TSR-jack: An In-Browser Crypto-jacking Detection Method Based on Time Series Representation Learning

Bo Cui[1,2] and Shuai Liu[1,2]

[1] Engineering Research Center of Ecological Big Data, Ministry of Education, China
[2] College of Computer Science, Inner Mongolia University
cscb@imu.edu.cn, 32209043@mail.imu.edu.cn

Abstract. As the economic value of cryptocurrencies continues to ascend, an increasing number of cybercriminals exploit malicious browser scripts to commandeer the system and network resources of victims for unauthorized cryptocurrency mining. This form of browser-based mining, termed crypto-jacking, imparts substantial impact and harm to society, enterprises, and users. However, existing research methods lead to poor transferability because they require model training using labeling data from detection devices. To solve this problem, we introduce a crypto-jacking detection method based on time series representation learning, called TSR-jack. Specifically, we first collect hardware runtime system data for a period of time and generate datasets. After slicing the time series datasets, they are used to train a time series representation model, which introduces the loss of temporal information to extract temporal features. The temporal features can capture similarities between different devices and therefore address model transfer difficulties. Finally, the time vectors obtained from representation learning are fed into SVM for classification. Experimental results show that our TSR-jack method outperforms similar dynamic feature detection methods, especially when considering transferability, with a precision improvement of more than 5%.

Keywords: Crypto-jacking· Illegal mining· Unsupervised Learning· Time series representation.

1 Introduction

Cryptocurrency acquisition relies heavily on computing power, wherein rewards are obtained by calculating subsequent hash values. As of 2023, the global digital currency market, as reported by CoinMarketCap [1], is saturated with over 20,000 distinct coins, accumulating a total market capitalization of approximately $1.1 trillion. The accessibility and substantial value of mining digital currencies have enticed many proactive participants. However, this burgeoning market has concurrently precipitated a significant issue: numerous cybercriminals illicitly appropriate victims' resources for cryptocurrency mining. This men-

ace exploits the resources of computers or mobile devices through embedded code to facilitate cryptocurrency mining, termed crypto-jacking [2].

Common illegal mining usually takes two forms [3]. One is browser-based mining, and the other is binary-based malicious code mining. Criminals prefer spreading distributed CPU mining through browsers than implanting binary malicious code for GPU mining [4]. This methodology leverages prevalent web technologies, such as JavaScript, to execute cryptocurrency mining on web pages, utilizing the resources of the victim's system. The malicious codes are silently activated in the background when users visit infected websites. A singular mining script can effortlessly commandeer thousands of devices. An escalating number of Internet-connected devices, encompassing computers and embedded systems, have fallen prey to these illicit activities. Therefore many scholars have proposed some detection methods.

The detection methods for browser-based crypto-jacking can be categorized into static analysis and dynamic analysis. Static analysis typically involves scrutinizing website domains and script variable names. However, this approach faces challenges when miners fine-tune scripts by modifying variable names and configurations, rendering accurate identification of the majority of mining scripts on the market difficult. Dynamic analysis, on the other hand, detects the dynamic impact generated by mining scripts on the device during runtime. Nevertheless, the current implementation of this method lacks transferability across different devices, as the detection method requires the extraction of mining data specific to the device under examination. Therefore, how to design an accurate and transferability browser-based mining detection method is an urgent problem.

To solve this problem, we propose a crypto-jacking detection method based on time series representation learning (TSR-jack). Although mining scripts have varying performance impacts on different devices, we believe that the impact of mining scripts over time on the performance of devices is similar between different devices. Therefore, we introduce a time series representation learning model to capture temporal information. Firstly, we analyze which hardware performance of the device will be affected by the mining process, collect runtime system data (e.g., Processor Time) affected by the mining process over time, and design two different datasets to ensure the usability as well as transferability of the approach. Secondly, we utilize contrastive learning to introduce the loss of temporal information, consider the variation of performance indices with the mining process, and look for similarities among different mining devices. Finally, the time vectors obtained from contrastive learning are fed into a classifier for classification. Experimental results show that our TSR-jack method outperforms similar dynamic feature detection methods, especially when considering transferability, with an precision improvement of more than 5%.

Our main contributions are as follows:

1. We collect two different datasets to measure the usability as well as the transferability of the method;

2. We propose a transferability time series representation learning model with the loss of temporal information to guide contrastive learning to detecte in-browser crypto-jacking;
3. Experiments show that our propose method can effectively detect browser mining script, especially when transferability is considered, and our method has a 5% performance improvement compared to other methods.

2 Related Work

In the domain of crypto-jacking detection, prevalent methodologies are broadly categorized into two primary classifications: the static analysis method [5] and the dynamic analysis method [6]. The static analysis method discerns code features by extracting them without code execution. Conversely, dynamic analysis methodologies facilitate the execution of software within controlled environments, such as virtual machines or sandboxes. Effective extraction of dynamic feature vectors is accomplished through the vigilant monitoring of critical data pertaining to malicious code, encompassing aspects such as request permissions and machine performance transformations [7].

Within the realm of binary-based malicious code detection, Li et al. [8] amalgamate boosting and bagging techniques to classify statically extracted features. Additionally, Li et al. [9] integrate static and statistical analysis methodologies, employing n-gram models and TF-IDF for feature vector extraction, subsequently refining these vectors through classifier mechanisms. Tang et al. [10] leverage dynamic analysis methodologies to extract API call sequences, subsequently utilizing colour mapping rules to articulate feature representations indicative of malware behaviour. Berecz et al. [11] judiciously select 10 APIs and 5 dynamic link libraries relevant to mining behaviors, alongside 5 Portable Executable (PE) file attributes, utilizing machine learning to construct a discerning detection model. Karn et al. [12] employ n-gram models to distill features from API sequences, each lasting one minute, from eight mining malware instances and eight benign applications. Xiao et al.[13] propose MagTracer, a novel GPU cryptojacking detection system that leverages magnetic leakage signals emanating from GPUs.

In the context of browser-based mining detection, the static analysis method predominantly identifies characteristic codes through the scrutiny of JavaScript (JS) scripts embedded within websites. In contrast, the dynamic analysis method focuses on discerning the dynamic repercussions of mining scripts on the host system.

Qin et al. [14] posit a method to ascertain malevolent web mining behaviour by correlating web code from anomalous sites with Python regular expressions, thereby identifying mining JS scripts embedded in the background of web pages. Liu et al. [15] present a methodology to detect the static mining behaviour of browsers by scrutinizing their inherent static mining characteristics. This method involves the extraction of heap snapshots and stack code characteristics from dynamically executing browsers, leveraging known malicious mining

samples. To counteract crypto miners, browser extensions such as No Coin [16] and MinerBlock [17] employ static methodologies to detect mining scripts and blacklist nefarious websites. However, static analysis encounters challenges in detecting nascent crypto-malware patterns due to the facile obfuscation or alteration of signature scripts [18].

Dynamic analysis methodologies vigilantly monitor system behaviours, particularly network activities, given that malware often employs specialized communication procedures. Muñoz et al. [19] succinctly summarize five communication behaviour features by analyzing differentiation between normal and mining malware traffic and constructing a detection model using machine learning algorithms. Caprolu et al. [20] compare traffic from three open-source mining clients with traffic from benign software, considering packet size and interval time to articulate features. This information is subsequently amalgamated with machine learning methodologies to construct a robust detection model. Traffic-centric detection methodologies necessitate awaiting the establishment of a network connection by mining malware before detection, incurring substantial detection delays. This approach encounters challenges when combatting mining malware employing evasion techniques, such as delayed packet sending, the addition of superfluous packets, integration of proxies, and packet stuffing. Ning et al. [21] and Mani et al. [22] concentrate on assessing the impact of mining malicious scripts on the host device's performance during execution. These methodologies leverage the host's processor, memory, and additional hardware resources as data inputs for training robust detection models. However, this form of detection is susceptible to interference from other concurrent processes within the same hardware environment and exhibits limited transferability.

3 Data Collection

This section outlines the overall process of data collection. Firstly, we briefly describe the in-browser crypto-jacking workflow. Secondly, we analyze previous crypto-jacking studies and select hardware performance that is significantly affected by crypto-jacking as features. Thirdly, we design a data collection process accordingly.

3.1 Working Principle of Malicious Browser Mining

Web browsers are susceptible to crypto-jacking attacks, and their detection presents a formidable challenge. Attackers can effortlessly instantiate mining operations with less than ten lines of JS script. The workflow of in-browser crypto-jacking is depicted in Fig. 1. The preparation phase initiates with the registration of a browser cryptocurrency mining JS script (1). Subsequently, the attacker merges this script with a legitimate website to deceive the victim (2), publishing the amalgamated site on the internet (3). Following the conclusion of the preparation phase, the mining phase commences when the victim accesses the aforementioned malicious website (4).

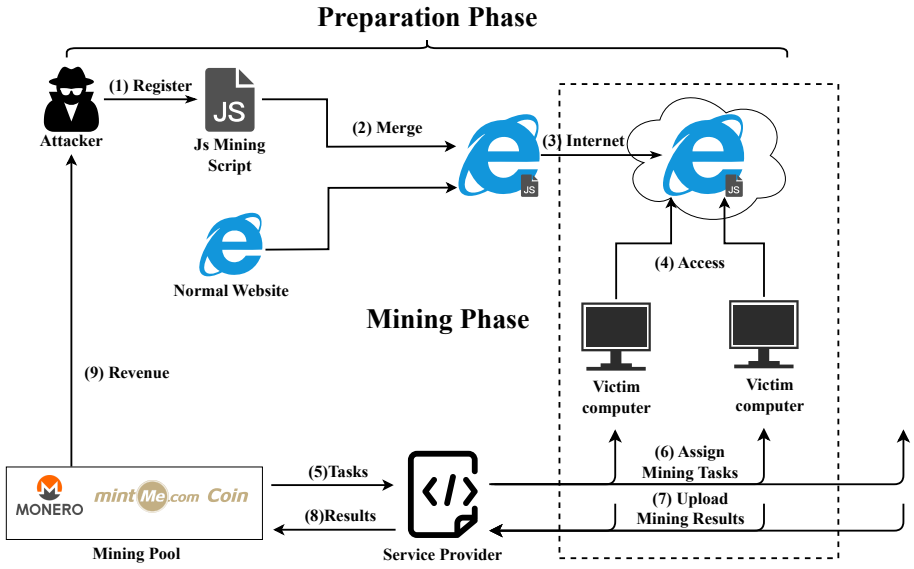

Fig. 1. In-browser Crypto-jacking Workflow

In the mining phase, the malicious site connects to the service provider via JS script, and the mining pool dispatches hash puzzle tasks to the service provider (5), which in turn forwards the tasks to the victim (6). The computed hash values are then sent back to the service provider (7), which returns the results to the mining pool (8). Ultimately, the attacker receives all earnings without any energy consumption (9).

3.2 Feature Selection

Effective feature selection is paramount in the context of detecting crypto-jacking activities. Traditional features, like API calls, DLL accesses, and file system registration activities, commonly employed for identifying malicious code injection or system file infections, prove ineffective in the specific context of in-browser mining scripts [21].

Given that in-browser mining scripts refrain from attempting to inject malicious code or compromise system files, the application of conventional features becomes inadequate. Notably, certain browser extensions, such as No Coin and MinerBlock, exhibit vulnerability to evasion strategies. Changing the name of the JS file and its corresponding references or modifying variable names within the JS file provides a simple yet effective means to evade these extensions [18]. The susceptibility of these extensions to evasion underscores their limitations in ensuring robust detection.

Furthermore, traffic-centric detection methods face significant challenges, primarily requiring a waiting period until the malicious mining software establishes

a network connection for detection. This inherent delay introduces considerable detection latency, making the detection of encrypted traffic a challenging task.

When utilizing variations in device runtime system data as features, the diverse configurations of different devices result in varying degrees of impact caused by the mining process. This limitation poses challenges in the transferability of the detection method to other devices that need to be monitored. However, these features capture the immediate impact caused by the mining process and accurately identify whether a device is actively running a mining operation. Given these considerations, we select device performance impact as a feature for dynamic analysis, providing a comprehensive assessment of the presence of mining tasks.

Our selection of device hardware features is as follows:

- CPU Utilization. CPU utilization refers to the percentage of time the CPU is actively executing tasks in a computer. Browser-based mining utilizes the Cryptonight algorithm [23], which is an AES-based hashing algorithm designed to efficiently leverage the CPU, as opposed to GPU/FPGA/ASIC.
- Memory. Memory is employed to store programs and data actively running on a computer. Anomalies in memory usage may suggest the presence of a potential mining process, given that mining typically requires substantial memory for executing related computational tasks.
- Network Interface. Browser-based mining employs the Stratum protocol to communicate with servers for authorization, job submission, transactions, etc. Irregularities in network interface activity may indicate mining-related operations.
- Disk Read/Write. During the blockchain synchronization process, mining operations may lead to intensive disk read/write activities. Such occurrences periodically happen throughout the mining process.

3.3 Datasets Design

During the data collection process, we need to ensure that the collected tools minimally affect the hardware performance of the device itself, so we utilize Performance Monitor, a performance monitoring tool that comes with windows. We collected 12 runtime system data related to CPU, Memory, Disk, and Network Interface as shown in Table 1.

Based on the existing research challenges, we design two datasets to evaluate the effectiveness and transferability of the methods, as illustrated in Fig. 2. The first dataset, aimed at testing the method's effectiveness, consists of runtime data from 7 devices: 5 devices of the same model and 2 devices of another model, referred to as the homogeneous dataset. The second dataset, intended to validate the method's transferability, comprises runtime data from 10 devices, each with a different model, including desktops and laptops, with varying CPUs from both Intel and AMD, referred to as the heterogeneous dataset.

Before formally collecting data, we design a mining webpage to simulate crypto-jacking, using the CoinIMP mining script. During data collection, we design two scenarios. The first scenario involves users using their devices normally,

Table 1. Runtime System Data.

Hardware	Parameter	Detailed information
cpu	Processor Time	Refers to the proportion of time the processor (CPU) spends executing computational tasks within a given period
	Interrupts/second	Indicates the number of interrupts occurring per second
	C1 Time	Time spent in the C1 power state
	C2 Time	Time spent in the C2 power state
	C3 Time	Time spent in the C3 power state
Memory	Page Reads/second	the rate at which pages of data are read from the disk into the computer's memory (RAM) per second
	Page Writes/second	the rate at which pages of data are written from the computer's memory (RAM) to the disk per second
	Page Fault/second	the rate at which these page faults occur per second
Network	Packets Received/second	the rate at which network packets are received by the network interface per second
	Packets sent/second	the rate at which network packets are transmitted or sent by the network interface per second
Disk	Disk Reads/second	the rate at which data is read from the disk per second
	Disk Writes/second	the rate at which data is written to the disk per second

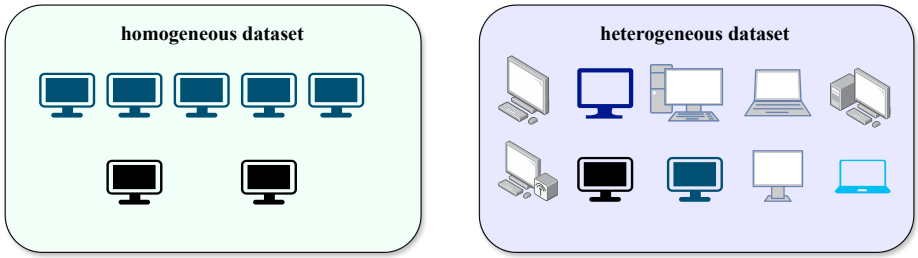

Fig. 2. Homogeneous Dataset and Heterogeneous Dataset

and we observe their activities, including listening to music, browsing the internet, playing games, and utilizing the device for machine learning model training. The second scenario involves users using their devices normally while also having our pre-designed mining webpage open. During the data collection process, we set the sampling rate to 1 Hz and collect data for 2 hours. For each device in each scenario, we obtain data in the format of 7200*12.

4 Problem Formulation

System runtime data is a visual representation of the hardware usage of a device. The value varies from device to device, but it demonstrates the changes caused by the mining script. It is difficult to disguise and deceive, so the mining process can be effectively detected based on the runtime data.

Given a set $\langle \mathcal{X}, \mathcal{F} \rangle$,where $\mathcal{X} = \{x_1, x_2, \cdots, x_Q\}$ denotes a Q time series, $\mathcal{F} = \{f_1, f_2, \cdots, f_N\}$, denotes the set of N selected runtime features. The goal is to learn a nonlinear embedding function f_θ that maps each x_i to its representation

r_i that best describes itself. The representation $r_i = \{r_{i,1}, r_{i,2}, \cdots, r_{i,T}\}$ contains representation vectors $r_{i.t} \in \mathbb{R}^K$ for each timestamp t, where K is the dimension of representation vectors.

5 Miner Detection Model

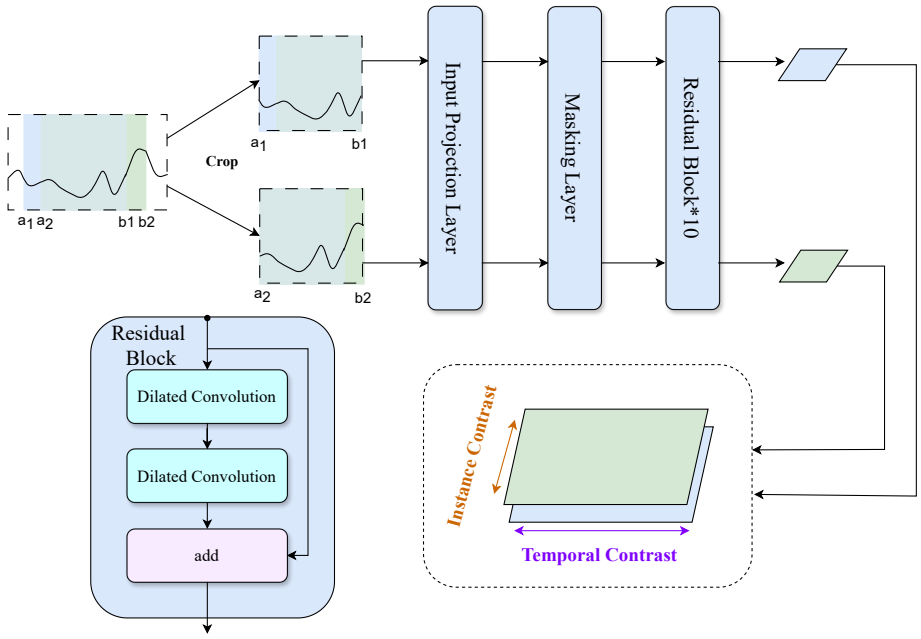

Fig. 3. Model Architecture

This section proposes a crypto-jacking detection method based on time series representation learning, called TSR-jack. The overall architecture of the model is shown in Fig. 3. We present the overall process of the time series $\mathcal{X}$ corresponding to the f_i feature. Initially, we crop the time series $\mathcal{X}$ to generate two segments with overlapping sub-sequences as positive instances. These trimmed segments are then input into the model for joint optimization using temporal and instance contrastive loss. The model comprises three components: an input projection layer, a masking layer, and residual blocks.

5.1 Input Projection Layer

For every input X_i, the input projection layer serves as a fully connected layer, skillfully mapping the feature X_i at timestamp t to a high-dimensional vector

z_i. The input projection layer also maps the number of features n of F to K, which facilitates feature extraction in subsequent residual blocks.

5.2 Masking Layer

The subsequent masking layer, a pivotal component in our approach, introduces a random selection and masking of high-dimensional vectors Z, thereby creating an enriched and augmented contextual perspective. This strategy enhances the model's ability to capture intricate relationships within the data. The latent vector $z_i = z_{i,t}$ following the Input Projection Layer is specifically masked along the time axis using a binary mask $m \in \{0,1\}^T$, where the elements are independently sampled from a Bernoulli distribution with p = 0.5. These masks are sampled independently during each forward pass of the encoder.

5.3 Residual Blocks

To further refine the contextual representations, we employ dilated convolutions in conjunction with ten residual structures. These structures are organized into blocks, with each block comprising two 1-D dilated convolutional layers. The strategic use of dilated convolutions is informed by their ability to offer an expanded receptive field, spanning diverse domains [24]. This architectural choice facilitates the extraction of nuanced contextual features across different aspects of the input data, contributing to the model's capacity for comprehensive understanding and representation of temporal dynamics.

5.4 Temporal and Instance Contrastive loss

To efficiently capture the time series information, we compute the loss of contrast between the time dimension and the instance dimension. In the input time series, segments with overlapping subsequences are considered as positive and segments without overlapping subsequences are considered as negative. Assuming i as the index of the input time series sample and t as the timestamp, representations $r_{i,t}$ and $r'_{i,t}$ at the same timestamp t but from two enhancements of x_i are denoted. The temporal contrast loss for the i-th time series at timestamp t can be formally expressed as:

$$\ell_{temp}^{(i,t)} = -\log \frac{\exp(r_{i,t} \cdot r'_{i,t})}{\sum_{t' \in \Omega} \left(\exp(r_{i,t} \cdot r'_{i,t'}) + \mathbb{I}_{[t \neq t']} \exp(r_{i,t} \cdot r_{i,t'}) \right)}, \tag{1}$$

where Ω represents the set of timestamps within the overlap of the two subseries, and $\mathbb{I}$ is the indicator function. The contrast loss at the instance level is denoted as:

$$\ell_{inst}^{(i,t)} = -\log \frac{\exp\left(r_{i,t} \cdot r'_{i,t}\right)}{\sum_{j=1}^{B} \left(\exp(r_{i,t} \cdot r'_{j,t}) + \mathbb{I}_{[i \neq j]} \exp(r_{i,t} \cdot r_{j,t}) \right)}, \tag{2}$$

where B denotes the batch size. We use other time series representations with timestamp t as negative samples in the same batch.

These two types of losses are complementary. Instance contrast provide insight into specific features, while temporal contrast aim to uncover dynamic trends over time. The total loss is defined as:

$$\mathcal{L}_{dual} = \frac{1}{QT} \sum_{i} \sum_{t} \left(\ell_{temp}^{(i,t)} + \ell_{inst}^{(i,t)} \right). \tag{3}$$

6 Experiments

This section demonstrates the effectiveness as well as the transferability of evaluating time series representation learning models for mining detection through extensive experiments.

6.1 Datasets

While considering both effectiveness and transferability, we need to ensure timely detection. Once a device is invaded by crypto-jacking, our method should be able to identify it immediately. Therefore, we slice the homogeneous and heterogeneous datasets collected in data collection. Along the time dimension, we divide the data into 5 time slices, resulting in each data point having a shape of 5*12, where 5 represents the length of the time series and 12 represents the number of features. Thus, for each scenario and each device, the data shape is (1440*5*12). Subsequently, we input the segmented data into the model for training.

6.2 Baseline Methods

As shown in Fig. 3, we propose a mining detection model incorporating temporal information. Our experiments comprehensively compare our proposed method with three potential baseline methods: Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Capjack.

1. *CNN*[25]: This method is a deep learning technique that is particularly suitable for processing image and video data and has been widely used in the field of computer vision. Its core idea is to gradually extract and learn features from data through components such as convolutional layers, pooling layers, and fully connected layers, thereby achieving the tasks of classification, recognition, and analysis of complex data. Our input data shape is (5*12), which can be viewed as a single-channel image. We use two convolutional layers for feature extraction and a fully connected dense layer for classification purposes.
2. *RNN*[26]: This method is also a type of deep learning method, mainly used for processing sequential data such as text and time series. Unlike CNN, RNN have memory capabilities, allowing them to capture temporal dependencies

in the data. This is crucial for capturing time-related features that may be involved in mining activities. The first layer of the RNN is an embedding layer, followed by a one-dimensional convolutional layer with ReLU activation. Then, an RNN layer is used, followed by a fully connected dense layer that flattens the output. The final output layer is another fully connected dense layer for classification.

3. *Capjack*[21]: This method is a pioneering work that introduces capsule networks [27] in the field of crypto-jacking detection. We duplicate the work of this paper. Firstly, they use a convolutional layer for the first step of feature extraction. Secondly, they use a main capsule layer for specific feature extraction, and a dynamic routing algorithm to learn the relationships between different capsules and improve the generalization ability and robustness of the model. Although the method allows for crypto-jacking detection, in order to achieve transferability, they had to extract a specific labeled data from the test device and feed it into the capsule network, and the resulting feature vector was used to train the classifier. This resulted in the need for labeled data extraction as well as separate training of classifiers for each device to be detected.

6.3 Performance Evaluation Metrics

To evaluate the performance of different methods for malicious mining detection, we consider three evaluation metrics, namely, Precision, Recall, and F-score. We repeat the experiments and report the average results. The three metrics are defined as follows:

$$\text{Precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}}$$

$$\text{Recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}$$

$$\text{F-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

6.4 Experiment settings

In the experiment, in the Input Projection Layer, for each feature f_j, it is mapped at each time step in the time series $\mathcal{X}$ to a sequence r_i with T elements, where the feature set $\mathcal{F} = \{f_1, f_2, \cdots, f_N\}$ has N=12 elements, the time series set $\mathcal{X} = \{x_1, x_2, \cdots, x_Q\}$ has Q=5 elements, the represented time series $r_i = \{r_{i,1}, r_{i,2}, \ldots, r_{i,T}\}$ has T=10 elements. During training, the masking layer uses a binary mask $m \in \{0, 1\}^T$, where the elements are independently sampled from a Bernoulli distribution with p = 0.5. The residual module utilizes dilated convolution for feature extraction, with a dilation convolution kernel size of 3. Finally, the model parameters are updated guided by temporal and instance

Table 2. Performance comparison of different datasets.

Method	Homogeneous Dataset			Heterogeneous Dataset		
	P	**R**	**F**	**P**	**R**	**F**
CNN	0.973	0.905	0.934	0.753	0.612	0.642
RNN	**0.993**	0.860	0.862	0.784	0.773	0.705
Capjack	0.914	0.938	0.914	0.776	0.866	0.806
TSR-jack	0.985	**0.950**	**0.965**	**0.837**	**0.885**	**0.851**

loss. After the model training is complete, we train the classifier SVM with the model-encoded time series representation matrix. All our models were trained on an NVIDIA A40-6Q GPU based on PyTorch .

During the experiment, we observe significant differences between devices. As a result, we do not use the traditional random shuffle split method to divide the data into training and testing sets. Instead, we partition the data into training and testing sets based on the devices, using $n-1$ devices for training and 1 device for testing. We repeat this process n times in Cross-Validation and calculate the average.

6.5 Performance on Homogeneous Data

To evaluate the effectiveness of our proposed method and the selected features, we first tested four methods on a homogeneous dataset and presented the results in the left half of Table 2. The results show that all four methods performed well, with an average of all three metrics exceeds 0.9. This indicates that the features we selected are effective and that mining programs do indeed impact hardware performance metrics. Specifically, CNN showed relatively balanced performance, with all three metrics at a medium to high level among the four methods. On the other hand, RNN's performance was somewhat extreme, with a very high precision of 0.993 , indicating its ability to detect all mining samples . However , it had the lowest Recall and F-score, suggesting that RNN misclassified many normal samples as mining samples, resulting in a high false positive rate. The Capjack method exhibited more balanced performance and had a higher Recall value compared to CNN and RNN methods. Our proposed method demonstrated the highest Recall and F-score on the homogeneous dataset, with Precision just below RNN by 1%. This demonstrates that our method can effectively detect mining scripts with a low false positive rate, making it an effective approach.

6.6 Performance on Heterogeneous Data

To assess the transferability of our proposed method, we select devices of different devices to construct a heterogeneous dataset and test four methods on this dataset. The experimental results are displayed in the right half of Table 2. The RNN model performs significantly better than the CNN model. This is attributed to considering temporal changes, which can reduce differences between

Table 3. Performance on WebMinePool.

Mining Script	WebMinePool											
Threads	1			2			3			4		
Models	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
CNN	0.39	0.133	0.199	0.305	0.091	0.141	0.769	0.692	0.728	0.807	0.875	0.84
RNN	0.927	0.858	0.891	0.932	0.925	0.929	0.908	0.658	0.763	0.93	0.883	0.905
Capjack	0.772	0.571	0.656	0.738	0.111	0.194	0.822	0.358	0.5	0.732	0.906	0.81
TSR-jack	**0.97**	**0.95**	**0.96**	**0.971**	**0.983**	**0.977**	**0.971**	**0.978**	**0.974**	**0.971**	**0.98**	**0.975**

Table 4. Performance on BrowserMiner.

Mining Script	BrowserMiner											
Power	10			30			50			70		
Models	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
CNN	0.457	0.175	0.253	0.827	**1**	0.905	0.827	**1**	0.905	0.828	**1**	0.906
RNN	0.925	0.825	0.872	0.937	**1**	0.967	0.937	**1**	0.967	0.937	**1**	0.968
Capjack	0.772	0.775	0.774	0.897	0.622	0.735	0.88	0.562	0.686	0.699	0.991	0.82
TSR-jack	**0.971**	**0.983**	**0.977**	**0.971**	0.989	**0.98**	**0.971**	0.989	**0.98**	**0.971**	0.989	**0.98**

devices. The Capjack method extracts a small subset of data from the test set for training the classifier. Despite not considering temporal changes, it still performs well, with a decrease in false positives, and outperforms RNN in terms of Recall and F-score. Our method achieves the highest values across all three metrics, with Precision increasing by 5%, Recall by 2%, and F-score by 4.5%. After training on the training set, our method can be directly used to test if devices are compromised by browser-based cryptojacking.

6.7 Performance on Different Scripts

In addition to transferability between devices, consider the ability to detect between different mining scripts. To ensure that our approach remains detectable across different mining scripts, we investigate browser mining scripts that are still publicly available and select two of them[18], WebMinePool and BrowserMiner, and conduct a series of experiments. WebMinePool allows for the configuration of thread values, indicating the number of threads simultaneously executing the mining algorithm. Each thread represents a parallel computing unit capable of executing computing tasks concurrently, thus enhancing mining efficiency. We tested thread values of 1, 2, 3, and 4 respectively. Meanwhile, BrowserMiner can configure power values, representing the percentage of system resources occupied by the mining algorithm. The higher the percentage of resource occupation, the faster the algorithm's computation speed. We tested power values of 10, 30, 50, and 70 respectively. We simulated malicious cryptojacking on a new device and collected data continuously for 10 minutes for each type of sample for detection. Using the heterogeneous dataset as the training set, we trained the model and evaluated it using the newly collected data as the test set.

We initially conducted experiments using the WebMinePool script, simulating different intrusion scenarios by changing the thread parameters (1, 2, 3, 4). As shown in Table 3, the four methods exhibited varying degrees of perception regarding the system performance changes caused by this mining script. At thread values 1 and 2, where the script had minimal impact on device performance, CNN, a feature-extraction method, could hardly detect the mining script. At thread values 3 and 4, CNN's performance improved. Capjack, due to its need for a brief test set for classifier training, could mitigate some transferable issues, thus performing better than CNN. However, at thread values 2 and 3, Capjack had a very low Recall value, which we speculate was due to the test set used in training the classifier lacking strong representativeness. RNN and TSR-jack, considering temporal information, outperformed the other methods, with TSR-jack showing the best results. It's worth noting that adjusting thread parameters had different impacts on each method, with our proposed method being relatively unaffected.

Subsequently, we conducted experiments using the BrowserMiner script, simulating real intrusion scenarios by adjusting the power parameters (10, 30, 50, 70). As shown in Table 4, the experimental results were similar to the previous discussions. The perception of the four methods regarding the system performance changes caused by the mining script remained varied. CNN performed the worst, followed by Capjack, RNN capturing temporal information ranked second, and our proposed method performed the best. Additionally, adjusting power parameters had varying impacts on each method as well. It's noteworthy that when the power value was 10, there was a significant difference in performance among the four methods. And when the power value is other values, there is not much difference in the performance of the four methods, even the Recall value of RNN and CNN is 1, which indicates that these two methods successfully identify all the mining scripts. Malicious mining scripts often lower these parameters to achieve stealthiness.

7 Conclusion

We propose a novel in-Browser crypto-jacking detection method, TSR-jack, which efficiently captures the temporal transformation characteristics of device runtime data. We construct a homogeneous dataset and a heterogeneous dataset. The effectiveness of the method and the transferability between devices is verified respectively. Two different mining scripts are utilized for script detection transferability experiments. We achieve high accuracy and a low false alarm rate, which improves the poor transferability problem in previous studies to a great extent. With this work, we hope to stimulate more research on the direction of utilizing deep learning, time series representation for crypto-jacking detection.

ACKNOWLEDGMENT

This paper is supported by the National Natural Science Foundation of China (61962042), and Natural Science Foundation of Inner Mongolia (2022MS06020), and the Central Government Guides Local Science and Technology Development Fund (2022ZY0064), and the University Youth Science and Technology Talent Development Project (Innovation Group Development Plan) of Inner Mongolia A. R. of China (Grant No. NMGIRT2318), and the Fund of Supporting the Reform and Development of Local Universities (Disciplinary construction).

References

1. Coinmarketcap. [Online], https://interest.coinmarketcap.com/
2. Hong, G., Yang, Z., Yang, S., Zhang, L., Nan, Y., Zhang, Z., Yang, M., Zhang, Y., Qian, Z., Duan, H.: How you get shot in the back: A systematical study about cryptojacking in the real world. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1701–1713 (2018)
3. Jayasinghe, K., Poravi, G.: A survey of attack instances of cryptojacking targeting cloud infrastructure. In: Proceedings of the 2020 2nd Asia pacific information technology conference. pp. 100–107 (2020)
4. securelist: The state of cryptojacking in the first three quarters of 2022 (2022)
5. Nath, H.V., Mehtre, B.M.: Static malware analysis using machine learning methods. In: Recent Trends in Computer Networks and Distributed Systems Security: Second International Conference, SNDS 2014, Trivandrum, India, March 13-14, 2014, Proceedings 2. pp. 440–450. Springer (2014)
6. Willems, C., Holz, T., Freiling, F.: Toward automated dynamic malware analysis using cwsandbox. IEEE Security & Privacy **5**(2), 32–39 (2007)
7. Tekiner, E., Acar, A., Uluagac, A.S., Kirda, E., Selcuk, A.A.: Sok: cryptojacking malware. In: 2021 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 120–139. IEEE (2021)
8. Li, S., Li, Y., Han, W., Du, X., Guizani, M., Tian, Z.: Malicious mining code detection based on ensemble learning in cloud computing environment. Simulation Modelling Practice and Theory **113**, 102391 (2021)
9. Li, S., Jiang, L., Zhang, Q., Wang, Z., Tian, Z., Guizani, M.: A malicious mining code detection method based on multi-features fusion. IEEE Transactions on Network Science and Engineering (2022)
10. Tang, M., Qian, Q.: Dynamic api call sequence visualisation for malware classification. IET Information Security **13**(4), 367–377 (2019)
11. Berecz, G.J., Czibula, I.G.: Hunting traits for cryptojackers. In: ICETE (2). pp. 386–393 (2019)
12. Karn, R.R., Kudva, P., Huang, H., Suneja, S., Elfadel, I.M.: Cryptomining detection in container clouds using system calls and explainable machine learning. IEEE Transactions on Parallel and Distributed Systems **32**(3), 674–691 (2020)
13. Xiao, R., Li, T., Ramesh, S., Han, J., Han, J.: Magtracer: Detecting gpu cryptojacking attacks via magnetic leakage signals. In: Proceedings of the 29th Annual International Conference on Mobile Computing and Networking. pp. 1–15 (2023)
14. Qin, Y., Liu, L., Gao, H., Liu, S.: Detection and prevention of malicious mining behaviors on web pages. Network Security Technology and Application **216**(12), 54–56 (2018)

15. Liu, J., Zhao, Z., Cui, X., Wang, Z., Liu, Q.: A novel approach for detecting browser-based silent miner. In: 2018 IEEE Third International Conference on Data Science in Cyberspace (DSC). pp. 490–497. IEEE (2018)
16. keraf: Nocoin. [Online], https://github.com/keraf/NoCoin
17. xd4rker: Minerblock. [Online], https://github.com/xd4rker/MinerBlock
18. Rajba, P., Mazurczyk, W.: Limitations of web cryptojacking detection: A practical evaluation. In: Proceedings of the 17th International Conference on Availability, Reliability and Security. pp. 1–6 (2022)
19. i Muñoz, J.Z., Suárez-Varela: Detecting cryptocurrency miners with netflow/ipfix network measurements. In: 2019 IEEE International Symposium on Measurements & Networking (M&N). pp. 1–6. IEEE (2019)
20. Caprolu, M., Raponi, S., Oligeri, G., Di Pietro, R.: Cryptomining makes noise: Detecting cryptojacking via machine learning. Computer Communications **171**, 126–139 (2021)
21. Ning, R., Wang, C., Xin, C., Li, J., Zhu, L., Wu, H.: Capjack: Capture in-browser crypto-jacking by deep capsule network through behavioral analysis. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications. pp. 1873–1881. IEEE (2019)
22. Mani, G., Pasumarti, V., Bhargava, B., Vora, F.T., MacDonald, J., King, J., Kobes, J.: Decrypto pro: Deep learning based cryptomining malware detection using performance counters. In: 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS). pp. 109–118. IEEE (2020)
23. Cryptonight phylosophy. [Online], https://cryptonote.org/inside
24. Bai, S., Kolter, J., Koltun, V.: An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv: Learning,arXiv: Learning (Mar 2018)
25. Kim, Y.: Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882 (2014)
26. Elman, J.L.: Finding structure in time. Cognitive science **14**(2), 179–211 (1990)
27. Sabour, S., Frosst, N., Hinton, G.E.: Dynamic routing between capsules. Advances in neural information processing systems **30** (2017)