Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

An optimized neural network with AdaHessian for cryptojacking attack prediction for Securing Crypto Exchange Operations of MEC applications

Uma Rani[1], Sunil Kumar[2], Neeraj Dahiya[3], Kamna Solanki[4], Shanu Rakesh Kuttan[5], Sajid Shah[6], Momina Shaheen[7] and Faizan Ahmad[8*]

Abstract

Bitcoin exchange security is crucial because of MEC's widespread use. Cryptojacking has compromised MEC app security and bitcoin exchange ecosystem functionality. This paper propose a cutting-edge neural network and AdaHessian optimization technique for cryptojacking prediction and defense. We provide a cutting-edge deep neural network (DNN) cryptojacking attack prediction approach employing pruning, post-training quantization, and AdaHessian optimization. To solve these problems, this paper apply pruning, post-training quantization, and AdaHessian optimization. A new framework for quick DNN training utilizing AdaHessian optimization can detect cryptojacking attempts with reduced computational cost. Pruning and post-training quantization improve the model for low-CPU on-edge devices. The proposed approach drastically decreases model parameters without affecting Cryptojacking attack prediction. The model has Recall 98.72%, Precision 98.91%, F1-Score 99.09%, MSE 0.0140, RMSE 0.0137, and MAE 0.0139. Our solution beats state-of-the-art approaches in precision, computational efficiency, and resource consumption, allowing more realistic, trustworthy, and cost-effective machine learning models. We address increasing cybersecurity issues holistically by completing the DNN optimization-security loop. Securing Crypto Exchange Operations delivers scalable and efficient Cryptojacking protection, improving machine learning, cybersecurity, and network management.

Keywords Mobile Edge Computing (MEC) Deep Neural network model, Post-training quantization, AdaHessian optimizer, Cryptojacking attack, Crypto Exchange Operations

*Correspondence:
Faizan Ahmad
fahmad@cardiffmet.ac.uk
[1] Department of CSE, World College of Technology & Management, Gurugram, Haryana, India
[2] Department of CSE, Guru Jambheshwar University of Science & Technology, Hisar, Haryana, India
[3] Department of CSE, SRM University Delhi-NCR, Sonipat, Haryana, India
[4] Department of Computer Science Engineering, UIET, Maharshi Dayanand University, Rohtak, Haryana, India
[5] Department of CSE, Chouksey Engineering College Bilaspur, Chhattisgarh, India
[6] Prince Sultan University, Riyadh, Saudi Arabia
[7] Department of Computing, University of Roehampton London, London, UK
[8] Cardiff School of Technologies, Cardiff Metropolitan University, Cardiff, UK

Introduction

Mobile Edge Computing (MEC) apps are vulnerable to cryptojacking attacks, which can compromise their security and performance. Deploying computing resources closer to end-users and devices at the network edge is what Mobile Edge Computing is all about. The platform's closeness makes it ideal for a variety of businesses, including bitcoin exchanges, by reducing latency and improving application efficiency. An online threat known as cryptojacking—or malicious crypto-mining—occurs when cybercriminals secretly employ a device's processing power to mine cryptocurrency. Because the computational resources in MEC are shared and dispersed, these attacks can have serious consequences. Devices in the MEC infrastructure are the targets of cryptojacking attacks, which aim to exploit their computing power. This can encompass both conventional computer resources and specialist gear like graphics processing units (GPUs) or tensor processing units (TPUs), which are frequently employed in cryptocurrency mining. Any illicit utilization of computer resources for cryptocurrency mining has the potential to drastically diminish the performance of MEC applications due to the fact that MEC depends on low-latency communication and fast processing at the network edge. Reductions in overall system efficiency and increases in reaction times are possible outcomes of increased resource use.

Many cryptojacking attempts aim to remain undetected by end users by operating invisibly in the background. Because of how covert these strikes are, they could be difficult to spot and counter quickly in a MEC setting. Since cryptojacking attacks in MEC might not display conventionally malevolent behavior, conventional security methods might not be enough to identify them. The dispersed nature of MEC also makes it more difficult to keep an eye on everything from one place. It takes a multipronged strategy to prevent cryptojacking in MEC. To achieve this goal, it is necessary to install intrusion detection systems that are specifically designed for MEC settings, establish strong access restrictions, update and patch software on a regular basis, and educate users about the dangers of using untrusted apps. The capacity to identify cryptojacking attacks can be improved by utilizing machine learning methods, as indicated in the preceding abstract. Algorithms like this can study resource consumption trends linked to cryptocurrency mining and sound the alarm when they see anything out of the ordinary. Because of the dispersed and resource-constrained nature of MEC settings, proactive security measures are required to mitigate the cryptojacking threat in MEC. The MEC ecosystem may be made more safe for cryptocurrency exchange activities by combining powerful detection algorithms with rigorous access restrictions and user education [1]. The victim's device may experience performance degradation, higher power bills, and even hardware failure as a result of the mining process. An example of a possible cryptojacking attack is as follows:

- Infection: The hacker exploits a hole in the target's defenses by inserting malicious code. This code, which is often written in JavaScript, is meant to operate invisibly.
- Distribution: The malicious code might be disseminated via a variety of vectors, including hacked websites, phishing emails, infected files, or malicious advertisements.
- Execution: Malicious code is run on a victim's device when they visit a hacked website or interact with the malicious content. It then begins mining cryptocurrency with the device's resources, whether Bitcoin, Monero, or Ethereum.
- Use of Materials: Due to the extensive computational activities being done by the mining script, the victim's device experiences a decrease in performance, increased fan activity, and higher energy consumption.
- Gains for the Aggressor: The bitcoin is mined and then sent to the wallet of the attacker. Since the infected machines are pooling their resources, the attacker can amass a large sum of bitcoin.

Several high-profile cases over the past few years illustrate the development of the cryptojacking danger [2]. Some early examples of cryptojacking attacks are as follows:

1. Coinhive: With the introduction of the JavaScript-based mining service Coinhive in 2017, website owners may use their users' CPU resources to mine the cryptocurrency Monero. While it was promoted as a non-intrusive way for websites to earn money, attackers soon began injecting Coinhive scripts onto hacked websites in an attempt to steal cryptocurrency.
2. Tesla Cloud Cryptojacking: In 2018, it was revealed that bitcoin miners have gained access to Tesla's (an electric vehicle company) cloud infrastructure. Intruders hacked into Tesla's Amazon Web Services (AWS) account and mined cryptocurrency using the company's computing resources.
3. Government website cryptojacking: In 2018, cryptojacking attacks hit a number of government websites throughout the globe, including those of the United Kingdom and the United States. In order to mine

cryptocurrencies using users' computers, attackers installed malicious malware into these sites.

4. Smominru Botnet: Since its discovery in 2017, the Smominru botnet has infected hundreds of thousands of machines throughout the world. It was designed to mine the Monero cryptocurrency and mostly affected Windows computers. The botnet propagated itself in a number of ways, one of which was through taking advantage of Windows security holes.
5. Kitty Malware and Drupalgeddon2: In 2018, cybercriminals used the Drupalgeddon2 vulnerability (CVE-2018–7600) to spread Kitty, bitcoin mining malware. This trojan exploited security holes in Drupal websites in order to mine cryptocurrency.
6. Android Malware Used to Mine Cryptocurrencies: Cryptojacking attacks have even reached mobile devices. Malicious applications and compromised websites have both been found to install bitcoin mining malware on Android devices.
7. Cryptojacking in Industrial Supply Networks: In certain incidents, hackers inserted cryptojacking software into supply chain procedures. In these incidents, hackers spread malware across a wide variety of devices by exploiting vulnerabilities in recently released software patches.
8. Watering Hole Attacks: Websites frequented by a specific demographic are the targets of "watering hole" attacks. The tactic has been used by attackers to insert cryptojacking scripts into websites frequented by targeted groups.
9. Cryptojacking Ransomware: There have been cases of cryptojacking being used in conjunction with ransomware, with the attackers threatening to keep mining on the victim's machine until a ransom is paid. Because of this, victims are under even more pressure to give in to the demands of their assailants.

These are but a few of the many historical cases of cryptojacking that have been documented. Attackers will certainly come up with new ways and strategies to take advantage of the increased interest in cryptocurrencies as the cryptocurrency landscape continues to change. Staying up-to-date on cybersecurity best practices and implementing suitable security measures can help people and businesses fend off these attacks and stop cryptocurrencies from being mined without permission [3]. Cryptojacking, or the illegal use of computational resources to mine cryptocurrency, has been combated with the use of artificial intelligence (AI). Algorithms based on artificial intelligence may "learn" typical system behavior and "spot" deviations, such as unexpected increases in CPU or GPU utilization. Cryptojacking is a common cause of these surges. Artificial intelligence can monitor and assess how processes and programs are functioning in real time. Resource-intensive calculations outside of typical user or system behavior are at the heart of cryptojacking. AI is capable of detecting these discrepancies. Artificial intelligence has been taught to spot signatures in the kinds of scripts or code used in cryptojacking attacks. These patterns can be found by AI models by examining active processes or network traffic. Systems driven by AI can keep a constant eye on server load and traffic. They are able to quickly detect resource use anomalies that may indicate cryptojacking [4].

Deploying AI models on the cloud allows for ubiquitous resource tracking. This has been used to spot instances that are acting strangely and using too much resources, both of which has been signs of cryptojacking. Features characteristic of cryptojacking has been extracted by AI models from network traffic, scripts, or processes. Predictive models using these traits can be used to identify active or attempted attacks. In order to counteract evolving cryptojacking methods and innovative attack patterns, AI systems may continually learn from fresh data [5]. Artificial intelligence systems can automatically take action against cryptojacking, such as isolating compromised machines, alerting system administrators, or killing off malicious code. Advanced threat detection skills are one way in which AI might supplement more conventional security measures. It can be used in tandem with other security measures to prevent cryptojacking, such as firewalls, antivirus programs, and intrusion detection systems. In order to lessen the likelihood of unknowing participation in cryptojacking attacks, AI-powered platforms can aid in teaching users about the warning indications of cryptojacking and safe online habits. The identification and countermeasures of cryptojacking can be greatly aided by artificial intelligence. A multi-layered security approach, including artificial intelligence, frequent software upgrades, network monitoring, and user education, is crucial for successful protection against cryptojacking, but remember that no solution is foolproof [6]. The critical contribution of this research is as follows:

- The purpose of this study is to ensures that cryptojacking attacks can be detected efficiently at the network's edge
- In this study, AdaHessian optimization enhances the model's training process by adapting learning rates and efficiently navigating the loss landscape.
- Post-training quantization reduces memory and computational demands by converting model weights and activations to lower bit-width representations.

- The model implements an importance scoring mechanism, incorporating insights from AdaHessian, to identify and prune less informative parameters.
- This process optimizes model size while preserving prediction accuracy.
- The suggested model outperforms the state-of-the-art in terms of precision, accuracy, and sensitivity.

The full study should be written as follows: "Review of Literature" section discusses previous research, "Dataset" section de-scribes the dataset in detail, "Method" section explains the proposed method, "Results and analysis" section describes the experi-mental results and analysis, and "Conclusion" section discusses the conclusion and future works.

Review of literature

Eskandari et al. [1] looked at the growing trend of browser-based cryptocurrency mining, namely Monero mining with Coinhive and related code-bases. In this paradigm, the user visits a website, downloads JavaScript code, which operates client-side in her browser and mines bitcoin (usually without her knowledge or agreement), and then pays the seigniorage to the hosting website. Intentionally, websites have used this to replace or supplement ad income; inadvertently, websites have served the code as a consequence of a breach (in which case the attacker has collected the seigniorage).

The detection of bitcoin miners using NetFlow/IPFIX network data is presented as a machine learning-based technique [5]. In contrast to DPI-based methods, our approach can detect miners with comparable accuracy at a fraction of the cost. Knowing whether or not bitcoin miners are sneaking onto their networks to use them without authorization is of utmost importance in this scenario. IP address lists from recognized mining pools, DNS traffic processing, and direct Deep Packet Inspection (DPI) across all traffic may all be used to identify them immediately. However, none of these techniques has been successful in identifying miners utilizing anonymous mining servers or has proven inexpensive enough for widespread deployment in real-world networks.

The static, dynamic, and economic elements of browser-based cryptojacking are comprehensively examined [4]. To 1) quantify their prevalence throughout the web, 2) highlight their platform preferences, and 3) investigate the complexity of their code, we undertake content-, currency-, and code-based classification of cryptojacking samples as part of our static analysis. To isolate cryptojacking code from non-malicious JavaScript, we use unsupervised learning, which improves accuracy to 96.4%. In our dynamic study, we look at how cryptojacking affects the utilization of vital system resources like the CPU and the battery. To further investigate the communication between the victim node and the dropzone cryptojacking server, we use browser fingerprinting. We also develop a theoretical framework to examine the practicality of cryptojacking as a complement to traditional forms of internet advertising. Based on our findings, the model is economically unrealistic due to a sizable negative profit and loss gap. Finally, we develop enhanced countermeasures for in-browser cryptojacking by utilizing insights from our analysis.

Yulianto et al. [2] included Taint analysis-based cryptojacking protection as a Chrome addon. In this study, the Man-In-The-Middle (MITM) attack was modeled and abused to test for security measures. In the event of a cryptojacking attack, users will be alerted via the suggested methodology. As a result, the user is able to inspect the features of the scripts that are actively processing in the site's background. This study demonstrates that taint analysis is a useful tool for protecting against cryptojacking. The taint analysis technique can identify 19 cryptojacking-infected websites out of a random sample of 100 websites.

The problem of cryptojacking, in which miners are discreetly placed inside browser code without the user's awareness, is investigated in detail in a new work [6]. As such, we examine the top 50,000 Alexa-ranked websites and discover a sizeable portion of them partaking in this predatory activity, frequently with highly disguised code. In addition, mining protection plugins like NoMiner don't catch such subtly buried occurrences. As a result, we suggest a machine learning approach that makes use of real-time, hardware-assisted profiling of browser code. We are able to accurately categorize mining programs (with a 99% success rate) based on their micro-architecture, and we can even detect when the mining code has been severely encrypted or obfuscated. We develop our own add-on for Chrome and demonstrate its superior performance compared to existing add-ons. The suggested architecture is compatible with all commercially available CPUs and imposes little burden on the user's computer.

Lachtar et al. [7] investigates a cross-platform, generic approach to identifying cryptojacking attempts. We present an end-to-end detection approach that makes use of subtle modifications to the microarchitecture to monitor instructions often employed by hash algorithms. Our approach adds almost no extra time to tests across a variety of SPEC 2006 workloads, as shown by the evaluation.

Tanana and Tanana [8] present a more robust detection tool for countering cryptojacking. They also provide a brief overview of the history of cryptojacking (also known as harmful mining) and a survey of the most significant efforts to far. Our earlier efforts in harmful mining

detection will be reviewed, as will our current detection engine, which is mostly based on CPU utilization algorithms. While prior work produced an 81% detection rate against a specified set of cryptojacking samples, we will integrate new measures for malicious mining identification, such as network consumption and calls to cryptographic libraries, to improve this to 93%. Finally, we'll talk about expanding the suggested detection method to GPU cryptojackers.

To far, no research has been conducted to determine whether or not particular technological aspects of a website might raise (reduce) the risk of being hacked for cryptojacking operations. To answer this question, Di Tizio & Nam Ngo [9],suggest a case–control research utilizing a dataset of cryptojacking websites gathered by a WebCrawler implementation of Minesweeper. Preliminary findings from our investigation suggest a link between a few different website features, but the data does not reach statistical significance. In order to have a deeper understanding of the implications of these connections, additional research is needed.

For both browser-based and executable-type cryptojacking examples [10], proposes a complicated detection approach based on CPU load by an application. Our method's corresponding prototype identification software was developed utilizing a decision tree algorithm. The software was successful 82% of the time when tested against a small subset of known cryptojacking samples in a controlled virtual machine environment. Finally, we'll talk about how the proposed method can be applied more broadly in the future.

To identify cryptojacking without needing any training data or prior knowledge of the attacks, Gomes et al. [11], provides a hybrid technique. Using unsupervised machine learning methods, our Cryp-tojacking Intrusion Detection Approach, Cryingjackpot, collects and combines information based on flow and performance counters to group hosts that exhibit similar behaviors. Using a synthetic and a hybrid dataset, we conduct experimental evaluations of Cryingjackpot, with F1-scores reaching 97%.

A protection against cryptojacking that operates on both the hardware and operating system levels has been presented [12]. Our approach is app-agnostic, unlike previous studies that only looked for cryptojacking in browsers. We demonstrate that common tracking instructions used in cryptographic hash functions has been exploited as robust fingerprints of cryptojacking attacks. We show that our system can withstand the attacks of cryptojacking malware, which frequently use multi-threaded and throttling evasion strategies. Through rigorous testing on a wide variety of workloads, including real-world consumer applications, we are able to accurately describe the stability of our system. Finally, testing using a suite of benchmark programs reveals that our proof-of-concept solution has negligible effect on overall performance.

It is proposed by Caprolu et al. [13] that network traffic alone, even when encrypted and intermingled with non-malicious traces, has been used to detect and identify the actions of crypto-clients. First, we conduct a comprehensive study of the actual network traces produced by Bitcoin, Monero, and Bytecoin, taking into account both the natural traffic and the traffic modified by a virtual private network. To recognize cryptocurrency-related behaviors including pool mining, solo mining, and active full nodes, we then present Crypto-Aegis, a Machine Learning (ML) based framework constructed using our research findings. Our approach has several desirable qualities, including device and infrastructure independence, and an impressive F1-score of 0.96 and an AUC for the ROC of 0.99. We feel that our methodology, backed by its great findings, pave the path for additional study in this field, given the scope and originality of the danger addressed.

Lightweight cryptojacking traffic detection based on network behavior characteristics for an ISP is designed by Hu et al. [14]. This approach does not require access to the payload of network traffic. Using a specially designed lab, we gather cryptojacking traffic and analyze it to see what distinguishing characteristics can be gleaned from the first four packets of a cryptojacking flow. Based on our experiments, we conclude that the machine learning classifier random forest can correctly and efficiently detect cryptojacking traffic using the extracted discriminative aspects of network traffic.

Using data collected from academic articles, two big cryptojacking sample datasets, and 45 notable attack incidents, Tekiner et al. [15] give a comprehensive overview of cryptojacking malware. As a result, several papers offered strategies for detecting cryptojacking malware based on a wide range of dynamic/behavioral traits. However, there is no systematic analysis of the literature that provides a thorough knowledge of the new cryptojacking malware. Finally, we offer guidance to the research community in this developing area by presenting lessons gained and future avenues for study.

IoT botnets have been on the rise over the past five years, and this article Borys et al. [16] explores this phenomenon in depth. However, an IP camera by itself is not capable of generating a Distributed Denial of Service. However, more than 150,000 IP cameras in a botnet may create 1 Tbps of bandwidth. Many people are caught off guard by botnets since their attacks and infections aren't as obvious as a distributed denial of service (DDoS), and in other circumstances, these cameras and printers are used to steal information or silently mine cryptocurrencies at the expense of the IoT device owner.

CIRCUIT is a method proposed by Hong et al. [17] to accurately identify cryptojacking websites. The JavaScript memory heap is where much of our attention lies since it can withstand attempts to obfuscate the script code and because it contains data about the objects declared and their reference relationships. The script code behavior of the website is then represented by a reference flow that is extracted from the JavaScript heap. Therefore, if a website has a reference flow for cryptojacking, CIRCUIT will conclude that the website is engaging in cryptojacking. Among the 300,000 most visited websites, we discovered 1,813 that were actually cryptojacking. In addition, we modeled the reported evasion tactics and took into account the fact that features of cryptojacking websites are now visible on legitimate websites as well, providing novel insights into cryptojacking.

For in-browser cryptojacking detection, Sachan et al. [18] uses temporal characteristics like query frequency and query burst, graph-based features like degree and diameter, and non-temporal features like the string-based. We utilize them to train ML systems on data spanning from just two hours to the whole history of humankind. Based on our findings, the best performance for supervised learning is achieved by K-Means with K = 2, while the greatest performance for unsupervised learning is achieved by DecisionTrees classifier with 59.5% Recall on cryptojacked DN. Comparing the cryptojacking DNs to other known malicious DNs, similarity analysis shows little to no difference. It also shows that state-of-the-art approaches has been improved by expanding their feature sets in order to better detect cryptojacking that occurs within a web browser. Our signature-based study further reveals that throughout the months of October-December 2021, not a single official Indian Government website was compromised by cryptojackers. However, by analyzing resource use, we are able to single out 10 distinct DNs with their own unique characteristics.

The Bayesian cryptojacking detector [19] takes into account the four primary cryptojacking activity metrics: CPU load, RAM utilization, network access, and calls to cryptographic libraries. The initial step of a detector's process is to compare the relevant metrics to predetermined thresholds derived from empirical studies of cryptojackers. The extended Bayes theorem is then used to assess the conditional probability of meeting or failing to meet predetermined cryptojacking infection criteria. The likelihood of a cryptojacker's success is then determined using the compared results and conditional probabilities. The detectors then make a call based on whether or not the calculated probability exceeds a predetermined threshold. Such an analysis yields an estimated detection rate of 0.90, a false-positive error rate of 0.013, and a false-negative error rate of 0.0056. In the final section of this work, we describe ways in which the cryptojacking detector has been enhanced.

A cryptojacking detection system [20], dubbed CJDetector, was developed using characteristics of the cryptojacking process. In particular, it detects malicious mining by tracking CPU activity and inspecting function call data. This method not only identifies the attack we outlined efficiently, but it can be used in general. CJDetector's recognition precision is 99.33%. Finally, we examined cryptojacking in action by testing Alexa's top 50,000 websites. While we did find that cryptojacking was decreasing in prevalence, we also observed that it is still a significant danger to networks.

Material and method

Dataset

Time-sequenced information on actual cryptojacking attacks has been found in the Cryptojacking Attack Timeseries Dataset [11]. To mine cryptocurrency without the victim's knowledge or permission is known as "cryptojacking." This data collection is gathered so that the features, trends, and patterns of such attacks over a certain time period has been studied and comprehended.

Data Features:

- Timestamp: The time and date of the attack.
- Attack Type: The specific flavor of the cryptojacking attack.
- Location: Where exactly this attack came from.
- Affected Systems: What networks or devices were hacked and what data was stolen.
- Hashrate: The amount of processing time squandered by the adversary's mining activities.
- Duration: The time frame of the attack.
- Coin Mined: Coins of the attack's cryptocurrency that were mined.
- Victim IP: The victim's Internet Protocol address.

Figure 1 demonstrate the data distribution of the device and associated attacks. Predictor importance, also known as feature or variable importance, can be used to assess a machine learning model's prediction ability. Knowing how significant the predictors are helps the authors comprehend the model, understand what drives the predictions, and have been choose or develop features to improve it. Different algorithms and situations have evaluated feature importance differently. Additionally, various models and data sets require different techniques. Combining techniques and domain experience helps understand predictive value in a machine learning problem. The dataset includes 3 CSV files, as described below.

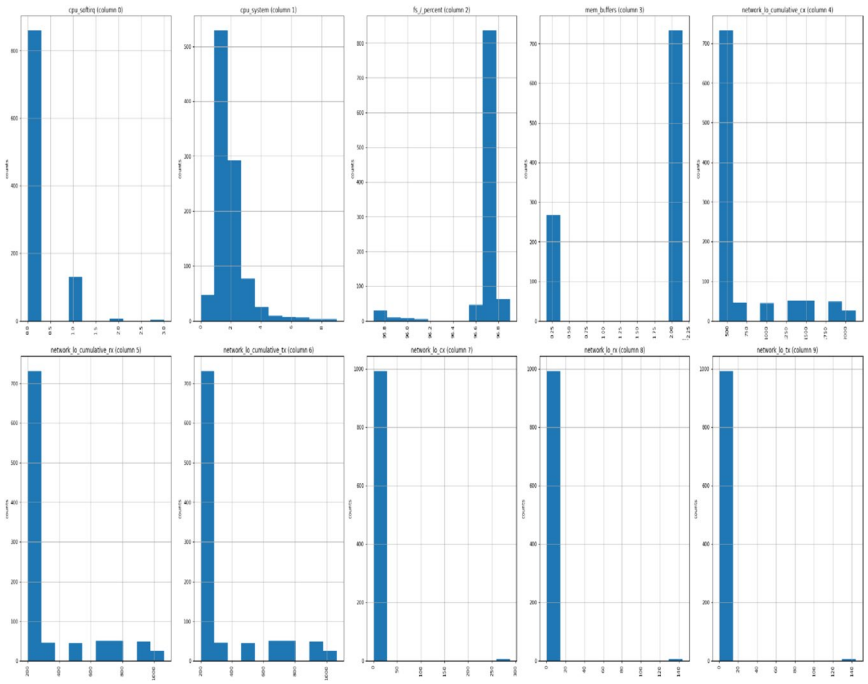

Fig. 1 Data Distribution

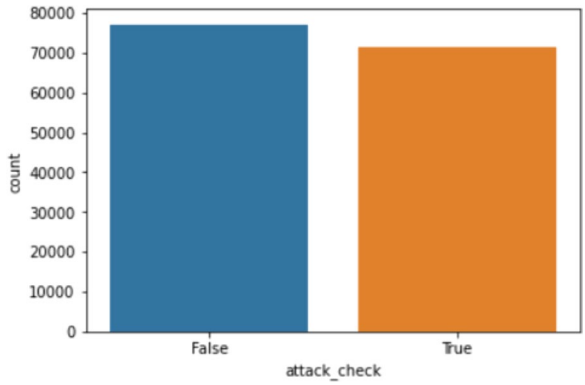

Fig. 2 Counting of attack_check

- Anormal dataset
- Normal dataset
- Complete dataset

Figure 2 demonstrate the count of attack check whether it is true or false. The true represent that the attack exists otherwise it is false. Handling missing values is a crucial step in the preprocessing of data for machine learning models. Missing data can negatively impact the performance and reliability of a machine learning model, and addressing it appropriately is essential for several reasons:

- *Avoiding Biased Analysis*: Ignoring missing values can lead to biased analysis and inaccurate model predictions. If the missing data is not handled, the model might learn patterns based on incomplete information, leading to incorrect conclusions.
- *Preserving Data Integrity*: Maintaining data integrity is vital for the accuracy of the model. Leaving missing values untreated can distort the relationships and patterns within the dataset, affecting the overall quality of the analysis.
- *Maintaining Model Performance*: Many machine learning algorithms cannot handle missing data during training. Handling missing values enables the model to be trained on a complete dataset, improving its performance and generalization on new, unseen data.

Common methods for handling missing values include imputation techniques (mean, median, or regression imputation), deletion of missing data, or more advanced methods such as multiple imputation. The choice of method depends on the nature of the data, the extent of missingness, and the specific requirements of the machine learning task at hand. Figure 3 highlights the percentage of the attack_check. IQR Method (Interquartile Range) is being applied to handle the outliers. This method is robust and less

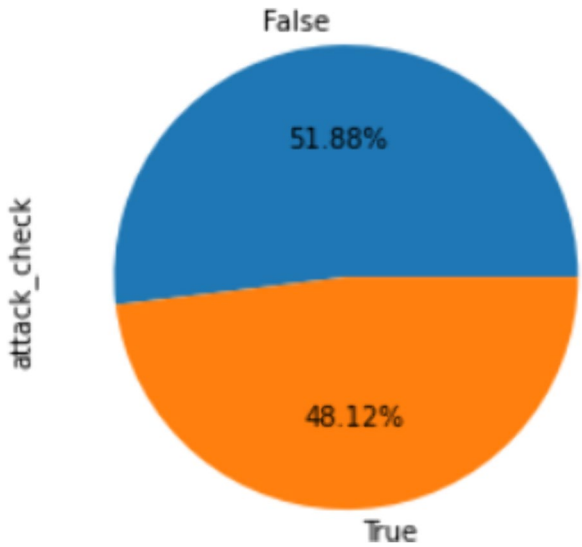

Fig. 3 % of attack_check

sensitive to extreme values compared to methods based on mean and standard deviation.

Method

Artificial neural networks (ANNs) with several layers of neurons are known as deep neural networks (DNNs). DNNs are versatile computing tools that can perform tasks such as speech recognition, language processing, and picture classification after being trained on big datasets.

Layered networks of linked neurons make up DNNs. Data to be processed by the DNN enters at the first layer, which is termed the input layer. The predictions made by the DNN are generated in the last layer, known as the output layer. The intermediate layers, known as hidden layers, are what really learn the data's salient characteristics [21].

Backpropagation is used to teach DNNs how to learn. The DNN's predictions are evaluated against the true results in backpropagation. The mistakes are sent back into the network, where they are used to fine-tune the neuron weights. Repeat this step until the DNN's predictions are satisfactory. While deep neural networks (DNNs) are a strong machine learning tool, they has been difficult to train and demand a lot of data. However, DNNs are becoming increasingly popular in machine learning research and applications because to their shown efficacy across a wide range of tasks [4]. Some of the many advantages of utilizing deep neural networks include:

- Data patterns are no longer a mystery to them.
- They have several applications.
- They've been proven useful in many different settings.

Some difficulties that arise while employing deep neural networks include:

- They are not always easy to teach.
- They need a mountain of information.
- They risk overfitting at times.

The deep neural networks are an effective machine learning technique. They are versatile and capable of learning intricate data patterns for use in many fields. However, they are notoriously difficult to train and need copious amounts of data.

Deep neural networks architecture

Artificial neural networks known as Deep Neural Networks (DNNs) include several layers between the input and output stages. These networks were developed to comprehend high-dimensional data sets and represent complicated functions [4–6, 8, 22]. An summary of their structure is as follows:

Input layer

The input layer takes in a wide variety of information that has been useful in predicting cyber attacks. Examples of such data include system logs, network traffic, and user trends. Each neuron in this layer represents a different dimension of this data.

Hidden layers

Between the input and output layers is where the majority of the network's processing takes place. A neural network's "depth" is equal to its number of hidden layers.

- Fully-Connected Layers: Every neuron in one layer communicates with every neuron in the layer above and below it.
- Convolutional Layers: Convolutional layers are mostly used for image identification tasks and apply a series of filters to the input to generate feature maps.
- Recurrent Layers: Connections in recurrent layers can loop back within the layer, making them useful for sequence prediction applications like language modeling.
- Normalization Layers: These layers help speed up the training process by standardizing the outputs of the layer below them.

- Dropout Layers: In order to avoid training a model to a specific data set, dropout layers occasionally change some of the input units to zero.

Activation functions

The system becomes non-linear once activation functions are applied. Rectified Linear Unit (ReLU), Sigmoid, and Tanh activation functions are all rather common.

Output layer

This layer's job is to generate the final prediction or categorization. In classification tasks, the number of neurons here is normally equal to the number of classes, but in regression tasks, it is equal to one [3].

Loss function and optimization

A loss function is used to measure a DNN's effectiveness. In order to train a model, this loss function is minimized using optimization procedures like stochastic gradient descent.

Backpropagation

The prediction error minimization method utilized by the network is called backpropagation. It modifies the network's weights and biases to reduce the inaccuracy.

Because of their flexibility, deep neural networks has been tailored to suit a wide range of datasets and applications. They have been used effectively in several fields, including those of natural language processing, video game playing, and picture and audio recognition.

Working

Undoubtedly, there are several processes that can be broken down into sub-steps and depicted in a thorough flow chart to describe the process of creating and executing a Deep Neural Network (DNN) for cyber attack prediction [11, 21, 23, 24]. The process is outlined in text form below:

- The first step is to define the issue that needs solving, such as categorizing cyberattacks or identifying suspicious activity in network data. The model collect raw data from the system, the network, and the users to better understand the problem.
- In Step 2, the model deal with missing values and normalize and scale the data. It selects and extracts relevant features from raw data, making it machine-learning-ready. It determines the specific DNN architecture to be employed, such as a Fully Connected DNN, a CNN, an RNN, or a combination of these. The data is sent into the DNN's input layer. Convolutional layers are used to process spatial patterns, while autoencoders are employed for outlier identification and feature representation.
- The hidden layers are put into action in Step 2.

✓ All Sub-Levels Interconnected
✓ RNN/LSTM Recurrent Layers
✓ Layers of Normalization
✓ Regularization through Dropout Layers

In this step, the model provide all of the neurons in the hidden layers non-linear activation functions like ReLU, Sigmoid, or Tanh. It have neurons representing the number of classes in the prediction issue or a single neuron representing binary classification implemented in the output layer.

- In Step 3, a loss function is selected based on the task at hand, such as the mean squared error in regression or the categorical cross entropy in classification.
- Next step 4, the optimization technique is chosen to reduce the loss, often stochastic gradient descent (SGD) or its variants such as Adam. The model is then trained with the training dataset and validated with test data to make any necessary adjustments and prevent overfitting.
- In step 5, Model performance indicators including as accuracy, precision, recall, and F1-score are used to test dataset evaluation. If the model's results satisfy constraints, it has been applied into production and use it to forecast cyber attacks in real time.
- Last step retrain and fine-tune the model when fresh data becomes available or as the nature of cyber threats changes.

While Deep Neural Networks (DNNs) have demonstrated great potential for predicting cyber attacks, they are not without their drawbacks shown in Fig. 4.

Here are some of the more significant difficulties and restrictions that may arise from employing DNNs in this setting [11]. In order to train properly, DNNs need a lot of information. The predictive capacity of the network has been jeopardized if there is insufficient high-quality, labeled data for incidents of cyber attacks [9, 10, 25, 26]. Training DNNs is resource-intensive since it requires specialized hardware like GPUs and a lot of processing power. The incomprehensibility of how DNNs arrive at their predictions has led to their being labeled "black-box" models [13]. This is a potential issue in the field of cybersecurity, since knowing the reasoning behind a prediction is often essential for making sound

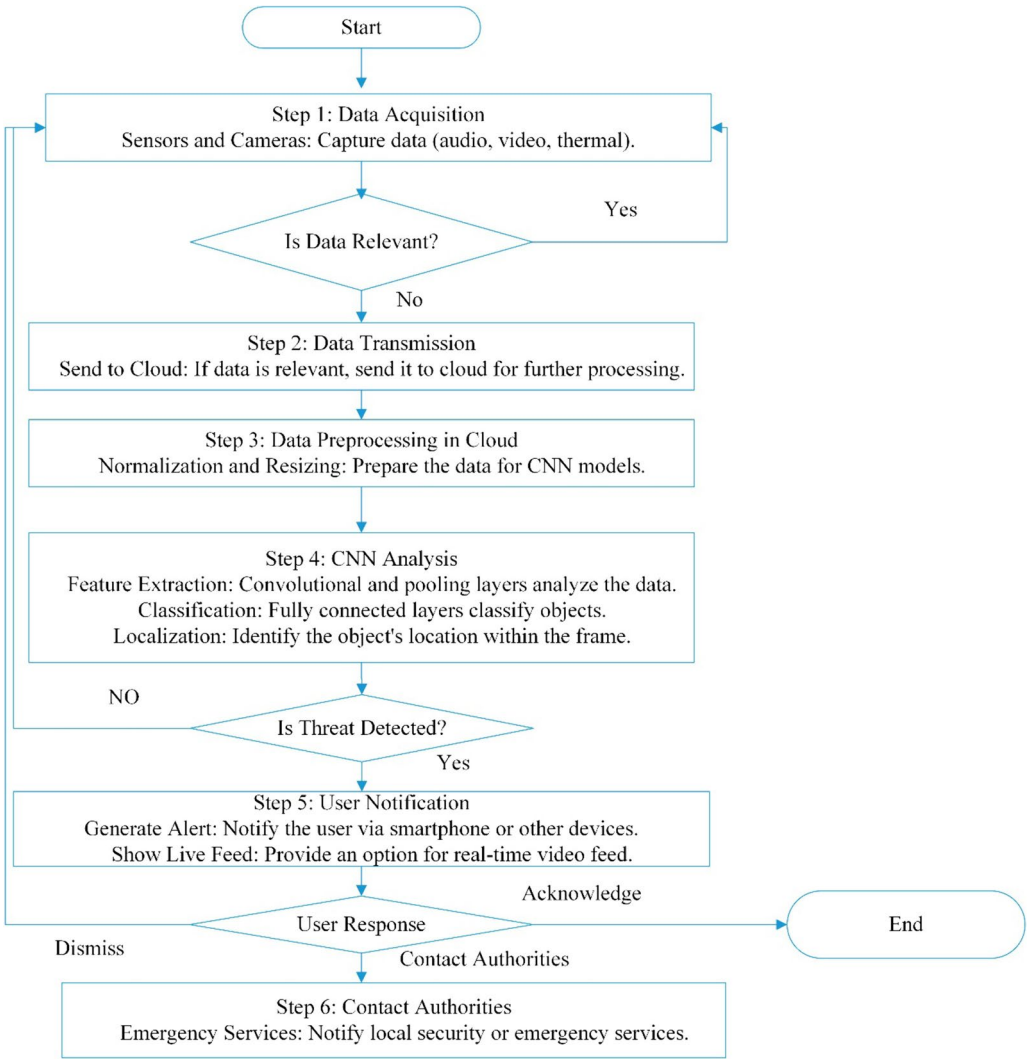

Fig. 4 Working of DNNs system

judgments [20]. When the data is unbalanced or lacking in diversity, deep learning models are more likely to overfit. Overfitting impedes the model's ability to generalize to novel input. The computing time required for training and inference by DNNs may not match real-time requirements, especially in systems that demand instant response, although they can be successful at spotting patterns suggestive of cyber threats [7, 15, 27–29].

As such, a model trained on historical data may not be enough for defending against emerging cyber threats [25]. As a result, the model needs to be updated and retrained frequently, which has been time-consuming and costly. While DNNs excel at automating feature learning, preprocessing procedures like feature extraction and selection still need for expert knowledge, especially when the input data originates from disparate sources like as logs, network flows, or system metrics [27]. There are ethical and privacy concerns since the training data may include sensitive or personally identifiable information (PII). Small changes to the input data can trick DNNs into making inaccurate predictions, making them susceptible to adversarial attacks. This is especially worrisome in the context of cybersecurity, when attackers may wilfully modify data in order to remain undetected. Understanding both the domain (cybersecurity) and the model (DNN) is necessary for the daunting task of modifying the DNN's complicated hyperparameters and architectural choices [12–14, 30–32].

These drawbacks have prompted studies into hybrid methods that integrate deep learning with more conventional forms of cybersecurity, as well as investigations into more interpretable machine learning models that can be relied upon and analyzed with more precision. Overfitting occurs when a machine learning

model learns the training data too well, accumulating noise or random oscillations instead of patterns. When applied to unknown data, this may reduce generalization performance. Deep neural networks and sophisticated decision trees can recall all training data, including noise. Insufficient training data can cause model overfitting and poor generalizability to new data. Overfitting occurs more often in models that allow too much complexity. For instance, a high-degree polynomial regression fits training data well but generalizes poorly. Dropout training was employed for this work. Each cycle, dropout randomly eliminates neurons from the network to minimize overreliance on one neuron.

Proposed methodology

In many machine learning applications, accuracy is more important than speed, and optimizing a DNN may help enhance both. It is more probable that a model will correctly predict or classify data if it has been adequately optimized. Optimization methods can greatly hasten the learning procedure. The ability to rapidly cycle through several training models and hyperparameter settings is crucial for experimental purposes [19, 33, 34]. The reduced memory and processing needs of an optimized model make it easier and cheaper to roll out to production settings. In the context of Deep Neural Networks (DNNs), optimizers are algorithms used to minimize (or maximize) the objective function J(θ)) over the neural network parameters θ, which could include weights and biases. The objective function, often referred to as the loss function or cost function, measures how well the neural network performs on the dataset.

Common types of optimizers

1. Stochastic Gradient Descent (SGD): The simplest and most widely used optimization algorithm. It updates each parameter θi according to the Eq. 1:

$$\theta \mathrm{i} = \theta \mathrm{i} - \alpha \frac{\partial \mathrm{J}}{\partial \theta \mathrm{i}} \tag{1}$$

where α is the learning rate.

2. Momentum: A variation of SGD that takes into account the past gradients to smooth out the update as per the Eqs. 2 and 3:

$$\mathrm{v} = \beta \mathrm{v} - \alpha \nabla \mathrm{J} \tag{2}$$

$$\theta = \theta + \mathrm{v} \tag{3}$$

where β is the momentum term.

3. Adagrad: It adapts the learning rate during training for each parameter θi depending on the historical gradient information for that parameter.

The Adagrad update formula for parameter θ at iteration t in the Eq. 4:

$$\theta_{t+1} = \theta_t - \left(\alpha / \left(\sqrt{(G_t + \varepsilon)}\right)\right) * \nabla \theta f(\theta_t) \tag{4}$$

Where:
θt_{+1}: The updated parameter at iteration t+1.
θ_t: The parameter at iteration t.
α: The learning rate.
G_t: The diagonal matrix containing the sum of squared historical gradients up to iteration t.
ε: A small constant (usually a small positive number, like 1e-8) added for numerical stability.
$\nabla \theta f(\theta_t)$: The gradient of the loss function with respect to parameter θ at iteration t.

4. RMSprop: Similar to Adagrad but introduces an exponentially decaying average to give more weight to recent gradients.

The RMSprop update formula for parameter θ at iteration t in the eq. 5:

$$\theta_{t+1} = \theta_t - \left(\alpha / \left(\sqrt{(G_t + \varepsilon)}\right)\right) * \nabla \theta f(\theta_t) \tag{5}$$

Where:
θ_{t+1}: The updated parameter at iteration t+1.
θ_t: The parameter at iteration t.
α: The learning rate.
G_t: The diagonal matrix containing the exponentially weighted moving average of squared gradients up to iteration t.
ε: A small constant (usually a small positive number, like 1e-8) added for numerical stability.
$\nabla \theta f(\theta_t)$: The gradient of the loss function with respect to parameter θ at iteration t.

5. Adam: Combines the ideas of Momentum and RMSprop. It keeps an exponentially decaying average of past gradients and the element-wise square of past gradients.

The Adam update formula for parameter θ at iteration t in the eqs. 6, 7, 8, 9 and 10:

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla\theta f(\theta_t) \quad (6)$$

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * (\nabla\theta f\theta_t))^2 \quad (7)$$

$$m_{t\_hat} = m_t/(1 - \beta_1{}^t) \quad (8)$$

$$v_{t\_hat} = v\_t/(1 - \beta_2{}^t) \quad (9)$$

$$\theta_{t+1} = \theta_t - (\alpha/(\sqrt{(v_{t\_hat} + \varepsilon)})) * m_{t\_hat} \quad (10)$$

Where:

θ_{t+1}: The updated parameter at iteration t+1.

θ_t: The parameter at iteration t.

α: The learning rate.

β_1 and β_2: Exponential decay rates for 1st and 2nd moment estimates, respectively.

ε: A small constant for numerical stability.

m_t: The 1st moment estimate (mean of gradients) at iteration t.

v_t: The 2nd moment estimate (uncentered variance of gradients) at iteration t.

t: The current iteration.

$\nabla\theta f(\theta^t)$: The gradient of the loss function with respect to parameter θ at iteration t.

The Eqs. 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10 define the working of the optimizers applied in neural networks. These equations involves various variables that plays major role in predictions of attack. Quantization and pruning are two methods for optimizing and minimizing the footprint of deep neural networks (DNNs) for usage on low-powered mobile devices [17, 18, 35–38]. When training DNNs, however, a second-order optimization approach called AdaHessian has been employed to boost training efficiency and convergence. Let's talk quickly about each of these methods:

- AdaHessian Optimization: AdaHessian is an optimization technique that broadens the scope of SGD (Stochastic Gradient Descent) and Adam, two classic gradient-based optimization tools. The training rates for each parameter are adaptively modified using second-order information. As an effective alternative to conventional optimization techniques, this strategy has the potential to speed up training for deep neural networks and enhance convergence. AdaHessian's hyperparameters, such learning rate, weight decay, and momentum, need to be tweaked for optimal performance.
- Quantization: By decreasing the accuracy of model parameters like weights and activations, we have quantization. For this reason, it is common practice in deep learning to transform floating-point quantities to fixed-point or integer representations with a smaller bit width. A model can be quantized, for instance, such that it operates on 8-bit integers rather than 32-bit floating-point values. This helps conserve memory and expedite inference on computers with efficient integer-processing capabilities. Common methods for quantizing DNNs include post-training quantization and training with quantization in mind.
- Pruning: In order to improve the performance of a trained DNN, it can be "pruned," which means that unused connections (weights) or even whole neurons (channels) are removed. The model's inference time and memory requirements has been decreased by pruning. It can also help the model generalize better by decreasing the amount of overfitting.

When accuracy drops as a result of pruning, it's necessary to retrain or fine-tune meticulously. Effective

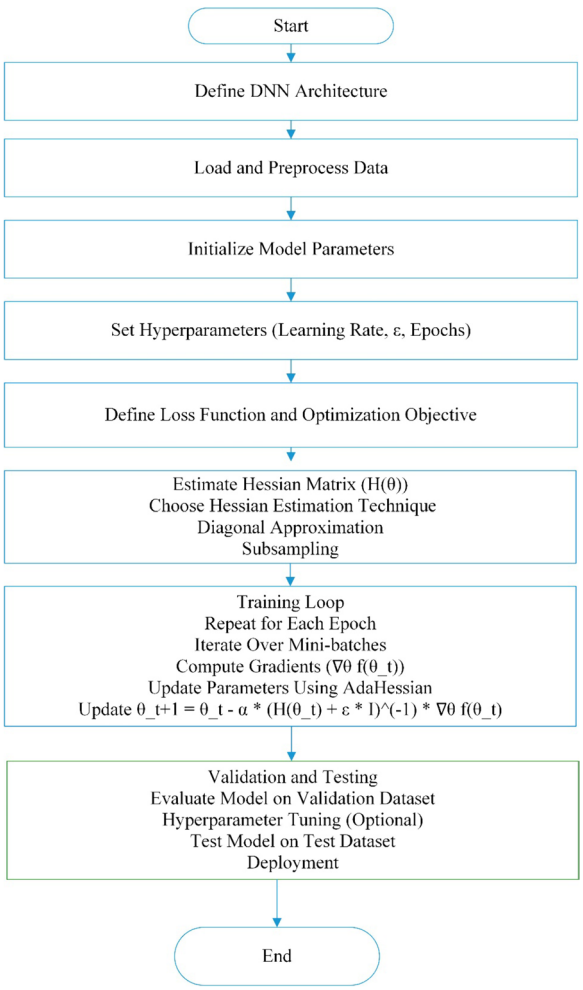

Fig. 5 Working of proposed model

optimization and deployment of DNNs has been achieved by combining AdaHessian with quantization and pruning. In proposed method, these methods can be coupled in Fig. 5 as follows:

Deep neural networks (DNNs) require many operations to combine pruning, post-training quantization, and AdaHessian optimization. The goal of this approach is to produce a DNN that is as small and efficient as possible without sacrificing performance. Here is an approach that takes advantage of all these methods:

Step 1. Initial Model Training with AdaHessian: The AdaHessian optimization technique is used to train the deep neural network at this stage. In this stage, we optimize the first version of the model as much as possible. It guarantees that a representative dataset is used for training and that the hyperparameters are adjusted appropriately.
Step 2. Model Evaluation after Training: This phase follows training and consists of an evaluation of the trained model's accuracy and performance using a validation dataset.
Step 3. Post-Training Quantization: In this process, the weights and activations of the trained DNN are transformed into representations with a smaller bit width. It is capable of employing quantization strategies, such as those used in TensorFlow and PyTorch. If you want to discover the optimal balance between model size and inference speed, you should try out various quantization levels (e.g., 8-bit, 4-bit).
Step 4. Importance Scoring for Pruning: The next thing to do is to assign weights and neuron weights in the quantized model important scores. Using these ratings, we may narrow down which parameters need to be trimmed. It may make it possible to calculate significance scores using techniques like magnitude-based pruning, saliency-based pruning, and Hessian-based pruning. AdaHessian is useful for calculating significance using the Hessian metric.
Step 5. Pruning Decision: In this phase, we established a cutoff value or criterion dependent on the significance ratings. At this cutoff, parameters (weights or neurons) are either kept or removed. Insignificant parameters are those that fall below the cutoff. Based on the model architecture and pruning approach, the author can choose to prune either individual weights, neurons, or channels.
Step 6. Pruning: Now it's time to prune the quantified model according to the established standards. If a neuron or its associated connection has a weight below the pruning threshold, it will be removed and make the necessary changes to the model's structure (such as deleting individual neurons and modifying the layers above and below them).
Step 7. Fine-Tuning after Pruning: this action to restore any accuracy lost as a result of pruning, retrain the model. Start training with a lower learning rate using the leftover weights from the trimmed model and adjusting the model's fine points after trimming and quantization has been done.
Step 8. Quantified and pruned model evaluation: This phase involves testing the improved model on a validation set. It evaluates its performance in relation to that of the original model and the quantized model, taking into account accuracy and other important criteria.
Step 9. Iterate if Necessary: The process of trimming and fine-tuning may require iterations based on the outcomes of step 8. To find the sweet spot between model size and accuracy, you may play around with hyperparameters like pruning threshold and fine-tuning time.
Step 10. Deployment: After the performance of the trimmed and quantized model has been evaluated and deemed satisfactory, it has been deployed to the target platform after careful consideration of the necessary hardware and software.
Step 11. Monitoring and Maintenance: Quantization and pruning may involve trade-offs that effect real-world performance, thus it is important to regularly check the performance of the deployed model in production. It's ready to fine-tune or retrain the model as needed to accommodate new or different data sets or parameters.

AdaHessian optimization, post-training quantization, and pruning all contribute to a compact and efficient DNN that can function in contexts with limited resources without sacrificing accuracy.

Pseudo code of proposed hybrid model

The Pseudo code of the proposed hybrid model is as follows.

```
// Pseudo code Proposed hybrid Model
Input: Cryptojacking Dataset
Output: Attack class
1. Apply data Pre-processing
2. Define DNN Architecture
2.1 model = Sequential()
2.2 model.add(Dense(128, activation = 'relu', input_shape = (feature_dim,)))
2.3 model.add(Dense(64, activation = 'relu'))
2.4 model.add(Dense(32, activation = 'relu'))
2.5 model.add(Dense(1, activation = 'sigmoid'))
3. Compile Model with AdaHessian Optimizer
3.1 model.compile(loss = 'binary_crossentropy', optimizer = AdaHessianOptimizer(), metrics = ['accuracy'])
4. Train the Model
4.1 model.fit(X_train, y_train, epochs = 50, batch_size = 32, validation_split = 0.2)
5. Model Evaluation
# Before Quantization
5.1 model.evaluate(X_test, y_test)
6. Post-Training Quantization
6.1 quantize_model = tfmot.quantization.keras.quantize_model
6.2 q_aware_model = quantize_model(model)
6.3 q_aware_model.compile(loss = 'binary_crossentropy',
6.4 optimizer = AdaHessianOptimizer(), metrics = ['accuracy'])
7. Fine-Tuning after Quantization
7.1 q_aware_model.fit(X_train, y_train, batch_size = 32, epochs = 10, validation_split = 0.2)
8. Re-Evaluate the Model
# After Quantization
8.1 q_aware_model.evaluate(X_test, y_test)
```

Finding the optimal configuration for your application requires some trial and error. In order to forecast cryptojacking attacks, this study merges three separate methods into a unified framework: pruning, post-training quantization, and AdaHessian optimization. The innovative aspect is the combination and complementarity of various techniques to improve speed and precision.

- *Optimal AdaHessian Functions:* This paper highlights the new use of AdaHessian, a second-order optimization method, to the problem of foreseeing cryptojacking attacks. AdaHessian gives you an edge over conventional optimization strategies due to its flexibility in adjusting learning rates and allowing you to investigate the loss landscape in novel ways.
- *Prioritizing Effectiveness:* The primary emphasis of this study is on efficiency without sacrificing the accuracy of predictions. This focus on low-overhead detection techniques is unusual in the field of cryptojacking attack forecasting.
- *Quantification for Efficient Use of Resources:* it focus on post-training quantization to cut down on model size and compute needs, making the model appropriate for contexts with limited resources. Quantization approach combination with machine learning for security is an underdeveloped area.
- *Value-Based Editing:* Insights from AdaHessian are utilized to create a unique scoring method for determining which branches to remove. Your method stands out since you use many methods to reduce the size of the model without sacrificing accuracy.
- *Analyzing Data Sets in the Real World:* In this study, we highlight the fact that we have extensively evaluated your methodology using real-world network traffic facts to prove its practicality and efficacy.
- *Analyzing the Differences:* This paper presents a comprehensive comparison of your strategy to existing approaches to predicting cryptojacking attacks. Bring to light how proposed unique method enhances efficiency and precision.
- *Threat to Safety:* The paper delves into the wider security consequences of your efforts. Highlight how your method's efficiency benefits can aid in the identification of cryptojacking attacks in a way that is both effective and scalable, hence improving cybersecurity.
- *Application in Real Life:* This study emphasizes the possibility for your solution to be used in practice on edge devices, routers, and network gateways, demonstrating the practical significance of your research.

The authors prove the originality and relevance of their method for forecasting cryptojacking attacks utilizing pruning, post-training quantization, and AdaHessian optimization by addressing these concerns and highlighting the specific contributions and innovations of their study.

Results and analysis

Experimental setup

Here we provide the outcomes of the simulations conducted on the cryptojacking dataset. The current models and the one that is being proposed are built using Python and its essential libraries, such as Numpy, Sci-kit, Matplot, Pandas, and Tensor Flow. This is all run on a computer with the following specifications: 16 GB of RAM, Core i7, 10700 processor, CPU @ 3.7Ghz, and Windows 11 operating system [16, 20, 39–42]. We transformed the dataset into picture datasets after doing the necessary pre-processing. Using k-fold cross-validation, the dataset is partitioned into two parts: training and testing. We have conducted a binary classification on the dataset using both the proposed and current deep learning models. The simulation parameters are displayed in Table 1.

Artificial neural networks use activation functions to mathematically operate on each neuron in a layer. It lets the network learn and approximate complex data by adding non-linearity. Different activation functions behave differently. Selecting an activation function should take into account the problem's characteristics, neural network architecture, and task performance. To determine the ideal activation functions for a neural network,

Table 1 Parameters used for simulation

Parameters	Details
CNN Model	Transfer Learning and Light-weight CNN (mobile-V3) with SVM
No Training Layers	100–250
Epochs	50–100
Optimizer used	**Stochastic Gradient Descent (SGD) Optimizer**
Model learning rate	**0.001**
Loss function	Categorical Cross Entropy (CCE) function
Activation function	SoftMax, ReLu
Batch size	256
Hidden Layer Architecture	(256,128,64)

experiments must consider each function's problem-related attributes [43–48].

Experimental results

This dataset replicates the real-world data in PCAPs by including benign and the most recent examples of common attacks. It additionally contains the findings of an analysis of the network traffic performed with CIC_Flow_Meter, complete with labelled flows organized according to the protocols, date and time stamp, origin and destination IP addresses, the source with destination port numbers, and attack. Figure 6 show that when trying to make sense of the connections between the many elements and qualities that make up a dataset, a correlation matrix has

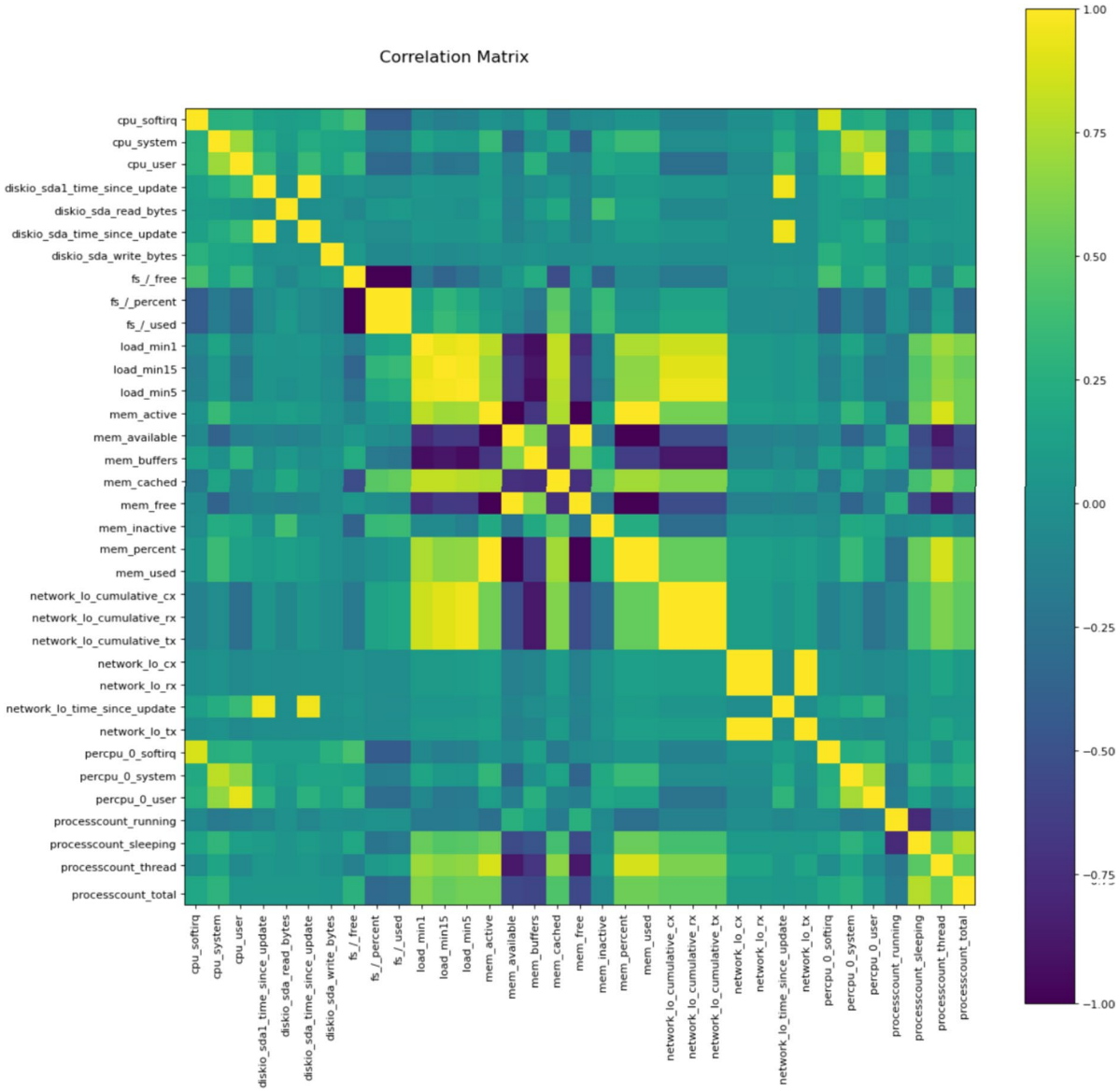

Fig. 6 Corelation matrix

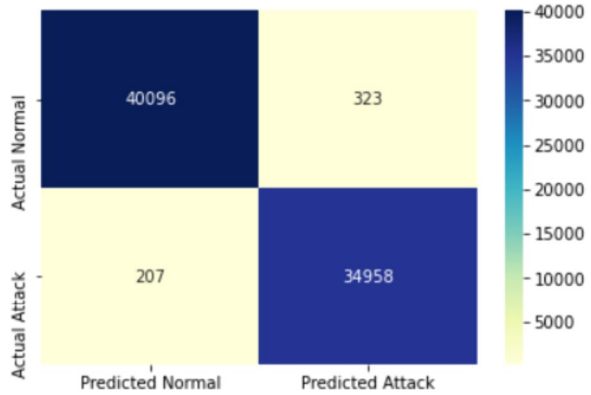

Fig. 7 Confusion matrix of proposed model

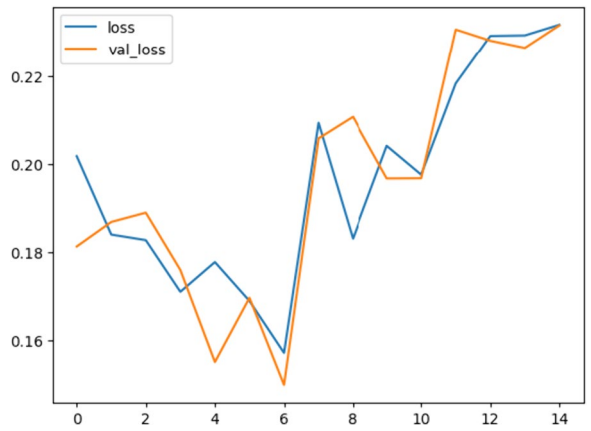

Fig. 8 Loss curve (Binary Classification) for the proposed model

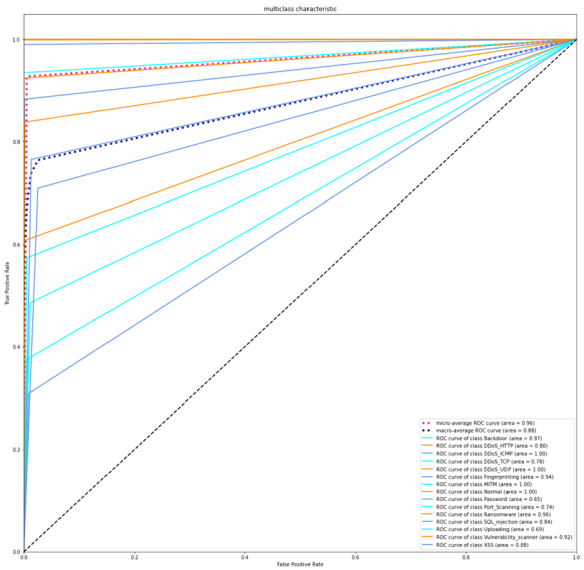

Fig. 9 ROC curve (Binary Classification) for proposed model

Table 2 Experimental results comparison

Optimizers	Recall	Precision	F1-Score
Stochastic Gradient Descent (SGD)	91.13%	93.62%	93.59%
Momentum	95.39%	96.69%	95.81%
Adagrad	94.85%	96.99%	96.48%
RMSprop	96.39%	96.79%	97.96%
Adam	97.05%	95.09%	96.05%
Proposed Method	99.72%	98.93%	99.12%

been quite helpful. Patterns and interdependencies that has been symptomatic of cyber attacks or abnormalities can be uncovered with the use of such a matrix.

In Fig. 7 we have measured the Confusion matrix results for all the methods, i.e., existing and proposed on the given dataset. False negatives can have serious consequences. False negatives in medical diagnostics can lead to missed or delayed treatment of actual illnesses. Security applications' intrusion detection systems might overlook serious threats if they generate false negatives. Model or test threshold decisions effect the false positive/negative trade-off. Some apps prioritize one over the other; adjusting the threshold can balance them.

Figure 8 shows an accuracy and loss curve, and Fig. 9 shows the ROC curve for given dataset for the proposed model.

Table 2 presents experimental results comparison for existing and proposed methods. CNN with Stochastic Gradient Descent (SGD) optimizer achieves a recall of 91.13%, precision 93.62%, F1-Score 93.59%. Momentum optimizer achieves Recall 95.39%, Precision 96.69% and F1-Score 95.81%. Adagrad optimizer achieves Recall 94.85%, Precision 96.99% and F1-Score 96.48%.

RMSprop optimizer achieves Recall 96.39%, Precision 96.79% and F1-Score 97.96%. The proposed method achieves Recall 99.72%, Precision 98.93% and F1-Score 99.12%. The model's high F1 score reflects a good recall-precision balance, allowing it to recognize positive and negative cryptojacking efforts. MEC applications improve bitcoin exchange security. AdaHessian optimization reduces false positives and negatives. To minimize unnecessary disruptions, limit false positives so legitimate processes are not mistaken for assaults. Reducing false negatives improves the model's cryptojacking detection and warning. High F1 scores show the model's cryptojacking detection skill. Enhancing the system's ability to notice and respond quickly to threats reduces the risk of crypto exchange attacks.

We have calculated various performance measuring parameters for existing and proposed methods. Table 3 presents the accuracy results prescribed dataset for existing and proposed methods for different classes. For class

Table 3 Accuracy results comparison for classification

Attack Class	Stochastic Gradient Descent (SGD)	Momentum	Adagrad	RMSprop	Adam	Proposed Hybrid model
Not malicous	89.12	91.07	85.95	87.68	92.34	98.85
Malicous	90.13	90.95	85.63	85.97	91.35	99.13

'not malicous', the proposed method achieves 98.85% accuracy, for ' 'not malicous" 99.13% accuracy.

Discussion

The increasing complexity of Deep Neural Networks (DNNs) requires improved optimization algorithms for practical implementation, especially in cybersecurity, where real-time decision-making is crucial. Our work uses pruning, post-training quantization, and AdaHessian optimization to solve the computational and security issues of DNNs that predict Cryptojacking attempts.

AdaHessian optimization

AdaHessian optimization in DNN training is our first important contribution. Despite its relevance in improving generalization and convergence, SGD, Adam, and RMSprop neglect the loss landscape's curvature. By adding second-order optimization, AdaHessian enhances the model's loss landscape navigation. Our research showed that AdaHessian's Cryptojacking prediction accuracy improved, showing its cybersecurity value.

Pruning

Network pruning removed unnecessary connections and neurons after DNN training. Pruning reduces model size and strengthens the network against overfitting. Edge devices, where computational resources are few, require a smaller form.

Post-training quantization

Post-training quantization converted float data to integers with a reduced bit width to minimize model size. The inference process was sped up and the memory footprint was considerably reduced without losing accuracy. Post-training quantization is desirable for current models since it does not need network reteaching.

Cryptojacking attack prediction

Throughout the trial, our model's predictive ability to prevent Cryptojacking assaults was critical. The model was quick and accurate when various optimization procedures were coupled. The approach is important in cybersecurity, where time and precision are crucial.

Limitations

Our study results are promising, but with limitations. No one has investigated the model's resilience to malevolent actors. Our technology has only been tested on some Cryptojacking attempts, hence its applicability to other cybercrimes is unknown.

This research strongly supports implementing complex optimization approaches into cybersecurity DNNs. We used AdaHessian optimization, pruning, and post-training quantization to create a computationally efficient and accurate Cryptojacking prediction system. More research is needed to validate the framework's resilience to hostile assaults and expand its cybersecurity applications.

Conclusion

To predict Cryptojacking attacks, we examined the challenging challenges of improving DNNs for cybersecurity applications in this research. Traditional DNNs are powerful, but their high computational cost and large model sizes make them unsuitable for resource-constrained applications. Pruning, post-training quantization, and AdaHessian optimization were used to solve these challenges. We found that AdaHessian optimization improves training, enabling Cryptojacking attack prediction with minimal computational power. Next, we pruned superfluous neurons and connections to reduce model size without impacting accuracy. Finally, post-training quantization reduced memory footprint and increased inference speed, making the model ideal for resource-constrained applications like edge computing in real life. Neural networks can predict cryptojacking attempts, which is useful. Cryptojacking involves unlawful cryptocurrency mining on computers. Attacks can be prevented with early detection. AdaHessian optimization and optimized neural networks enhance training time and efficiency. Optimization method AdaHessian accelerates neural network convergence. Crypto exchange operations must be protected from financial and reputational losses. Proactive security can be improved by machine learning prediction models. Many applications have come from the study. Before deploying machine learning models in cyberspace, they stress the need for extensive optimization and security methods. Then, they demonstrate that high processing cost and model

size reductions do not affect prediction accuracy. We conclude that our technique lays the groundwork for cyber security research using cutting-edge optimization methods. We use advanced optimization techniques and real cybersecurity applications to offer a powerful, inexpensive, and scalable cryptojacking solution. Our work optimizes machine learning and safeguards the digital environment against new crime.

Acknowledgements
The authors would like to acknowledge the support of EIAS (Emerging Intelligent Autonomous Systems) Data Science Lab, Prince Sultan University, KSA.

Authors' contributions
The authors confirm contribution to the paper as follows: study conception and design: UR, SK and ND; data collection: KS and SRK; analysis and interpretation of results: SS, MS and FA; draft manuscript preparation: ND, SRK and FA. All authors reviewed the results and approved the final version of the manuscript.

Funding
This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Availability of data and materials
Publicly available datasets were analyzed in this study.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare no competing interests.

Received: 9 December 2023 Accepted: 9 March 2024
Published online: 18 March 2024

References

1. Eskandari S, Leoutsarakos A, Mursch T, Clark J (2018) A first look at browser-based cryptojacking. Proceedings - 3rd IEEE European Symposium on Security and Privacy Workshops, EURO S and PW. pp 58–66. https://doi.org/10.1109/EuroSPW.2018.00014
2. Yulianto AD, Sukarno P, Warrdana AA, Al Makky M (2019) Mitigation of cryptojacking attacks using taint analysis. 2019 4th International Conference on Information Technology, Information Systems and Electrical Engineering, ICITISEE 2019. pp 234–238. https://doi.org/10.1109/ICITISEE48480.2019.9003742
3. Burgess J, Carlin D, O'Kane P, Sezer S (2019) MANiC: Multi-step assessment for crypto-miners. 2019 International Conference on Cyber Security and Protection of Digital Services, Cyber Security. pp 1–8. https://doi.org/10.1109/CyberSecPODS.2019.8885003
4. Saad M, Khormali A, Mohaisen A (2019) Dine and dash: static, dynamic, and economic analysis of in-browser cryptojacking. ECrime Researchers Summit, ECrime. pp 1–12. https://doi.org/10.1109/eCrime47957.2019.9037576
5. Munoz JZI, Suarez-Varela J, Barlet-Ros P (2019) Detecting cryptocurrency miners with NetFlow/IPFIX network measurements. 2019 IEEE International Symposium on Measurements and Networking, M and N 2019 - Proceedings. https://doi.org/10.1109/IWMN.2019.8804995
6. Tahir R, Durrani S, Ahmed F, Saeed H, Zaffar F, Ilyas S (2019) The Browsers Strike Back: Countering Cryptojacking and Parasitic Miners on the Web. Proceedings - IEEE INFOCOM. pp 703–711. https://doi.org/10.1109/INFOCOM.2019.8737360
7. Lachtar N, Elkhail AA, Bacha A, Malik H (2020) A cross-stack approach towards defending against cryptojacking. IEEE Comput Archit Lett 19(2):126–129. https://doi.org/10.1109/LCA.2020.3017457
8. Tanana D, Tanana G (2020) Advanced behavior-based technique for cryptojacking malware detection. 2020 14th International Conference on Signal Processing and Communication Systems, ICSPCS 2020 - Proceedings. pp 16–19. https://doi.org/10.1109/ICSPCS50536.2020.9310048
9. Di Tizio G, Nam Ngo C (2020) Are you a favorite target for cryptojacking? A case-control study on the cryptojacking ecosystem. Proceedings - 5th IEEE European Symposium on Security and Privacy Workshops. Euro S and PW 2020:515–520. https://doi.org/10.1109/EuroSPW51379.2020.00075
10. Tanana D (2020) Behavior-based detection of cryptojacking malware. Proceedings - 2020 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology, USBEREIT. pp 543–545. https://doi.org/10.1109/USBEREIT48449.2020.9117732
11. Gomes G, Dias L, Correia M (2020) CryingJackpot: network flows and performance counters against cryptojacking. 2020 IEEE 19th International Symposium on Network Computing and Applications, NCA. https://doi.org/10.1109/NCA51143.2020.9306698
12. Lachtar N, Elkhail AA, Bacha A, Malik H (2021) An application agnostic defense against the dark arts of cryptojacking. Proceedings - 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN. pp 314–325. https://doi.org/10.1109/DSN48987.2021.00044
13. Caprolu M, Raponi S, Oligeri G, Di Pietro R (2021) Cryptomining makes noise: detecting cryptojacking via Machine Learning. Comput Commun 171:126–139. https://doi.org/10.1016/j.comcom.2021.02.016
14. Hu H, Shu Z, Song X, Cheng G, Gong J (2021) Detecting cryptojacking traffic based on network behavior features. 2021 IEEE Global Communications Conference, GLOBECOM 2021 - Proceedings. pp 1–6. https://doi.org/10.1109/GLOBECOM46510.2021.9685085
15. Tekiner E, Acar A, Uluagac AS, Kirda E, Selcuk AA (2021) SoK: Cryptojacking malware. Proceedings - 2021 IEEE European Symposium on Security and Privacy, Euro S and P. pp 120–139. https://doi.org/10.1109/EuroSP51992.2021.00019
16. Borys A, Kamruzzaman A, Thakur HN, Brickley JC, Ali ML, Thakur K (2022) An evaluation of IoT DDoS cryptojacking malware and Mirai Botnet. 2022 IEEE World AI IoT Congress, AIIoT. pp 725–729. https://doi.org/10.1109/AIIoT54504.2022.9817163
17. Hong H, Woo S, Park S, Lee J, Lee H (2022) Circuit: a Javascript memory heap-based approach for precisely detecting cryptojacking websites. IEEE Access 10:95356–95368. https://doi.org/10.1109/ACCESS.2022.3204814
18. Sachan RK, Agarwal R, Shukla SK (2022) DNS based in-browser cryptojacking detection. 2022 4th International Conference on Blockchain Computing and Applications, BCCA. pp 259–266. https://doi.org/10.1109/BCCA55292.2022.9922245
19. Gaidamakin N, Tanana D (2022) Naïve Bayes cryptojacking detector. Proceedings - 2022 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology, USBEREIT 2022. pp 259–262. https://doi.org/10.1109/USBEREIT56278.2022.9923349
20. Xu G, Dong W, Xing J, Lei W, Liu J, Gong L, Feng M, Zheng X, Liu S (2023) Delay-CJ: A novel cryptojacking covert attack method based on delayed strategy and its detection. Digit Commun Netw. https://doi.org/10.1016/j.dcan.2022.04.030
21. Varlioglu S, Gonen B, Ozer M, Bastug M (2020) Is cryptojacking dead after coinhive shutdown? Proceedings - 3rd International Conference on Information and Computer Technologies, ICICT. pp 385–389. https://doi.org/10.1109/ICICT50521.2020.00068
22. Nahmias D, Cohen A, Nissim N, Elovici Y (2019) TrustSign: trusted malware signature generation in private clouds using deep feature transfer learning. Proceedings of the International Joint Conference on Neural Networks. pp 1–8. https://doi.org/10.1109/IJCNN.2019.8851841
23. Aktepe S, Varol C, Shashidhar N (2020) MiNo: the chrome web browser add-on application to block the hidden cryptocurrency mining activities. 8th International Symposium on Digital Forensics and Security, ISDFS. https://doi.org/10.1109/ISDFS49300.2020.9116443

24. Romano A, Zheng Y, Wang W (2020) MinerRay: semantics-aware analysis for ever-evolving cryptojacking detection. Proceedings - 2020 35th IEEE/ACM International Conference on Automated Software Engineering, ASE. pp 1129–1140. https://doi.org/10.1145/3324884.3416580
25. Gomes F, Correia M (2020) Cryptojacking detection with CPU Usage Metrics. 2020 IEEE 19th International Symposium on Network Computing and Applications, NCA. https://doi.org/10.1109/NCA51143.2020.9306696
26. Nukala VSKA (2020) Website Cryptojacking Detection Using Machine Learning : IEEE CNS 20 Poster. 2020 IEEE Conference on Communications and Network Security, CNS. https://doi.org/10.1109/CNS48642.2020.9162342
27. Caviglione L, Mazurczyk W, Repetto M, Schaffhauser A, Zuppelli M (2021) Kernel-level tracing for detecting stegomalware and covert channels in Linux environments. Comput Netw 191:108010. https://doi.org/10.1016/j.comnet.2021.108010
28. Nunes P, Antunes M, Silva C (2021) Evaluating cybersecurity attitudes and behaviors in Portuguese healthcare institutions. Proc Comput Scie 181(2019):173–181. https://doi.org/10.1016/j.procs.2021.01.118
29. Piasecki S, Urquhart L, McAuley PD (2021) Defence against the dark artefacts: Smart home cybercrimes and cybersecurity standards. Comput Law Secur Rev 42:105542. https://doi.org/10.1016/j.clsr.2021.105542
30. Guo H, Yu X (2022) A survey on blockchain technology and its security. Blockchain Res Appl 3(2):100067. https://doi.org/10.1016/j.bcra.2022.100067
31. Markopoulou D, Papakonstantinou V (2021) The regulatory framework for the protection of critical infrastructures against cyberthreats: Identifying shortcomings and addressing future challenges: the case of the health sector in particular. Comput Law Secur Rev 41:105502. https://doi.org/10.1016/j.clsr.2020.105502
32. Slijepčević D, Henzl M, Daniel Klausner L, Dam T, Kieseberg P, Zeppelzauer M (2021) k-Anonymity in practice: how generalisation and suppression affect machine learning classifiers. Comput Secur 111:102488. https://doi.org/10.1016/j.cose.2021.102488
33. Szczepaniuk EK, Szczepaniuk H (2022) Analysis of cybersecurity competencies: recommendations for telecommunications policy. Telecommunications Policy 46(3):102282. https://doi.org/10.1016/j.telpol.2021.102282
34. Wang E, Zurowski S, Duffy O, Thomas T, Baggili I (2022) Juicing V8: a primary account for the memory forensics of the V8 JavaScript engine. Forensic Sci Int Digit Investig 42:301400. https://doi.org/10.1016/j.fsidi.2022.301400
35. Adjibi BV, Mbodji FN, Bissyande TF, Allix K, Klein J (2022) The devil is in the details: unwrapping the cryptojacking malware ecosystem on android. Proceedings - 2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation, SCAM. pp 153–163. https://doi.org/10.1109/SCAM55253.2022.00023
36. Chen L, Xia Y, Ma Z, Zhao R, Wang Y, Liu Y, Sun W, Xue Z (2022) SEAF: a Scalable, Efficient, and Application-independent Framework for container security detection. J Inform Sec Appl 71:103351. https://doi.org/10.1016/j.jisa.2022.103351
37. Varlioglu S, Elsayed N, Elsayed Z, Ozer M (2022) The dangerous combo: fileless malware and cryptojacking. Conference Proceedings - IEEE SOUTHEASTCON. pp 125–132. https://doi.org/10.1109/SoutheastCon48659.2022.9764043
38. Wu MH, Huang JH, Chen JX, Wang HJ, Chiu CY (2022) Machine Learning to Identify Bitcoin Mining by Web Browsers. 2022 IEEE 2nd International Conference on Computation, Communication and Engineering, ICCCE. pp 66–69. https://doi.org/10.1109/ICCCE55785.2022.10036239
39. Cabrera-Arteaga J, Monperrus M, Toady T, Baudry B (2023) WebAssembly diversification for malware evasion. Comput Secur 131:103296. https://doi.org/10.1016/j.cose.2023.103296
40. Chatzoglou E, Kouliaridis V, Kambourakis G, Karopoulos G, Gritzalis S (2023) A hands-on gaze on HTTP/3 security through the lens of HTTP/2 and a public dataset. Comput Secur 125:103051. https://doi.org/10.1016/j.cose.2022.103051
41. Firdaus A, Aldharhani GS, Ismail Z, Ab Razak MF (2022) The summer heat of cryptojacking season: detecting cryptojacking using heatmap and fuzzy. International Conference on Cyber Resilience, ICCR. pp 1–5. https://doi.org/10.1109/ICCR56254.2022.9995891
42. Sarefo S, Dawson M, Banyatsang M (2023) An exploratory analysis of the cybersecurity threat landscape for Botswana. Proc Comput Sci 219(2022):1012–1022. https://doi.org/10.1016/j.procs.2023.01.379
43. Al-kahtani MS, Mehmood Z, Sadad T, Zada I, Ali G, ElAffendi M (2023) Intrusion detection in the internet of things using fusion of GRU-LSTM deep learning model. Intell Autom Soft Comput 37(2):2283
44. Dalal S, Lilhore UK, Faujdar N, Simaiya S, Ayadi M, Almujally NA, Ksibi A (2023) Next-generation cyber attack prediction for IoT systems: leveraging multi-class SVM and optimized CHAID decision tree. J Cloud Comput 12(1):137
45. Enilov M, Mishra T (2023) Gold and the herd of Cryptos: saving oil in blurry times. Energy Econ 122:106690. https://doi.org/10.1016/j.eneco.2023.106690
46. Ha T, Yang H, Hong S (2023) Automated weak signal detection and prediction using keyword network clustering and graph convolutional network. Futures 152:103202. https://doi.org/10.1016/j.futures.2023.103202
47. Lilhore UK, Dalal S, Simaiya S (2024) A cognitive security framework for detecting intrusions in IoT and 5G utilizing deep learning. Comput Secur 136:103560
48. Moreno-Sancho AA, Pastor A, Martinez-Casanueva ID, Gonzalez-Sanchez D, Triana LB (2023) A data infrastructure for heterogeneous telemetry adaptation. Application to Netflow-based cryptojacking detection. Proceedings of the 26th Conference on Innovation in Clouds, Internet and Networks, ICIN. pp 105–112. https://doi.org/10.1109/ICIN56760.2023.10073490

Publisher's Note