distribution. To this end, we use a slicing method [19] to slice the recorded data along the time dimension with a window size of 5, yielding 360 samples per application.

Several CNN architectures are experimented. The classic VGG16 architecture repetitively stacks 3×3 kernel blocks with max pooling layer. It has demonstrated proven success in learning complex relations in data [20]. We develop similar network architectures by stacking 3×3 kernels followed by max pooling layer to better suit the runtime system data that involves 12 channels and the data points on each channel is a 1D time series. The extracted feature vector is fed into a dense classifier with a softmax loss function, which classifies the applications. Due to the limits of data set, we do not use 16 layers to avoid overfitting; instead, we implement several architectures with 1 or 2 convolutional layers, plus 1 dense layer and 1 softmax layer.

The CNN models are trained in *Tensorflow* [21] with Nvidia 1080 Ti GPU. For comparison, we also implement a baseline 3-layer neural network with dense connections and a support vector machine (SVM) using LibSVM [22]. The primary performance metric is the accuracy, i.e., the fraction of correctly recognized applications. We utilize 4-fold cross validation for performance evaluation, where we randomly divide a dataset into 4 parts and use three parts for training and one for testing. This process is repeated four times such that each part is used for testing once.

TABLE III
COMPARISON OF ACCURACY FOR DIFFERENT MODELS.

Model	Single APP	Miner Detection	False Positive
SVM	0.8123	0.1977	0.2033
DNN-3	0.8025	0.2193	0.2151
VGG-3	0.9518	0.2319	0.2466
VGG-4	0.9727	0.2331	0.2452
Mix-trained CNN	NA	0.5933	0.4017
KNN-MLL	NA	0.3433	0.2263
CapsNet	0.9531	0.9895	0.0103

As shown in Table III (under the column of "Single APP"), the test accuracy of the CNN models (denoted as VGG-3 to VGG-4) can be as high as 0.97, when we consider the applications that run individually only. However, as discussed earlier, users intend to perform multitasking. To this end, we collect data of mixed applications in several common combinations such as miner-web-music, web-music, and music-game with equal quantity. The testing results based on the mixed applications are shown in the third column of Table III. As can be seen, the detection rate decreases dramatically to as low as 20%. At the same time, the false positive rate is rather high, usually more than 20%. One possible reason behind the poor performance is that the models are trained by the data of running individual applications but tested under mixed applications. Given the two data distributions are not homogeneous, the poor results are anticipated. Does it help to train the neural networks by using data based on mixed applications? More specifically, we collect a number of samples with mixed applications in various combinations and label them as including miner or not including miner. However, the highest accuracy it can achieve is still low, i.e., 59% (see Mix-trained CNN in the table).

E. Lessons Learned

The above results show that the trained CNN models achieve high accuracy in single-app detection but fail under mix-app scenarios. This can be visualized by a t-Distributed Stochastic Neighbor Embedding (t-SNE) graph [23]. t-SNE is a technique for dimensionality reduction that can be used for the visualization of high-dimensional datasets. We reshape the collected samples by using t-SNE to map them from 5×12 to 2D. Fig. 4(a) illustrates five classes in different colors. The first four classes correspond to individual applications, while the last class, i.e., mix, represents the samples of mixed-applications: game-miner-video. Fig. 4(a) illustrates that the 5 single apps are generally classifiable since they were mapped to different areas on the 2D space. However, the mixed-app samples are scattered all over the space without a clear boundary. Similar to objects that are on top of each

Authorized licensed use limited to: National University Fast. Downloaded on February 20,2023 at 13:51:22 UTC from IEEE Xplore. Restrictions apply.

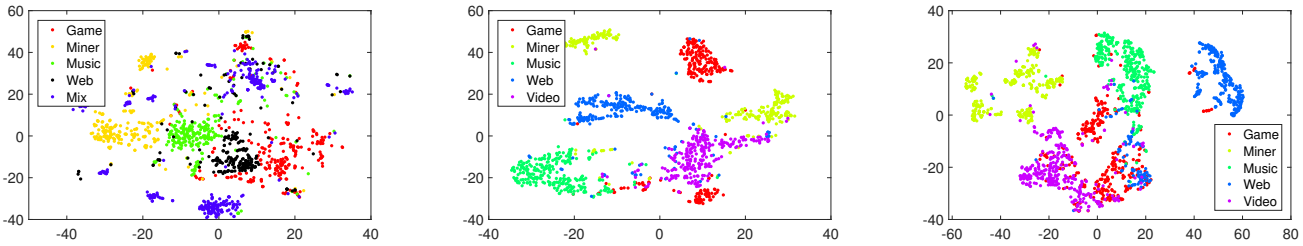


Fig. 4. (Best view in color) t-SNE visualization of raw and CapsNet-extracted features. (a) t-SNE visualization of raw features. (b) t-SNE of AppCaps layer outputs of CapsNet on the original device. (c) t-SNE of AppCaps layer outputs of CapsNet on the new device of a different model.

other, mix-app can be considered as the merge of resource utilization of the processes[1]. In this perspective, it becomes clear that a machine learning model with the capability of recognizing mixed/overlapped samples is essential to solve this complicated problem.

It is also worth pointing out the relevant work on K-Nearest Neighbor Multi-label Classifier (KNN-MLL), which obtains a multi-label vector for a testing data sample by performing a frequency count on the multi-label vectors of its k nearest neighbors [24]. Neither CNN or KNN-MLL is able to detect miner in the mix-app settings. CNN is a multi-classification algorithm where its outputs are normalized by the soft-max function to make them sum to unit. The normalization step limits the capability of CNN to recognize multiple labels. KNN-MLL attempts to identify multi-labels for a testing sample purely based on information represented in the labels of its neighbors. This approach may be sufficient for some applications, but it fails in our experiments (see the results in Table III). The good news is the latest development of CapsNet emerges to be a promising solution.

III. Miner Detection Based on CapsNet

The observations and lessons learnt from the preliminary exploration motivate us to adopt the latest Capsule Network (CapsNet) [12]. In contrast to CNN and KNN-MLL, CapsNet has a different working mechanism. It first identifies whether the learnt properties of each class are presented in a given sample, and then uses lengths of the property vectors to represent posterior probabilities for multiple classes. There is no constraint on those probabilities that they must sum to unit. CapsNet intrinsically creates a new underlying mechanism to relate spatial parts such that the neural operations are more robust, e.g., being invariant to image rotation and able to identify overlapped digits (as illustrated in Fig. 2). In close analogy, crypto miner along with other processes can be considered as mixed data distributions in space.

In the context of malicious miner detection, each data sample can be regarded as a $n \times 12$ sized image, where n is the number of sampled points. Accordingly, a scenario with mixed applications can be treated as an "image" with overlapping objects. Therefore, it is sensible to anticipate that a properly designed CapsNet would improve the miner detection rate, especially in the settings where the miner is mixed with other applications. As far as we know, this is the first work to introduce CapsNet to the field of malware detection.

[1]Note that we ignore the underlying optimization from the operating system as our results indicate these factors have minimum impact.

A. CapsNet Architecture

Although the machine learning community has not discovered generalized approaches to optimize CapsNet architecture, a well engineered system can usually be identified by manageable efforts to explore the design space. To this end, we have experimented a range of architectural options for CapsNet and arrived at a design that works well in most scenarios. In fact, our preliminary experiments show that miner detection is not highly sensitive to the CapsNet architecture.

The proposed architecture contains one convolutional layer and two capsule layers as illustrated in Fig. 5. The first layer, i.e., the convolutional layer, has 32 kernels with size of $3 \times 3 \times 1$ and stride 1, followed by ReLU activation. This layer's job is to detect basic features of the input data sample. Layer 2, i.e., the PrimaryCaps layer, has 8 primary capsules that receive the basic features detected by the previous layer and produce combinations of the features. The third layer, called AppCaps, has k capsules, one for each application. Dynamic routing is employed between capsules. The output of AppCaps is a $16 \times k$ matrix, which essentially includes k vectors, each with a size of 16. The length of a vector (i.e., the square root of sum of squares of the vector elements) will give us the probability of the presence of the corresponding application [12].

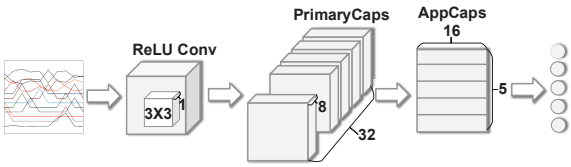


Fig. 5. CapsNet architecture.

To understand the effectiveness of the CapsNet-based approach, we have carried out preliminary experiments. Similar to the experimental setting introduced in Sec. II-D, we choose 5 applications and run each application individually to record the 12 runtime system parameters for 30 minutes at a sampling rate of 1 Hz. The data is then sliced with a window size of 5 to generate 360 samples per application. We again conduct training based on the 4-fold cross validation approach, i.e., we randomly divide the dataset into 4 parts and use three parts

Authorized licensed use limited to: National University Fast. Downloaded on February 20,2023 at 13:51:22 UTC from IEEE Xplore. Restrictions apply.

for training and one for testing. The testing result is given in Table III (see the column under Single APP). Note that this result is based on the assumption of only one application is running at a time. With no surprise, it achieves high accuracy of 0.95. But this does not necessarily make it a better solution than CNN, because our goal is to effectively detect a miner under the mix-app settings. In other words, while training is based on data by running each application individually, we anticipate the trained CapsNet to detect the miner even when it is mixed with other simultaneously running applications. To this end, we further collect 3600 mix-app samples for testing purpose only. These samples mix 2, 3, or 4 applications. As shown in Table III, the testting result is stunningly promising, with a detection rate of as high as 0.99, which is in a sharp contrast to the CNN approaches that are unable to detect more than 25% of the mining activities. At the same time, the false positive rate is as lows as 0.01. Note that it is reasonable to observe a higher "Miner Detection" accuracy than "Single App". It is because the former only targets at the miner, while the latter intends to detect all 5 classes of applications. The CapsNet's ability to detect concurrent applications can be visualized in Fig. 6. As discussed earlier, the output of the AppCaps layer is a 16×5 matrix, or 5 vectors. The length of a vector, measured by the square root of the sum of squares of its elements, indicates the probability of the presence of the corresponding application. We reshape the 16×5 matrix to a 1× 80 array for convenient visualization. In this array, the first interval (including elements 1-16) corresponds to the first vector; the second interval (i.e., elements 17-32) represents the second vector; so on and so forth. We have marked each interval by their corresponding application (as shown at the top of Fig. 6). If the length of a vector is large, we should

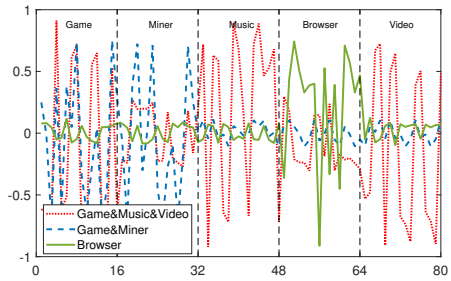


Fig. 6. Visualization of the output of AppCaps layer.

observe large absolute values in the corresponding interval. Fig. 6 illustrates three curves obtained from three experiments, i.e., by running web browser only, or running both game and miner, or running game, music and video simultaneously. The green solid line represents the sample of running web-browser only. It has dramatically larger absolute values in the 4th interval (i.e., [48,64]), showing that it belongs to the class of web browser. The greater values of the blue dashed line in intervals 1 and 2 reveal that both game and miner are running in the system. Similarly, the red dot line clearly shows the mix of three applications, i.e., game, music, and video.

B. Miner Detection across Device Models

We have demonstrated the effectiveness of CapsNet for miner detection, with a detection accuracy of as high as 99%.

TABLE IV
CAPSNET SINGLE DEVICE, CROSS DEVICE AND CROSS MODEL PERFORMANCE.

Model	Miner Detection
Single Device CapsNet 5x12	0.9895
Cross Device CapsNet 5x12	0.9633
Cross Model CapsNet 5x12	0.2108
Cross Model CapsNet 15x12	0.2174
Cross Model CapsNet 25x12	0.2091
Cross Model CapsNet with Vector Projection	0.2970
Two-layer CapJack	0.8763
Window-Based Two-layer CapJack	0.9973

While the results are encouraging, it is worth pointing out that, in the above discussion, the training and testing data are gathered from the same device. In reality, users own different devices, and worse yet, the devices are often in different models. Ideally, we want to train a CapsNet that is applicable to all devices. However, this usually results in poor performance as evidenced by the results shown in Table IV, where "Single Device" denotes the experimental setting where training and testing are carried out based on the data from a single device; "Cross Device" indicates the experiments conducted in a way that training is based on data from a device, while testing is on a different device but of the same model; "Cross Model" shows the results where training and testing are conducted on different devices in different models (e.g., training on a Dell OptiPlex 7440 and testing on a Dell Precision 5520).

As shown in Table IV, sufficiently high detection accuracy (i.e., 96%) is achievable under the Cross Device setting, because the devices are similar as long as they are in the same model. However, the Cross Model performance is deteriorated sharply, with the detection rate barely around 20%. We have explored several options to slice the testing data (with a window size of 5, 15, and 25, respectively). They all yield similar results. The poor performance is not unexpected though, given the dramatic difference between training and testing datasets since they are obtained from very different devices. It is clear that the approach to directly apply a pre-trained CapsNet on other devices in different models is ineffective.

1) Observations on Feature Clusters: The unsatisfactory results motivate us to explore possible methods to address the issue of miner detection across device models. Since CapsNet has demonstrated supreme performance on a given device, the trained CapsNet model appears capable to extract the features of individual applications and draw a precise boundary between the clusters in the feature space. Therefore, when it is applied across different devices, it is sensible to speculate that the trained CapsNet will still extract the features of the applications. However, the feature space may have been shifted, thus leading to misclassification.

To show this conjecture, we conduct an experiment by collecting samples of individual applications that run on the new device. We input them to the CapsNet model. Each output is a probability array, labeled by corresponding application. Based on our conjecture, samples of the same application will fall into a cluster in the feature space. To visualize the samples

Authorized licensed use limited to: National University Fast. Downloaded on February 20,2023 at 13:51:22 UTC from IEEE Xplore. Restrictions apply.

in the feature space, we again use t-SNE to map them to a 2-D space as illustrated in Fig. 4. Fig. 4(b) shows the t-SNE based on the samples collected on the original device, which are well clustered and have clear boundaries. Fig. 4(c) is based on the results on the new device. As can be seen, they are still well clustered, but the shapes of the clusters have been changed and their boundaries have been shifted.

2) Naive Approaches for Feature Clusters Transformation: Clearly, if we can locate the feature clusters of the new device and redraw the boundaries, we may be able to transfer a trained CapsNet model to new devices. To this end, we have explored a seemingly reasonable, but unsuccessful approach, aiming to recover the new feature boundaries using vector projection. The basic idea is to collect a small number of samples on the new device, which can be done quickly. We can even simplify the process by offering synthetic application binaries that mimic the applications' system behaviors without real installation. Then a linear transformation can be established between the feature space of the previously trained CapsNet model and the shifted feature space based on the new device. Subsequently, a sample collected on the new device can be projected back to the previously trained feature space for classification. We implement this approach and summarize its results in Table IV (denoted by "Cross Model CapsNet with Vector Projection"). As can be seen, it yields poor performance of around 30% accuracy. After a careful analysis, we discover that the vectors of individual applications are non-orthogonal in the feature space and thus the projection does not precisely preserve the classification probabilities.

3) A Two-Layer Approach to Recover Feature Boundary: As demonstrated earlier, a trained CapsNet model can effectively extract the features even when it is applied to different devices. The problem is that the shapes and boundaries of the feature clusters have been changed, so the previously trained CapsNet model cannot be directly applied to a new device. But we can employ it as the first layer, and use its output to build a second layer classifier. This approach is named *Two-Layer CapJack*. More specifically, as shown in Fig. 7, the CapsNet was trained by the data collected from device A. In order to transplant it to the device B, a small number of samples must be collected on the latter. In our experiments, as few as 50 samples can suffice the needs, which can be completed within one minute given the sampling rate of 1 Hz.

The new samples are fed to the trained CapsNet, each yielding an output that is a $16\times$ 5 matrix, or 5 vectors. The length of each vector is calculated as the square root of the sum of squares of the vector elements, which shows the probability of the presence of the corresponding application. Thus, we arrive at a 1×5 probability array. One probability array is produced for each sample. The probability arrays are labeled according to the presence of miner. Thus we can accumulate a small set of training data, which are used to train an SVM.

To classify any sample from device B, the sample will first pass through CapsNet to get the probability array, which is subsequently used as the input of the trained SVM. The output of the SVM is the probability of the sample including or not including a miner. Note that this is a few-shots model adjustment, where the number of samples collected from the new device is rather limited. Compared to other machine learning models, SVM performs better at dealing with small datasets. In the meantime, since CapsNet can effectively extract features of applications, we combine these two techniques to construct a 2-layer classification system to achieve the best performance.

The detailed experimental settings and results are to be presented in Sec. IV, but a quick look of the two-layer CapJack's performance can be found in Table IV. The miner detection rate improves dramatically to 0.88.

Note that the above discussion is based on a single sample. The accuracy can be further increased to nearly 1.0 by using a window-based two-layer CapJack approach. More specifically, instead of collecting a single sample for miner detection, we consider a time window during which the system runtime data are sampled. As to be shown in the next section, a small window (e.g., 11 seconds) would sufice to achieve high performance. The two-layer CapJack is applied to test each sample in the window. The vast majority of them (i.e., 88%) should report correct results. Thus, a majority vote is taken to determine whether a miner is present or not. Let p denote the detection rate of a single sample, then the overall detection probability in a window with n samples can be calculated as $1-\sum_{i=\lceil n/2 \rceil}^{n}(1-p)^i p^{n-i}\binom{n}{n-i}$. Our experiments verify the result and further show that the window-based two-layer CapJack enables fast and accurate miner detection.

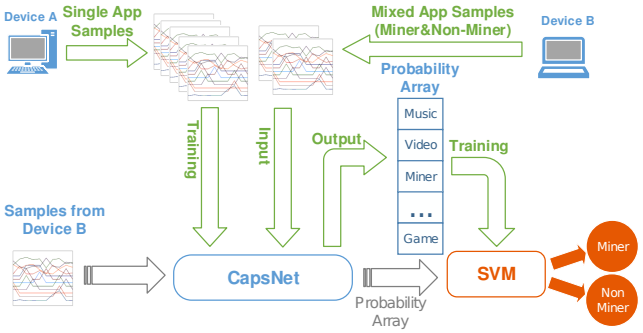


Fig. 7. The proposed two-layer classification system.

IV. Experimental Results

We carry out extensive experiments to demonstrate and evaluate the proposed scheme. Our default experimental setting includes five applications: a music player (Spotify), a video player (Local Video), a game (Human: Fall Flat), web-browsing, and the Coinhive miner. More applications (including additional miners) are used in some experiments to be discussed later. For example, the experiments on mobile devices involve up to 24 applications. To mimic the real-world malicious mining, we develop a PHP-based website which incorporates the CoinHive Javascript.

We first gather training data by running each application individually to collect 12 system runtime parameters (as summarized in Table II). We record each application for 30 minites with a sampling rate of 1 Hz. We further slice the data along the time dimension with a window size of 5, yielding 360 samples per application. Each sample is labeled by the

Authorized licensed use limited to: National University Fast. Downloaded on February 20,2023 at 13:51:22 UTC from IEEE Xplore. Restrictions apply.

corresponding application. The data collection is completed on 10 Dell workstations (Model: OptiPlex 7440) and 3 Dell laptops (Model: Precision 5520). The training is completed on a PC with i7-4770 processor and GTX-1080Ti GPU.

A series of experiments are conducted under different settings to evaluate the accuracy and robustness of the proposed scheme. The testing data are collected based on mixed applications (i.e., running multiple applications simultaneously), from the same device and different devices in different models. The detailed results and analyses are summarized below.

A. Impact of Number of Applications

In general, the classification accuracy of a machine learning model decreases with the increase of the number of classes. While the same principle is presumably applicable to CapJack too, it is worth a quantitative study to understand the robustness of the proposed scheme. To this end, we vary the maximum number of mixed applications to evaluate the miner detection rate. As shown in Table V, the proposed scheme adapts to the number of mixed applications gracefully. When the experiment is conducted on a single device, i.e., the testing and training data are gathered from the same device, the miner detection rate is maintained above 92%, even when all 5 applications are running simultaneously. Note that, there are actually many more background processes (e.g., those that are part of the operating system) running at the same time when we collect the testing data samples. The proposed approach appears very robust to such background noise and interference. Similar trend is observed when the testing is conducted across different device models, although the overall detection rate is naturally lower (ranging from 90% to 81%).

TABLE V
DETECTION ACCURACY WITH DIFFERENT NUMBER OF MIXED APPS.

Model	Number of APP	Miner Detection
Single Device CapsNet	2	0.9937
Single Device CapsNet	3	0.9849
Single Device CapsNet	4	0.9525
Single Device CapsNet	5	0.9216
Cross Model Two-Layer CapJack	2	0.9008
Cross Model Two-Layer CapJack	3	0.8841
Cross Model Two-Layer CapJack	4	0.8531
Cross Model Two-Layer CapJack	5	0.8115

B. Detection of Different Miners

The proposed scheme is trained on the CoinHive miner. The hackers may obviously utilize different miners to initiate their attacks. Can CapJack detect other similar miners without retraining? To this end, we further consider three other miners: cpuminer [25], Ufasoft miner [26], and bfgminer [27]. We collect new testing data by running them in various combinations together with other applications. As shown in Table. VI, the proposed approach (without retraining) can effectively detect the existence of the new miners with an accuracy of above 96% on a single device and 86% in the cross model settings. The results demonstrate the robustness of the proposed scheme. The promising results are attributed to the fact that although the miners can be implemented in different ways, their underlying principle remains the same (for similar crypto-currencies). Thus the runtime system parameters show similar patterns in the CapJack's feature space.

TABLE VI
DETECTION ACCURACY FOR DIFFERENT MINERS.

Model	Miner Detection
Single Device Miner A & Music	0.9777
Single Device Miner A & B & Game	0.9681
Single Device Miner B & C & Video	0.9726
Single Device Miner A &B & C & Web	0.9611
Single Device Miner A & B & C & Music & Web	0.9564
Cross Model Miner A & Music	0.9138
Cross Model Miner A & B & Game	0.9125
Cross Model Miner B & C & Video	0.8991
Cross Model Miner A & B & C & Web	0.8953
Cross Model Miner A & B & C & Music & Web	0.8620

C. Window Size in Window-Based Two-Layer CapJack

Note that Tables V & VI show the average detection accuracy based on individual samples. As introduced in Sec. III-B3, the window-based two-layer CapJack can effectively achieve perfect miner detection. More specifically, we implement an online version of the proposed scheme, which continuously record the 12 system runtime parameters, again at a sampling rate of 1 Hz. The samples within a predefined window are tested by the two-layer CapJack scheme. Then a majority vote is employed to determine if a miner is present. We vary the window size from 3 to 15 samples. The results are illustrated in Fig. 8. As can be seen, the false negatives and false positives decrease sharp to below 0.01 when the window size reaches 7. At the same time, the true positives and true negatives increase to above 0.99. This observation matches the detection probability derived in Sec. III-B3. Note that, given each sample is 5 seconds and the consecutive samples overlap 4 seconds, we need merely 11 seconds to accumulate 7 samples. Here we used the sampling rate of 1 Hz. Hence the time window is 11 seconds. If we want to detect the miner sooner, we can increase the sampling rate to reduce the window duration.

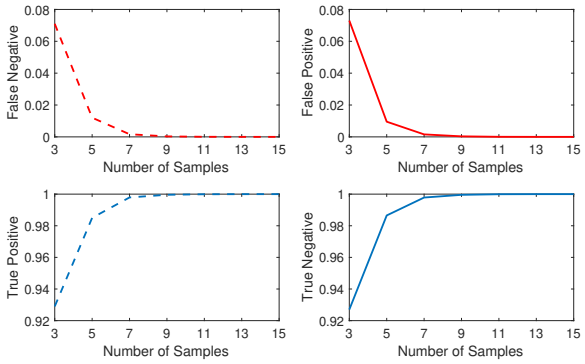


Fig. 8. Impact of window size.

D. Miner Detection on Mobile Device and Cloud Server

The above experiments are conducted based on PCs. We also test the proposed scheme on mobile devices and cloud servers, which are frequently targeted by hackers. It is worth pointing out that, although the two-layer CapJack works well in cross model settings, the different models are all PCs. Our results show that it is challenging to adapt the trained

Authorized licensed use limited to: National University Fast. Downloaded on February 20,2023 at 13:51:22 UTC from IEEE Xplore. Restrictions apply.

TABLE VII
PERFORMANCE ON MOBILE DEVICES AND CLOUDS.

Setting	Number of APP	Miner Detection
Mobile Single Device	1	0.9813
Mobile Single Device	2	0.9694
Mobile Single Device	3	0.9233
Mobile Cross Model	1	0.8839
Mobile Cross Model	2	0.8626
Mobile Cross Model	3	0.8288
AWS Single Device	1	0.9923
AWS Single Device	2	0.9796
AWS Single Device	3	0.9587
AWS Cross Model	1	0.9136
AWS Cross Model	2	0.8905
AWS Cross Model	3	0.8633

PC model to mobile devices or clouds, due to the dramatic difference between these computing platforms. Therefore, we need to train new models for them. However, it is a manageable effort, given PC, mobile and cloud are the only three vulnerable platforms frequently targeted by malicious miners.

To this end, we use 20 Samsung S7 and 5 iPhone 7 plus to collect data. We select 8 application types for training: video, browser, music, email, call, chat, miner, and game. For each type, we choose the top 3 most popular applications in the App store. Thus, a total of 24 applications are considered in the experiments. Since most mobile users do not run more than 3 applications simultaneously, we construct various experimental settings by mixing up to three applications. In each experiment, we collect the mobile device's runtime system parameters as training and testing samples.

To study the performance on cloud servers, we use Amazon Web Services (AWS) for our experiments. We chose 5 popular applications on the cloud server for sample collection: Miner, Scrapper, Web-server, Shadowsocks, and Media Streamer. We create five t2.micro and five t2.xlarge AWS EC2 ubuntu instances for running the experiments and collecting the same system runtime parameters as discussed before for PCs.

The experimental results are summarized in Table VII. As can be seen, the miner detection accuracy shows a similar trend as the results obtained from PCs, demonstrating the wide applicability of the proposed scheme on various computation platforms. When the window-based approach is adopted, we can again achieve a perfect detection accuracy.

V. CONCLUSION

In this paper we have proposed an innovative approach, named CapJack, to detect malicious cryptocurrency mining activities by using the latest CapsNet technology. To the best of our knowledge, this is the first work to introduce CapsNet to the field of malware detection. It is particularly effective to detect malicious miners under multitasking environments where multiple applications run simultaneously. Built upon the success of the CapsNet-based approach, we have further developed a two-layer classification system, which can effectively transform a pretrained model to detect miners on new devices. This is intrinsically important to achieve practical usability given the wide variety of devices used by victims. The work has delivered a well engineered prototype. The experiments have revealed valuable empirical insights into the design space and the application of CapsNet for detecting malicious mining activities. Experimental data have shown the appealing performance of CapJack, with a detection rate of as high as 87% instantly and 99% within a window of 11 seconds.

REFERENCES

[1] Bitcoin vs history's biggest bubbles. https://money.cnn.com/2017/12/08/investing/bitcoin-tulip-mania-bubbles-burst/index.html.
[2] Digital Currency Economy. https://www.forbes.com/sites/katinastefanova/2018/04/09/digital-currency-economy-what-is-the-future-of-your-bitcoins/.
[3] S. Eskandari, A. Leoutsarakos, T. Mursch, and J. Clark, "A first look at browser-based cryptojacking," *arXiv preprint arXiv:1803.02887*, 2018.
[4] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," *Communications of the ACM*, vol. 61, no. 7, pp. 95–102, 2018.
[5] Best Cloud Mining Providers of 2018. https://www.techradar.com/news/best-cloud-mining-providers-of-2018.
[6] McAfee Labs Sees Criminals "Infect and Collect" in Cryptocurrency Mining Surge. https://www.businesswire.com/news/home/20180626006679/en/McAfee-Labs-Sees-Criminals-"Infect-Collect"-Cryptocurrency.
[7] E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classification: A survey," *Journal of Information Security (JIS)*, vol. 5, no. 02, p. 56, 2014.
[8] NoCoin. https://github.com/keraf/NoCoin.
[9] MinerBlock. https://github.com/xd4rker/MinerBlock.
[10] H. Geng, Y. Zhemin, Y. Sen, Z. Lei, N. Yuhong, Z. Zhibo, Y. Min, Z. Yuan, Q. Zhiyun, and D. Haixin, "How you get shot in the back: A systematical study about cryptojacking in the real world," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2018.
[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 1097–1105, 2012.
[12] S. Sara, N. Frosst, and G. E. Hinton, "Dynamic Routing Between Capsules," in *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 3856–3866, 2017.
[13] R. Recabarren and B. Carbunar, "Hardening stratum, the bitcoin pool mining protocol," *Proceedings on Privacy Enhancing Technologies Symposium (PETS)*, vol. 2017, no. 3, pp. 57–74, 2017.
[14] CryptoNight Phylosophy. https://cryptonote.org/inside.
[15] Virus Total. https://www.virustotal.com/.
[16] Javascript Obfuscator. https://github.com/javascript-obfuscator/javascript-obfuscator.
[17] M. Graziano, D. Canali, L. Bilge, A. Lanzi, and D. Balzarotti, "Needles in a haystack: Mining information from public dynamic analysis sandboxes for malware intelligence," in *Proceedings of USENIX Security Symposium*, pp. 1057–1072, 2015.
[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.
[19] R. Ning, C. Wang, C. Xin, J. Li, and H. Wu, "DeepMag: Sniffing Mobile Apps in Magnetic Field through Deep Convolutional Neural Networks," in *Proceedings of IEEE International Conference on Pervasive Computing and Communication (PerCom)*, 2018.
[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[21] [Online] Available at:. https://www.tensorflow.org.
[22] C.-C. Chang and C.-J. Lin, "LIBSVM – A Library for Support Vector Machines," *Proceedings of ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.
[23] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.
[24] M.-L. Zhang and Z.-H. Zhou, "A k-nearest neighbor based algorithm for multi-label classification," in *IEEE-Conference on Granular Computing (GRC)*, vol. 2, pp. 718–721, 2005.
[25] cpuminer. https://github.com/pooler/cpuminer.
[26] Ufasoft Miner. http://ufasoft.com/open/bitcoin/.
[27] bfgminer. https://github.com/luke-jr/bfgminer.

Authorized licensed use limited to: National University Fast. Downloaded on February 20,2023 at 13:51:22 UTC from IEEE Xplore. Restrictions apply.