International Journal of Safety and Security Engineering
Vol. 16, No. 1, January, 2026, pp. 141-151
Journal homepage: http://iieta.org/journals/ijsse

Cybersecurity Approaches to Cryptojacking Detection: A Computer Application Perspective

Alaa K. Faieq, Mohammad M. Rasheed*

College of Engineering, University of Information Technology and Communications, Baghdad 10013, Iraq

Corresponding Author Email: mohammad.rasheed@uoitc.edu.iq

https://doi.org/10.18280/ijsse.160112

Received: 15 November 2025
Revised: 8 January 2026
Accepted: 18 January 2026
Available online: 31 January 2026

Keywords:
cryptojacking detection, machine learning, Random Forest, Support Vector Machine, Multilayer Perceptron, time-aware validation

ABSTRACT

Cryptojacking is the covert use of computing resources for cryptocurrency mining; it is a rising threat to a variety of different environments. This paper compares three supervised classifiers, i.e., Random Forest (RF), Support Vector Machine (SVM, RBF kernel), and Multilayer Perceptron (MLP), for the cryptojacking detection on a public dataset of 95,312 time-stamped observations (58 numeric features). An anti-leakage preprocessing pipeline is enforced, and models are evaluated under stratified random splitting, chronological splitting, cross-validation, and rolling-origin time validation. Under random splitting, all classifiers get near-perfect accuracy (RF: 1.0000, SVM: 0.9999, MLP: 1.0000). Under chronological splitting, both RF and MLP maintain their performance, while SVM shows a decrease in recall compared to before, highlighting the importance of time-aware validation. All errors are false negatives only. Area Under the Precision-Recall Curve (PR-AUC) and a majority class baseline are reported to deal with the class imbalance. RF feature importance analysis shows that memory and filesystem are the key metrics in classification, which can be used to guide lightweight monitoring. Efficiency benchmarks are supplied to aid in deployment-focused model selection.

1. INTRODUCTION

Cryptojacking is a form of malicious behavior that involves using a victim's computing resources without the victim's consent to mine cryptocurrencies. Unlike traditional ransomware, cryptojacking is not self-identifying by encrypting data and is therefore stealthy and persistent, and is slowly eating through Central Processing Unit (CPU) cycles, memory, and energy while reducing system performance. This makes cryptojacking an efficient method of monetary gain for attackers, particularly when it is implemented in cloud providers, container clusters, or enterprise endpoints, where there is access to abundant compute resources and a common occurrence of misconfiguration [1-3]. To evaluate machine-learning-based cryptojacking detection under reproducible conditions, this study uses a publicly available time-series dataset of process observations labeled as normal or cryptojacking [4]. Besides direct loss of performance, cryptojacking may raise operational costs (e.g., increased energy consumption or increased cloud bills) and may act as a lead-up to more general compromise since it shows an attacker has already achieved code execution and persistence.

Cryptojacking is available in multiple vectors. Web-based cryptojacking is used in the browser environment with the help of JavaScript or WebAssembly and mines while the user is visiting a malicious or compromised site [5, 6]. Host-based cryptojacking involves the use of mining binaries or injecting mining code into legitimate processes, often accompanied by persistence mechanisms, privilege escalation, and evasion techniques [1, 2]. Cloud and container environments are an enticing target as exposed services, unsecured dashboards, and weak credentials can give access, and the mining activity can be masked in fluctuating background workloads [3, 7, 8]. Attackers also communicate with mining pools using protocols like Stratum, which can be tunneled or proxied to avoid network monitoring [9].

Cryptojacking is hard to detect as the miners are trying to remain stealthy, deployments are spread across many different environments, and resource-usage patterns can interleave with legitimate compute-intensive workloads. Traditional methods of signature-based detection can be circumvented with obfuscation, packing, and frequent code updating [10, 11]. Network-based approaches may prove limited when communications are encrypted, proxied, or embedded within seemingly normal traffic [9, 12]. Consequently, a lot of recent works investigate behavioral indicators and machine learning to distinguish between cryptojacking and normal behavior based on features extracted from system calls, CPU and memory statistics, hardware performance counters, or network-flow characteristics [7, 13-15]. A key question is how well such models generalize across datasets and operational conditions, and which features provide robust discriminatory power.

Many machine learning studies use random splitting for the purpose of evaluation, but if the observations are time-dependent or made under changing conditions of the system, random splitting causes inflation of the performance because it allows temporally correlated patterns to appear in the

training and testing sets. Therefore, assessing models by a chronological split is a more realistic indication of the generalization over time, and is more representative of the operational deployment conditions.

This paper examines the use of machine learning for cryptojacking detection using the supervised learning approach from a computer application perspective. We apply 3 popular classifiers, RF, SVM, and MLP, to a publicly available time series dataset of process observations labelled as normal or cryptojacking [4]. Features are preprocessed (missing values and normalization), and performance is measured with normal metrics and cross-validation. We also analyze RF feature importance to find out which attributes have the most importance in the classification. The work provides a reproducible benchmark on a public dataset and a controlled comparison between complementary supervised learning families (ensemble, margin-based, and neural network) with interpretability through feature importance analysis.

This work makes four main contributions: (1) evaluation of RF, SVM, and MLP models for cryptojacking detection based on available feature set, (2) report of standard classification metrics, Area Under the Precision-Recall Curve (PR-AUC) for imbalanced evaluation and allow for interpretability outputs (e.g., feature importance where applicable), (3) benchmarking on efficiency indicators relevant to efforts made on deployment, e.g., training time and inference latency and footprint, and (4) comparison of random vs chronological splitting and rolling-origin time validation to evaluate robustness with time-aware validation.

The rest of this paper is organized as follows. Section 2 revises the works related to cryptojacking detection, i.e., host-based, network-based, and Machine Learning (ML)-based approaches, and problems such as evasion and concept drift. Section 3 is about the description of the dataset, preprocessing, model configuration, and evaluation protocol. Experimental results, model performance, and feature importance are reported in Section 4. The implications, limitations, and recommendations for deployment are discussed in Section 5. Section 6 concludes the paper and suggests directions for future research.

2. RELATED WORK

The problem of cryptojacking has attracted an impressive amount of research as a result of the speed at which it has proliferated, the relative lack of risk taken by threats, and the fact that detection of such activity reliably remains a difficult task in a variety of environments. Existing literature covers this area from a variety of complementary viewpoints, which include empirical measurement and profiling of cryptojacking campaigns, browser and host-centric detection methods, security aspects specific to cloud and containerized environments, and classification frameworks based on machine learning technologies.

2.1 Behavioral and semi-supervised approaches

Konoth et al. [10] developed a more reliable solution known as MineSweeper compared to those that employ blacklists. It involved using CPU consumption to discover cryptomining that it was hidden in the browser and served as a way to mine cryptocurrency and generate profit without the knowledge of its users, as well as utilized obfuscation and the latest web technologies to avoid detection and have the maximum benefit.

The systematic analysis of cryptojacking in the wild, as presented in Hong et al. [1], is much-needed and specifically covers the deployment and monetization of mining scripts and binaries and campaign dynamics in the ecosystem. Jayasinghe et al. [3] discuss cryptojacking from a security and privacy point of view and provide challenges for detection and mitigation. Such measurement and characterization efforts help to identify common indicators of compromise and explain how attackers change over time.

In related work on security detection, Danesh et al. [16] examined behavioral patterns for stealth and slow scanning worms in Transmission Control Protocol traffic, and this work shows how learning-based methods can be used in support of the detection of anomalies in network streams. Additional work by other researchers focuses on security classification and signature optimization in related areas of abnormal behavior, such as DoS detection and worm traffic signature optimization [17-19]. Although these studies are not directly concerned with cryptojacking, they provide information to consider in the design and evaluation of features for security monitoring.

2.2 Browser-based cryptojacking and evasion

Web-based cryptojacking was popularized by websites like Coinhive, which allowed website operators (and bad actors) to mine Monero in the user's browser. Eskandari et al. [5] offer an early study of browser-based cryptojacking and analyze behavioral characteristics like CPU consumption patterns and script properties that can be used for detection. Abdul Aziz et al. [6] discuss Coinhive's drive-by cryptojacking behavior and its effects, reaffirming the viability of in-browser mining attacks. More recent research takes evasion strategies into consideration and how cryptojacking scripts try to evade modern-day detection tools. Chmiel and Rajba [11] analyze practical methods that attackers employ to get around web cryptojacking defenses. Together, these studies suggest the importance of having good signals as well as static signatures and are an inspiration for cautious feature selection.

2.3 Host-based and system-level detection

A large number of works utilize the host-level metrics for the detection of cryptojacking. Gomes and Correia [13] analyze the detection of cryptojacking using the measurements of CPU usage and show that under certain circumstances, resource-based measurements could be used to distinguish miners from one another. Lachtar et al. [7] propose a cross-stack defence that combines multiple layers of observations, and that the combination of system and application indicators can provide a more robust defence detection. In the context of Linux cloud environments, Park et al. [8] propose the detection and response of cryptojacking under the consideration of the host-based detection scheme with a lightweight scheme. Collectively, these approaches identify the potential of using system-level telemetry but also emphasize the importance of making a distinction between mining and normal high-load workloads.

2.4 Deep learning approaches

Machine learning-based detectors are often trained with

feature vectors, which are based on system metrics, network flows, or low-level execution traces. Caprolu et al. [14] present that cryptomining "makes noise" and that cryptojacking can be detected by ML based on observable patterns. This underscores the critical importance of informative features. Aponte-Novoa et al. [20] reconsider classifier-based cryptojacking detection on websites and conclude that performance is highly dependent on the feature set and threat model. Ying et al. [15] propose CJSpector, which uses hardware traces and deep learning, suggesting that microarchitectural signals are useful for detection. In containerized environments, Kim et al. [21] reveal cryptojacking container identification with the help of eBPF-based runtime monitoring based on ML in containerized environments to signify the shift toward cloud-native monitoring.

2.5 Evasion and real-time detection challenges

Attackers have started to use evasion to weaken detectors even more. Cryptojacking codes can be set to reduce the CPU load, randomize execution, or inject malicious code dynamically into the browser environment, which makes it difficult to detect signatures and cannot be analyzed using a static analysis [10, 11]. On endpoints, miners can be fileless or memory resident, which can mean that they are harder for traditional antivirus scanning to catch, and this also relies more on behavioral monitoring [22, 23]. Real-time constraints are also equally important; detectors have to be run with minimal overheads, especially in the server and cloud environments, where performance is of paramount importance. Pott et al. [24] proposed a detection method that involves hardware performance counters in Windows machines to detect cryptojacking attacks, in which a cryptocurrency (cryptojacking) is secretly tapped into a device's processing power. By combining both CPU and GPU metric performance, the method makes the detection more reliable but with little performance impact, and facilitate it could be used for continuous monitoring on live systems.

2.6 Online and adaptive methods

Beyond the issue of what to detect offline, a number of studies tackle issues of deployment-oriented and adaptive considerations. Orzechowski et al. [25] demonstrate that cryptojacking monitoring based on eBPF can be used to support a low-overhead implementation of cryptojacking in modern Linux environments to support near-real-time feature collection. More generally, abnormal populations change over time (concept drift), which can lead to the degradation of fixed detectors; Chakravorty and Elsayed [26] propose the need for drift-aware evaluation and model updating because of the evolution of abnormal families. These results suggest that operational cryptojacking detectors may require periodic retraining, drift-free, in order to maintain functionality and to be resilient to adaptive attackers [20].

2.7 Gap in the literature

Despite the substantial research on cryptojacking, there are a number of gaps. First, many works test an individual detector or compare different models in different experiments, which makes it difficult to draw any conclusion about which classifier family is most suitable in a whole architecture. Second, reproducibility is often limited: important aspects of implementation (such as feature preprocessing, missing value handling, training-testing splits, and hyperparameter setup) are not always reported. Third, interpretability is often neglected. While there are many papers with high accuracy, few give insight into which system-level signals are used in decision making, which is of paramount importance for practical deployment, where the cost of monitoring and telemetry limitations will be paramount. At last, the operational environment is often simplified. In real life, at the time of deployment, the commonly encountered problems include workload variance, encrypted channel, proxying, and adaptation by the attacker (concept drift). The present work tries to remedy these problems by comparing the RF, SVM, and MLP algorithms in a controlled pipeline and using a public dataset and inclusive interpretation in terms of feature importance analysis.

3. DATA AND METHODOLOGY

This research is based on a supervised learning workflow using the following steps: (i) dataset acquisition, (ii) data preprocessing and normalization, (iii) model training, and (iv) performance evaluation, as shown in Figures 1 and 2. To make a fair comparison between the models, the same data set, features, and train-test data split are used when training the RF, SVM, and MLP.

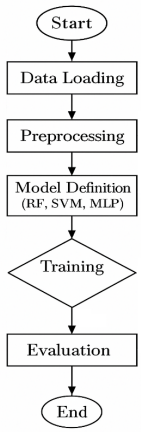

Figure 1. Machine learning pipeline flowchart

3.1 Data description

The dataset used for this study was created by combining two subsets that were labelled as normal and abnormal activity. The merged data set has 95,312 time-stamped observations with 84 columns (including label fields). Normal samples have a sample size of 80,851, while abnormal samples have a sample size of 14,461 (label=0 and label=1, respectively). For chronological evaluation, the timestamp field was only used to maintain the temporal order, but it was not considered a predictive feature. After data cleaning (metadata fields removal, numeric features selection), 58 features were selected for the purpose of model training. The raw columns consist of CPU and process activity-related metrics, system descriptor fields, and a timestamp. System descriptor fields were considered metadata and were not used in the training feature set. Class imbalance in favour of the normal activity: label=0 (80,851 samples), label=1 (14,461 samples) (N=95,312).

Missing values were quantified in all columns. A total of 30 missing values were found, and they only affected 2 samples (0.0021% of the dataset). Given this negligible missingness, missing entries were imputed with zero in order to keep all the samples. Table 1 summarizes the extent of missing values in the dataset, showing that only 30 values were missing and that just two samples (0.0021%) were affected.

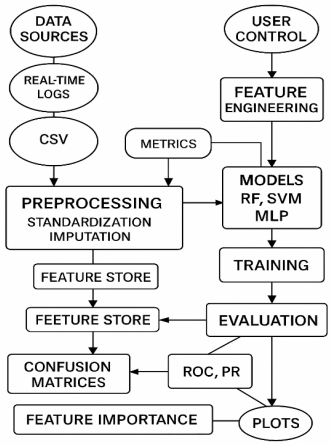

Figure 2. Preprocessing of the data

Table 1. Missing values summary

Metric	Value
Total missing values	30
Samples with ≥1 missing value	2
Percentage of samples affected	0.0021%

Table 2. Hyperparameters

Model	Parameter	Value
RF	n_estimators	100
RF	max_depth	None
RF	max_features	sqrt
RF	criterion	gini
RF	min_samples_split	2
SVM	kernel	rbf
SVM	C	1.0
SVM	gamma	scale
MLP	hidden_layer_sizes	(100, 50)
MLP	activation	relu
MLP	solver	adam
MLP	learning_rate_init	0.001
MLP	batch_size	auto
MLP	max_iter	500
All	random_state	42

Note: RF = Random Forest; SVM = Support Vector Machine; MLP = Multilayer Perceptron.

3.2 Preprocessing, evaluation protocol, and hyperparameter setting

Feature scaling was applied because models such as SVM and MLP are sensitive to feature magnitude. Standardization was performed using a z-score transformation so that each feature has a mean of zero and a variance of one. To prevent data leakage, the scaler was fitted on the training subset only and then applied to both the training and test subsets.

The timestamp field was used to impose chronological ordering for time-aware evaluation; it was not used as a predictive feature. The dataset was divided into training (70%) and testing (30%), and three evaluation protocols were applied. First, a stratified random split (random_state = 42) was used to maintain class proportions. Second, a chronological split was used by sorting samples by timestamp, training on the earliest 70%, and testing on the most recent 30%. Third, a 5-fold stratified cross-validation was performed to assess stability under resampling. The same split was applied to all classifiers so that a direct and fair comparison could be made. Performance was measured using confusion matrices, accuracy, precision, recall, F1-score, Area Under the Receiver Operating Characteristic Curve (ROC-AUC), and PR-AUC.

Hyperparameters were selected using commonly adopted defaults with limited tuning to reduce the risk of overfitting. RF was configured with a sufficient number of estimators to stabilize predictions. SVM used an RBF kernel to support non-linear separation. MLP was configured with two hidden layers, ReLU as the activation function, and Adam optimization, and convergence was monitored. Table 2 summarizes the hyperparameter settings used in this study.

3.3 Support Vector Machine

The Support Vector Machines (SVMs) categorise the data by training a maximum-margin decision boundary over a (possibly high-dimensional) feature space, which results in the distinction between normal and abnormal samples. An RBF kernel enables the SVM to learn non-linear relationships implicitly by mapping input features into a higher-dimensional space without necessarily computing it. An RBF kernel enables the SVM to learn non-linear relationships implicitly by mapping input features into a higher-dimensional space without explicitly computing the transformation. As shown in Figure 3, this Non-linear separation is also significant in the detection of cryptojacking since normal workloads and miners can overlap single measurements but become disaggregated when many measurements are all looked at together. Support vectors-representative samples which lie along the margin define the boundary of the decision. The SVM uses slack variables and a penalty term for misclassifications and thus gives tolerance to overlapping classes. However, the SVM is also vulnerable to the choice of the kernels, the regularization, and the scaling of features, which requires thorough preprocessing and cross-validation [14].

The SVM builds a maximum-margin hyperplane to separate normal from abnormal samples. Given a training set $\{(x_i, y_i)\}_{i-1}^{N}$ with $y_i \in \{-1, +1\}$, the primal problem is:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \xi_i \tag{1}$$

subject to:

$$y_i(w^\top \phi(x_i)+b) \geq 1 - \xi_i\,,\ \xi_i \geq 0\ \forall i \tag{2}$$

Here, $\phi(\cdot)$ is a feature-mapping (implicit via a kernel K), C controls the margin-slack trade-off, and ξ_i are hinge-loss slack variables. The corresponding dual formulation is:

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j K(x_i, x_j) \text{ s.t. } 0 \leq \alpha_i \leq C, \sum_i \alpha_i y_i = 0 \tag{3}$$

Once solved, the weight vector in feature space is:

$$w = \sum_{i-1}^{N} \alpha_i y_i \phi(x_i) \quad (4) \qquad f(x) = \text{sign}(\sum_{i-1}^{N} \alpha_i y_i K(x_i, x) + b) \quad (5)$$

and classification of a new sample x uses the decision function:

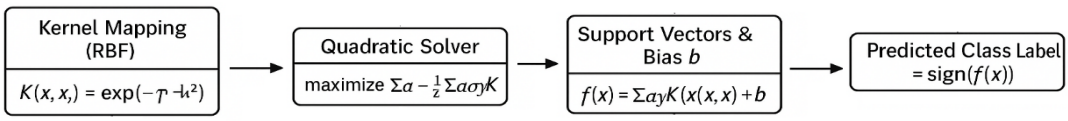

Figure 3. Support Vector Machine (SVM) (RBF Kernel) architecture

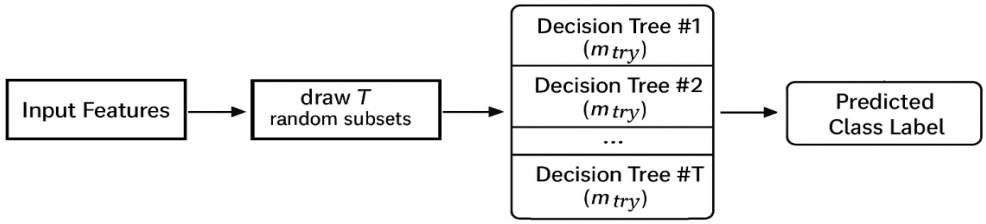

Figure 4. Random Forest (RF) ensemble architecture

3.4 Random Forest

Random Forest (RF) is an ensemble technique that constructs multiple decision trees, each trained on a bootstrap sample and a random subset of features, to reduce variance and increase robustness to noise and correlated features. Each tree partitions the feature space to decrease Gini impurity at each node, and the ensemble prediction is determined by majority vote across all trees. RF provides interpretable feature-importance values that indicate the relative contribution of each feature to classification, which can guide telemetry prioritization in operational deployments. The training process is parallelizable, and inference is fast, making RF suitable for real-time monitoring, as shown in Figure 4. Its ensemble structure also captures feature interactions effectively [27, 28].

An RF constructs an ensemble of T decision trees, each grown on a bootstrap sample and a random subset of features. Let $h_t(x) \in \{0,1\}$ denote the class vote of tree t. The ensemble prediction is:

$$H(x) = \arg\max_{c\in\{0,1\}} \sum_{t=1}^{T} \mathbf{1}\{h_t(x) = c\} \quad (6)$$

Each tree split is chosen to minimise the Gini impurity at a node:

$$I_G(p) = 1 - \sum_{k=0}^{1} p_k^2, p_k = \Pr\{y = k\} \quad (7)$$

The impurity decreases from a split into child nodes L(left) and R(right) is:

$$\Delta I = I_G\left(p^{(P)}\right) - \frac{N_L}{N_P} I_G\left(p^{(L)}\right) - \frac{N_R}{N_P} I_G\left(p^{(R)}\right) \quad (8)$$

where, N_P, N_L, N_R are parent and child node sample counts. The class probability estimate is:

$$\hat{p}(\, y = c \mid x\,) = \frac{1}{T}\sum_{t=1}^{T} 1\{h_t(x) = c\} \quad (9)$$

and the feature importance for feature j aggregates impurity decreases over all splits on j:

$$\text{FI}_j = \sum_{t=1}^{T} \sum_{s\in\mathcal{S}_t^j} \Delta I_{t,s} \quad (10)$$

where, $\mathcal{S}_t^j$ is the set of splits on feature j in tree t. Finally, the out-of-bag error can be estimated by:

$$\text{OOB}_{\text{Err}} = \frac{1}{N}\sum_{i=1}^{N} 1\{y_i \neq H_{\text{oob}}\,(x_i)\} \quad (11)$$

where, $H_{\text{oob}}\,(x_i)$ is the majority vote of trees not trained on x_i.

3.5 Multilayer Perceptron

Multilayer Perceptron (MLP) is a feedforward neural network with input, hidden, and output layers that can learn complex nonlinear relationships between process features and class labels. The model was configured with two hidden layers (100 and 50 units) to capture interactions between system-level features. Inputs were standardized to ensure stable gradient updates during training. Figure 5 illustrates the adopted MLP architecture. Training is slower than RF but remains feasible for offline analysis. The layer depth and number of neurons provide flexible capacity adaptation. Despite the black-box nature of neural networks, feature ablation studies can be employed to assess which features contribute most to the learned representations [15].

The MLP is a feedforward neural network with L layers. Denote $a^{(0)} = x$ the input and, for layer $\ell = 1, \ldots, L$,

$$z^{(\ell)} = W^{(\ell)} a^{(\ell-1)} + b^{(\ell)}\,,\; a^{(\ell)} = \sigma\left(z^{(\ell)}\right) \quad (12)$$

where, $W^{(\ell)}$ and $b^{(\ell)}$ are weights and biases, and $\sigma(\cdot)$ is the activation (e.g., ReLU or sigmoid):

$$\sigma(u) = max(0, u) \text{ or } \sigma(u) = \frac{1}{1+e^{-u}} \quad (13)$$

For binary classification, the output layer produces $\hat{y} = a^{(L)} \in (0,1)$, and the cross-entropy loss over N samples is:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} [y_i \ln \hat{y}_i + (1 - y_i) \ln(1 - \hat{y}_i)] \quad (14)$$

Training uses backpropagation. The output-layer error is:

$$\delta^{(L)} = \hat{y} - y \quad (15)$$

and the hidden-layer errors propagate as:

$$\delta^{(\ell)} = \left(W^{(\ell+1)\top}\delta^{(\ell+1)}\right) \circ \sigma'\left(z^{(\ell)}\right) \quad (16)$$

where, o denotes element-wise multiplication. Finally, parameters are updated via gradient descent with learning rate η:

$$W^{(\ell)} \leftarrow W^{(\ell)} - \eta\delta^{(\ell)}a^{(\ell-1)\top}, b^{(\ell)} \leftarrow b^{(\ell)} - \eta\delta^{(\ell)} \quad (17)$$

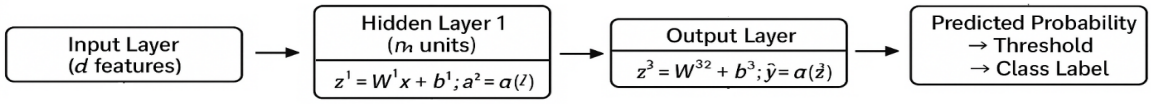

Figure 5. Multilayer Perceptron (MLP) architecture

4. RESULTS

This part shows the result of the performance of the RF, SVM, and MLP on the held-out test set and on 5-fold cross-validation. Accuracy, precision, recall, F1-score, ROC-AUC, and PR-AUC are used as the reporting of the results. Types of errors are characterised with the help of confusion matrices, and interpretability is provided with the help of RF feature importance.

4.1 Performance metrics

Table 3 displays important performance variables for the three models in the same test split. RF scored perfectly, and SVM as well as MLP scored at near-perfect accuracy with slight differences in recall on the random split. The values of ROC-AUC and PR-AUC show that they are highly separable using the random-split protocol.

Because the dataset exhibits a class imbalance (normal: 80,851; abnormal: 14,461), PR-AUC is reported alongside ROC-AUC. ROC-AUC can be optimistic under imbalance because the true-negative rate benefits from the large majority class, whereas PR-AUC focuses on the minority (positive) class and is therefore more informative when false negatives carry a higher cost.

4.2 Random vs. chronological split

Table 4 is used to evaluate the performance of the model when time is taken into consideration during the validation process by comparing the performance of models using random and chronological splitting. The findings support that good performance in terms of random splitting may not be associated with stable generalization in the long run, and hence the need to carry out temporal validation before real-life application. To facilitate transparency, the random-split test distribution included N0 = 24256 and N1 = 4338, and the chronological test distribution included N0 = 21412 and N1 = 7182.

A majority-class (dummy) baseline was added to put the metrics reported into context. The dummy classifier only predicts the majority class (label = 0), and its accuracy on the random split is 0.8483 and on the chronological split is 0.7489, with zero recall and F1 of the minority class, as shown in Figure 6. This proves that the trained models not only learn useful discriminative patterns but are not just trying to exploit the imbalance of classes.

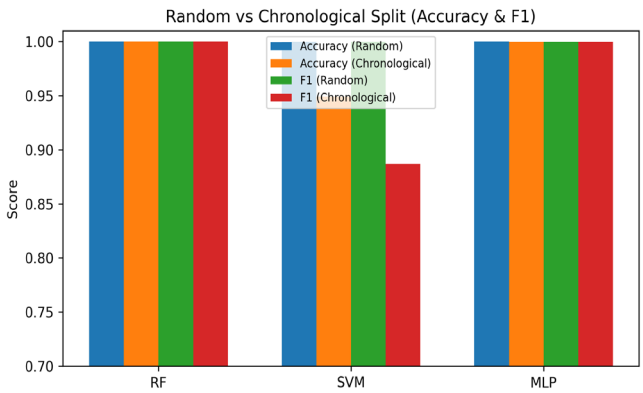

Figure 6. Random vs. chronological split (Accuracy and F1)

The number of false positives is zero in all the models and splits, which means that there were no cases of normal samples being falsely classified as cryptojacking. False negatives are solely errors (missed detections), and this is in line with the fact that certain cryptojacking samples can be similar to normal high-load activity at or near the decision boundary. SVM takes up most of the errors: three in the random splitting and 1,458 in the chronological splitting. As detailed in Table 5, all models produced zero false positives, whereas classification errors were entirely due to false negatives, with SVM showing the largest increase under the chronological split.

Table 3. Performance metrics on the held-out test set (random split)

Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC	PR-AUC
RF	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
SVM	0.9999	1.0000	0.9993	0.9997	1.0000	1.0000
MLP	1.0000	1.0000	0.9998	0.9999	0.9998	0.9998

Note: RF = Random Forest; SVM = Support Vector Machine; MLP = Multilayer Perceptron.

Table 4. Random split vs. chronological split

Model	Split	Accuracy	Precision	Recall	F1-Score	AUC	PR-AUC
RF	Random	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
RF	Chronological	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
SVM	Random	0.9999	1.0000	0.9993	0.9997	1.0000	1.0000
SVM	Chronological	0.9490	1.0000	0.7970	0.8870	1.0000	1.0000
MLP	Random	1.0000	1.0000	0.9998	0.9999	0.9998	0.9998
MLP	Chronological	0.9999	1.0000	0.9997	0.9999	0.9997	0.9998

Note: RF = Random Forest; SVM = Support Vector Machine; MLP = Multilayer Perceptron.

Table 5. Confusion matrix counts for all models

Model	Split	TN	FP	FN	TP
RF	Random	24,256	0	0	4,338
SVM	Random	24,256	0	3	4,335
MLP	Random	24,256	0	1	4,337
RF	Chronological	21,412	0	0	7,182
SVM	Chronological	21,412	0	1,458	5,724
MLP	Chronological	21,412	0	2	7,180

Note: TN = true negatives; FP = false positives; FN = false negatives; TP = true positives; RF = Random Forest; SVM = Support Vector Machine; MLP = Multilayer Perceptron.

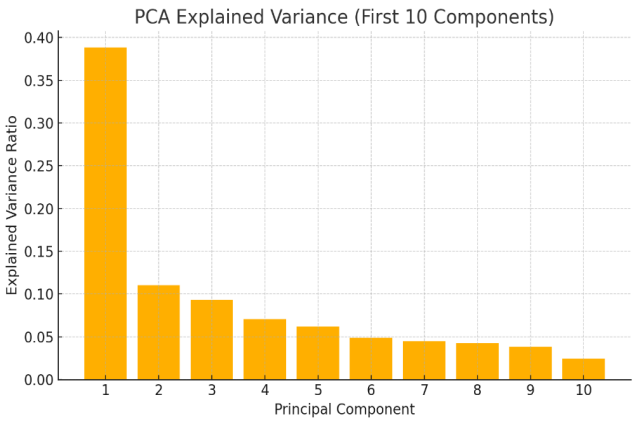

Figure 7. Principal Component Analysis (PCA) explained variance

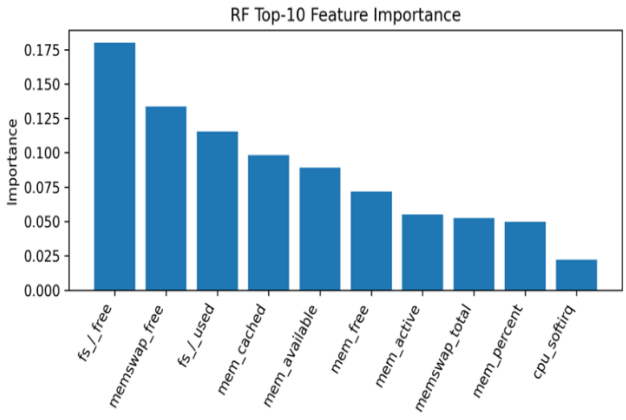

Figure 8. Random Forest (RF) Top-10 feature importance

4.3 Principal Component Analysis

In order to study structure and redundancy within the feature space, Principal Component Analysis (PCA) of the standardized features was conducted. The explained-variance profile indicates that there is a high level of correlation between the original features, as shown in Figure 7. Considering that the dataset has 58 features, PCA is not needed to classify based on their performance, but it can give some understanding of the relationships among features.

4.4 Analysis of feature importance

The interpretability was supported by RF feature-importance analysis that compared the features based on the importance of their contribution to the decisions of the classifier.

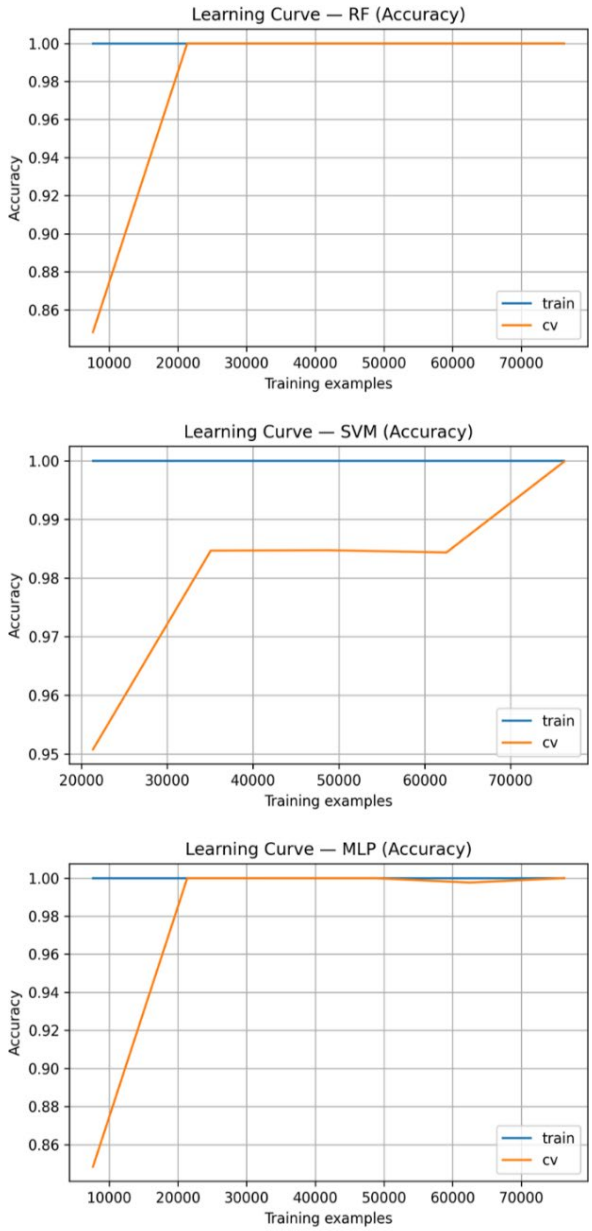

Figure 9. Learning curves

The top 10 features mostly consist of memory and filesystem usage metrics (nine out of ten), as shown in Figure 8 and Table 6. Only a single feature related to CPUs (cpu_softirq) found in the top 10. This implies that, in this data set, cryptojacking activity will be best defined in terms of memory allocation and disc space usage, and not by the direct CPU metrics. This observation is operationally viable: the monitoring systems can give precedence to the memory and filesystem telemetry to detect with efficiency.

4.5 Cross-validation results

Five-fold cross-validation was conducted on the standardized data to assess the stability and generalizability in a resampling of data. Table 7 presents the average and fold variation. SVM shows the highest variability, especially when time is binned using GroupKFold (std = 0.0009), indicating its sensitivity to temporal distribution changes that were present in the chronological split.

4.6 Rolling-origin time validation

A rolling-origin (expanding-window) evaluation was also conducted to further evaluate the temporal robustness. Training takes place on all the data until a cutoff point is reached, and testing takes place on the next block, and the window is increased successively. Table 8 reports results for two representative folds: an early fold (Fold 2, training on the first ~33% and testing on the next ~17%) and a late fold (Fold 5, training on the first ~83% and testing on the final ~17%).

The findings of the chronological split are supported by the rolling-origin findings. The worst performance of SVM is achieved at Fold 5 (accuracy 0.9065, recall 0.7932) when the test window is the most recent part of the data and the ratio of classes changes (test N0 = 8703, N1 = 7182). RF and MLP are almost flawless on both folds. This confirms that the chronological split SVM recall deficit is not due to a single partition but a time-forward evaluation pattern, and that there is a shift of the distribution which biases the SVM decision line disproportionately. As shown in Figure 9, the learning curves for RF, SVM (RBF), and MLP (training vs validation accuracy as a function of training set size). The results of cross-validation indicate that RF has zero variance across folds, but SVM has the highest variability.

Table 6. Random forest (RF) top-10 feature importance (Gini-based, mean decrease in impurity)

Rank	Feature	Importance
1	fs_/_free	0.1801
2	memswap_free	0.1336
3	fs_/_used	0.1153
4	mem_cached	0.0982
5	mem_available	0.0892
6	mem_free	0.0717
7	mem_active	0.0550
8	memswap_total	0.0525
9	mem_percent	0.0498
10	cpu_softirq	0.0221

Table 7. Cross-validation accuracy summary (5-fold)

Model	Mean Accuracy	Std	Notes
RF	1.0000	0.0000	5-fold stratified CV; time-binned GroupKFold mean=1.0000, std=0.0000
SVM	0.9999	0.0001	5-fold stratified CV; time-binned GroupKFold mean=0.9993, std=0.0009
MLP	1.0000	0.0000	5-fold stratified CV; time-binned GroupKFold mean=0.9999, std=0.0002

Note: RF = Random Forest; SVM = Support Vector Machine; MLP = Multilayer Perceptron.

Table 8. Rolling-origin time validation results

Model	Fold	Accuracy	Precision	Recall	F1-Score	FP	FN
RF	RollingFold2	1.0000	1.0000	1.0000	1.0000	0	0
SVM	RollingFold2	0.9999	1.0000	0.9994	0.9997	0	1
MLP	RollingFold2	1.0000	1.0000	1.0000	1.0000	0	0
RF	RollingFold5	1.0000	1.0000	1.0000	1.0000	0	0
SVM	RollingFold5	0.9065	1.0000	0.7932	0.8847	0	1,485
MLP	RollingFold5	0.9998	1.0000	0.9996	0.9998	0	3

Note: RF = Random Forest; SVM = Support Vector Machine; MLP = Multilayer Perceptron.

5. DISCUSSION

This part explains the implications of experimental findings, not restricted to raw scores of accuracy. We interpret the results in comparison with SVM using the selected set of features, relate the results to the existing literature on cryptojacking and abnormal detection, and discuss practical implications for implementation, including the overhead monitoring, the choice of the threshold, and the cost of false alarms in operation. At the end, we conclude by summarizing the major limitations of the dataset and the experiment environment and laying out the way to enhance generalizability in subsequent studies.

5.1 Interpretation of results

For SVM, the distinction between the random and chronological results hints at sensitivity to temporal order and possible distribution shift over time in the dataset. The rolling-origin evaluation (Section 4.6) confirms that this is not an artifact of a single partition: SVM recall drops to 0.7932 at Fold 5, consistent with the chronological-split finding. Meanwhile, the majority-class dummy baseline (accuracy 0.8483 random, 0.7489 chronological) confirms that all trained models learn meaningful patterns well above trivial class-frequency prediction.

Although random-split evaluation can indicate high separability within a given dataset, it may not reflect how well

a model will perform when the operational environment changes over time. Therefore, high accuracy under random splitting should be interpreted with caution, and claims regarding real-time or production deployment are better supported by time-aware evaluation and/or external validation.

One would want to select a model for deployment that would balance both detection performance and computational cost. The efficiency benchmarks highlight significant differences between models in inference latency and memory use, which may well have impacts on feasibility in a resource-constrained system. Accordingly, the model selection should consider the operational goal, which can be latency, memory footprint, or interpretability.

The experimental results show that RF and MLP are nearly perfect on the held-out test set, while SVM has slightly less accuracy. This is due to the difference in classifiers. RF enjoys ensemble averaging between trees and is also resistant to noise and interaction between features. MLP with standardized data can learn non-linear decision boundaries and will be able to accommodate the geometry of the feature space. Non-linear separation can also be modelled with SVM using an RBF kernel, although it is also sensitive to parameter choice, thereby influencing the recall and stability. The very small number of SVM and MLP error values must represent samples that are close to the class boundaries or are unusual system behavior. Since the cryptojacking workloads may be similar to legitimate high-load processes, edge cases are likely to occur in structured datasets. The observed high accuracy of RF and MLP is almost perfect and thus should be interpreted with a lot of care. This rate of performance indicates the high separability of classes in the Cryptojacking Attack Time Series Dataset, as mining activity generates unique resource-consumption trends that can be distinguished with statistically significant difference from normal behavior. The values were calculated using reproducible experiments that were run on Google Colab using a constant random seed (random_state=42), and the anti-leakage pipeline ensures that no information about the test is used in training. Although this dataset has a high degree of separability, this doesn't just mean that it performs the same way in environments characterised by higher amounts of operational heterogeneity, where the workload diversity, attacker evasion, and concept drift may impair discriminative clarity. The loss of SVM recall that is shown for both chronological and rolling (origin or origin rolling) testing (Sections 4.2 and 4.6) suggests some change of distribution within the data set, which can negatively affect the model performance and highlight the impact of the temporal dynamics on model deployment assumptions.

5.2 Comparison with prior work

The performance attained is in line with what has been reported in the literature on the detection of cryptojacking using machine-learning-based classifiers, where classifiers that are trained on labeled telemetry from the system level are usually highly accurate. Gomes and Correia explored the problem of cryptojacking detection using CPU usage data and proved that resource-based indicators are capable of differentiating miners and normal processes given controlled conditions [13]. According to Caprolu et al. [14], observable behavioral noise due to mining activity can be used by an ML classifier. The current research not only validates these results but expands upon them with a controlled three-classifier comparison under a single pipeline, including time-sensitive analysis that demonstrates how performance can be negatively impacted by distribution shift, which has not always been a focus in the previous literature.

Compared to browser-based detection studies, which typically focus on JavaScript features and in-page behavior [5, 6, 11], this work operates on host-level system metrics and therefore complements web-centric approaches. Lachtar et al. proposed a cross-stack defense integrating multiple observation layers [7], and Kim et al. [21] demonstrated eBPF-based container-level detection; both highlight the value of system-level signals, which aligns with the feature-importance findings reported here (Table 6), where memory and filesystem metrics dominate. The near-perfect RF and MLP performance observed across all evaluation protocols suggests that, when the feature set adequately captures mining-induced resource changes, supervised models can provide highly effective detection. However, the strong class separability observed in this dataset may not generalize to more adversarial or heterogeneous environments, which motivates the external validation recommended in Section 5.4.

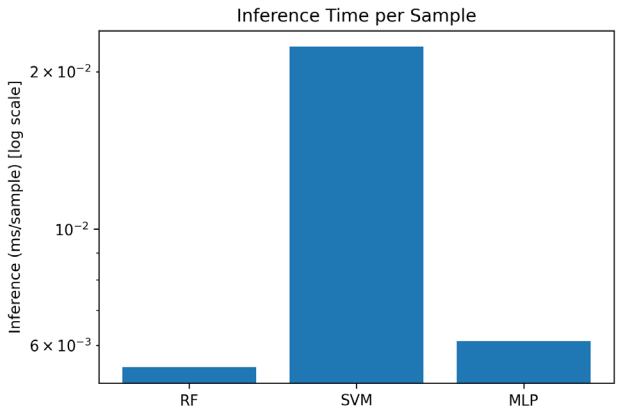

Figure 10. Inference time per sample (log scale)

Table 9. Training time, inference, and memory

Model	Training Time (s)	Inference (ms/sample)	Memory (MB)
RF	4.7941	0.0055	92.38
SVM	13.9751	0.0223	92.36
MLP	22.0425	0.0061	92.36

Note: RF = Random Forest; SVM = Support Vector Machine; MLP = Multilayer Perceptron.

5.3 Recommendations for implementation

In practice, the choice of classifier must also be based on factors such as monitoring overhead, detectability, interpretability, and operational needs in addition to detection accuracy. RF offers good performance, interpretable feature-importance scores, which can be used to guide security teams to prioritize telemetry collection, and the shortest inference time (0.0055 ms/sample). SVM is sometimes practical, though it might demand parameter fine-tuning, and it is susceptible to variations in feature scales and time distributions. MLP has a high accuracy and inference latency equal to RF (0.0061 ms /sample), as shown in Figure 10, but further tuning is required to ensure stability when there is a change in the workload. To ensure the reliability of detection over time, the drift monitoring and the periodical revalidation of the recent data are suggested. In order to reduce overhead, the detectors can

narrow on a small set of features based on the memory and filesystem usage statistics that ranked the top-10 RF importance rankings (Table 6), with the exception of the CPU-related feature of cpu_softirq appearing at the top ten. Selection of the threshold also needs to be allowed in order to trade off false positives and false negatives according to the priorities of operations. Besides predictive performance, computational efficiency was also determined in order to facilitate deployment-oriented comparison. Table 9 provides a summary of training time, inference latency per sample, and memory footprint of each model.

5.4 Limitations

This research has several limitations. First, only one publicly available dataset with binary, normalised/abnormal labels is used to evaluate it, and this might not be reflective of the real-life workloads and attacker plans. Second, independently collected datasets should also be validated externally to confirm generalizability.

Future work will focus on (1) validation of the approach on external datasets and additional system environments, (2) drift-aware or online learning strategies, (3) learning curve analysis to understand data sufficiency, and (4) refinement of feature engineering and calibration to improve robustness under chronological evaluation.

6. CONCLUSIONS

This paper compared three supervised machine-learning classifiers, RF, SVM, and MLP, for cryptojacking detection using a public time-stamped dataset. Models were evaluated under random and chronological splits, rolling-origin time validation, and 5-fold cross-validation, providing complementary evidence on performance and stability. Predictive metrics, PR-AUC for imbalanced evaluation, and efficiency benchmarks were reported to support deployment-oriented model selection. A majority-class baseline confirmed that model performance is well above trivial class-frequency prediction. These results support time-aware evaluation and reproducible reporting when discussing deployability in operational cryptojacking detection scenarios.

This assessment should be expanded in various ways in the future. First, it has to be tested on bigger and more heterogeneous datasets, such as GPU-intensive cases and those that are cross-platform, to ensure generalizability to other datasets. In addition to this, the adaptation to concept drift and ingesting streaming data would make for a stronger resilience against evolving attack patterns.

REFERENCES

[1] Hong, G., Yang, Z., Yang, S., Zhang, L., Nan, Y., Zhang, Z., Yang, M., Zhang, Y., Qian, Z., Duan, H. (2018). How you get shot in the back: A systematical study about cryptojacking in the real world. In CCS '18: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, Canada, pp. 1701-1713. https://doi.org/10.1145/3243734.3243840

[2] Carlin, D., Burgess, J., O'Kane, P., Sezer, S. (2020). You could be Mine(d): The rise of cryptojacking. IEEE Security & Privacy, 18(2): 16-22. https://doi.org/10.1109/MSEC.2019.2920585

[3] Jayasinghe, K., Poravi, H., Jayasekara, P. (2020). A survey of attack instances of cryptojacking targeting cloud infrastructure. In APIT '20: Proceedings of the 2020 2nd Asia Pacific Information Technology Conference, Bali Island, Indonesia, pp. 100-107. https://doi.org/10.1145/3379310.3379323

[4] Jayasinghe, K. (2025). Cryptojacking Attack Time Series Dataset. Kaggle. https://www.kaggle.com/datasets/keshanijayasinghe/cryptojacking-attack-timeseries-dataset, accessed on Mar. 05, 2025.

[5] Eskandari, S., Leoutsarakos, A., Mursch, T., Clark, J. (2018). A first look at browser-based cryptojacking. In 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), London, UK, pp. 58-66. https://doi.org/10.1109/EuroSPW.2018.00014

[6] Abdul Aziz, A., Ngah, S., Ti Dun, Y., Fui Bee, T. (2020). Coinhive's Monero drive-by crypto-jacking. IOP Conference Series: Materials Science and Engineering, 769(1): 012065. https://doi.org/10.1088/1757-899X/769/1/012065

[7] Lachtar, N., Elkhail, A.A., Bacha, A., Malik, H. (2020). A cross-stack approach towards defending against cryptojacking. IEEE Computer Architecture Letters, 19(2): 126-129. https://doi.org/10.1109/LCA.2020.3017457

[8] Park, G., Kim, J., Choi, J., Kim, J. (2025). CryptoGuard: Lightweight hybrid detection and response to host-based cryptojackers in Linux cloud environments. In ASIA CCS '25: Proceedings of the 20th ACM Asia Conference on Computer and Communications Security, Hanoi, Vietnam, pp. 1617-1631. https://doi.org/10.1145/3708821.3736186

[9] Recabarren, R., Carbunar, B. (2017). Hardening stratum, the bitcoin pool mining protocol. Proceedings on Privacy Enhancing Technologies (PoPETs), 2017(3): 57-74. https://doi.org/10.1515/popets-2017-0028

[10] Konoth, R.K., Vineti, E., Moonsamy, V., Lindorfer, M., Kruegel, C., Bos, H., Vigna, G. (2018). MineSweeper: An in-depth look into drive-by cryptocurrency mining and its defense. In CCS '18: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, Canada, pp. 1714-1730. https://doi.org/10.1145/3243734.3243858

[11] Chmiel, K., Rajba, P. (2024). How to evade modern web cryptojacking detection tools? A review of practical findings. In ARES '24: Proceedings of the 19th International Conference on Availability, Reliability and Security, Vienna, Austria, pp. 1-10. https://doi.org/10.1145/3664476.3670936

[12] Feng, Y., Sisodia, D., Li, J. (2020). POSTER: Content-agnostic identification of cryptojacking in network traffic. In ASIA CCS '20: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, Taipei, Taiwan, China, pp. 907-909. https://doi.org/10.1145/3320269.3405440

[13] Gomes, F., Correia, M. (2020). Cryptojacking detection with CPU usage metrics. In 2020 IEEE 19th International Symposium on Network Computing and Applications (NCA), Cambridge, MA, USA, pp. 1-10. https://doi.org/10.1109/NCA51143.2020.9306696

[14] Caprolu, M., Raponi, S., Oligeri, G., Di Pietro, R. (2021). Cryptomining makes noise: Detecting cryptojacking via

Machine Learning. Computer Communications, 171: 126-139. https://doi.org/10.1016/j.comcom.2021.02.016
[15] Ying, Q., Yu, Y., Tian, D., Jia, X., Ma, R., Hu, C. (2022). CJSpector: A novel cryptojacking detection method using hardware trace and deep learning. Journal of Grid Computing, 20: 31. https://doi.org/10.1007/s10723-022-09621-2
[16] Danesh, H., Karimi, M.B., Arasteh, B. (2024). Cmshark: A netflow and machine-learning based crypto-jacking intrusion-detection method. Intelligent Decision Technologies. https://doi.org/10.3233/IDT-240319
[17] Rasheed, M.M., Faieq, A.K., Hashim, A.A. (2021). Development of a new system to detect denial of service attack using machine learning classification. Indonesian Journal of Electrical Engineering and Computer Science, 23(2): 1068-1072. https://doi.org/10.11591/ijeecs.v23.i2.pp1068-1072
[18] Wang, Z., Peng, D., Tong, W., Wang, Z., Wang, W., Tang, Z. (2026). GRABIN: Detecting in-browser cryptojacking via graph representation of runtime behaviors. SSRN. https://doi.org/10.2139/ssrn.6257139
[19] Alorainy, W.S. (2025). CryptojackGuard: A multi-modal framework for proactive cryptojacking detection and mitigation. IEEE Access, 13: 196460-196478. https://doi.org/10.1109/ACCESS.2025.3631529
[20] Aponte-Novoa, F.A., Povedano Álvarez, D., Villanueva-Polanco, R., Sandoval Orozco, A.L., García Villalba, L.J. (2022). On detecting cryptojacking on websites: Revisiting the use of classifiers. Sensors, 22(23): 9219. https://doi.org/10.3390/s22239219
[21] Kim, R., Ryu, J., Kim, S., Lee, S., Kim, S. (2025). Detecting cryptojacking containers using eBPF-based security runtime and machine learning. Electronics, 14(6): 1208. https://doi.org/10.3390/electronics14061208
[22] Rai, A., Im, E.G. (2025). MemCatcher: An in-depth analysis approach to detect in-memory malware. Applied Sciences, 15(21): 11800. https://doi.org/10.3390/app152111800
[23] Khushali, V. (2020). A review on fileless abnormal analysis techniques. International Journal of Engineering Research & Technology (IJERT), 9(5): 46-49. https://doi.org/10.17577/ijertv9is050068
[24] Pott, C., Gulmezoglu, B., Eisenbarth, T. (2023). Overcoming the pitfalls of HPC-based cryptojacking detection in presence of GPUs. In CODASPY '23: Proceedings of the Thirteenth ACM Conference on Data and Application Security and Privacy, Charlotte, NC, USA, pp. 177-188. https://doi.org/10.1145/3577923.3583655
[25] Orzechowski, N., Zuppelli, M., Caviglione, L. (2025). Cryptojacking detection using eBPF and machine learning techniques. In 2025 10th International Conference on Information and Network Technologies (ICINT), Melbourne, Australia, pp. 22-29. https://doi.org/10.1109/ICINT65528.2025.11030875
[26] Chakravorty, A., Elsayed, N. (2026). Detecting cryptojacking in cloud environments: A systematic review of AI-based defenses, deployment challenges, and research gaps. Research Square. https://doi.org/10.21203/rs.3.rs-8627518/v1
[27] Rasheed, M.M., Faieq, A.K., Hashim, A.A. (2020). Android botnet detection using machine learning. Ingénierie des Systèmes d'Information, 25(1): 127-130. https://doi.org/10.18280/isi.250117
[28] Ghaleb, M. (2025). Design and evaluation of BE-RF framework on multi-dataset: A network intrusion detection system using ensemble learning with Random Forest. International Journal of Safety and Security Engineering, 15(11): 2219-2227. https://doi.org/10.18280/ijsse.151103