




Article

# A Real-Time Hybrid Approach to Combat In-Browser Cryptojacking Malware

Muhammad Haris Khan Abbasi <sup>1</sup>, Subhan Ullah <sup>1</sup> , Tahir Ahmad <sup>2,\*</sup>  and Attaullah Buriro <sup>3</sup> 

<sup>1</sup> Department of Computer Science, National University of Computer and Emerging Sciences (NUCES-FAST), Islamabad 44000, Pakistan

<sup>2</sup> Center for Cybersecurity, Brunno Kessler Foundation, 38123 Trento, Italy

<sup>3</sup> Faculty of Computer Science, Free University Bozen-Bolzano, 39100 Bolzano, Italy

\* Correspondence: ahmad@fbk.eu

**Abstract:** Cryptojacking is a type of computer piracy in which a hacker uses a victim's computer resources, without their knowledge or consent, to mine for cryptocurrency. This is made possible by new memory-based cryptomining techniques and the growth of new web technologies such as WebAssembly, allowing mining to occur within a browser. Most of the research in the field of cryptojacking has focused on detection methods rather than prevention methods. Some of the detection methods proposed in the literature include using static and dynamic features of in-browser cryptojacking malware, along with machine learning algorithms such as Support Vector Machine (SVM), Random Forest (RF), and others. However, these methods can be effective in detecting known cryptojacking malware, but they may not be able to detect new or unknown variants. The existing prevention methods are shown to be effective only against web-assembly (WASM)-based cryptojacking malware and cannot handle mining service-providing scripts that use non-WASM modules. This paper proposes a novel hybrid approach for detecting and preventing web-based cryptojacking. The proposed approach performs the real-time detection and prevention of in-browser cryptojacking malware, using the blacklisting technique and statistical code analysis to identify unique features of non-WASM cryptojacking malware. The experimental results show positive performances in the ease of use and efficiency, with the detection accuracy improved from 97% to 99.6%. Moreover, the time required to prevent already known malware in real time can be decreased by 99.8%.

**Keywords:** in-browser cryptojacking; cryptomining; Monero; cryptojacking detection; cryptojacking prevention; WASM



**Citation:** Khan Abbasi, M.H.; Ullah, S.; Ahmad, T.; Buriro, A. A Real-Time Hybrid Approach to Combat In-Browser Cryptojacking Malware. *Appl. Sci.* **2023**, *13*, 2039. <https://doi.org/10.3390/app13042039>

Academic Editors: Christina Thorpe and Stephen O'Shaughnessy

Received: 16 January 2023  
Revised: 26 January 2023  
Accepted: 2 February 2023  
Published: 4 February 2023



**Copyright:** © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

With the advancement of technology, humans have started finding new secure ways to transfer money online, leading them to the development of blockchain. The first blockchain application, Bitcoin, was created in 2009 by Satoshi Nakamoto [1]. Since then, many other cryptocurrencies have been developed [2], with over 2000 existing today [3]. Cryptocurrency eliminates the need for centralized management and maintains user privacy through decentralized systems and consensus algorithms such as "Proof of Work (PoW)", which requires solving a complex mathematical task to verify transactions. This process is known as cryptomining or simply mining, and the person who solves the puzzle is called a miner [4].

Mining normally requires high computational resources and hence relies on more advanced equipment (requiring expensive hardware) to help earn mining awards [5]. Thus, hackers have employed a technique called cryptojacking [6] to earn mining rewards without spending money on buying such advanced machines. This malware is used to utilize the resources of other people's computers, mobiles, and devices without their consent, allowing the hacker to mine cryptocurrency without the victims' permission, saving the expense of

buying advanced computational machines. Cryptojacking malware attacks became more prevalent after 2017 and have continued to emerge since then [7].

In-browser cryptojacking has recently gained exponential growth and thus has attracted security researchers' attention, from both industry and academia, to find efficient approaches to detect this malware [8]. In addition to the detection work by Darabian et al. [9], some researchers discuss the prevention methodology to mitigate the cryptojacking attack. The study by Yulianto et al. [10] detects the malicious script of cryptojacking by measuring the CPU usage and notifying the user in case of some malicious script in the background of the website. The method presented by Razali and Shariff [11] kills the malicious process after detecting it by comparing it with the blacklist items. Another approach by Bian et al. [12] suspends the execution of the script if detected as malicious. The existing approaches are effective only in detecting in-browser cryptojacking if it has web-assembly (WASM) code instructions [13]. Hence, the malware developed in other languages, such as JavaScript and advanced obfuscation techniques, are not detected. A solution to this problem may include checking the website against a blacklist of already identified web-based cryptojacking malware.

The proposed approach in this paper is a novel hybrid method for in-browser malware detection and prevention. It combines dynamic and static malware analysis techniques by first comparing the URL of a website visited by a user to an existing blacklist of in-browser cryptojacking malware URLs. The website is blocked immediately if the URL is present on the blacklist. Otherwise, it undergoes a static code analysis, and if found to be malicious, it is blocked. If not, a dynamic analysis is performed. The approach includes an interpreting mode to identify malicious code blocks, a detection mode to measure the CPU time of the code block, and a defense mode to suspend it if malicious. The proposed solution allows for flexible suspension intervals for malicious code blocks based on the confidence level. The approach has an overall accuracy of 99.6% on a dataset of 1000 samples and a low-performance overhead, with an increase of only 7% compared to a dynamic analysis alone. It is suitable to be run in real-time systems.

The main contributions of this work are the following:

- The proposal of a novel hybrid approach—combining blacklisting detection (as the 1st), signature-based detection (as the 2nd), and a dynamic approach (as the 3rd) defensive layer.
- The advancement of the state of the art in terms of the detection accuracy (from 97% to 99.6%).
- An extension of the malware protection to non-WASM cryptojacking.

The rest of this paper is organized as follows. Section 2 presents the literature review and the problem statement. A brief description of the proposed approach is provided in Section 3. Section 4 provides the implementation details. The experimental evaluation and discussion of results are provided in Section 5, and Section 6 concludes this paper.

## 2. Literature Review

In this section, we survey the most relevant papers. Tekiner et al. [14] reviewed cryptojacking attacks by analyzing 26,000 cryptojacking samples collected from 45 major attacks created from two unique datasets (VirusTotal and PublicWWW). They classified cryptojacking attacks into two main classes: in-browser and host based. Additionally, they discussed the main sources of such attacks, the mechanism, and the most common platform to launch such malware. They reported that only 15 out of 42 studies focused on detection methods and only three also performed prevention along with detection. A summary of the state-of-the-art approaches categorized as detection, prevention, or both is provided in Table 1.

### 2.1. Static Approaches

Rodriguez et al. [15] presented a static approach based on machine learning SVM techniques. They used the dataset of Alexa 1 Million and selected the features of WASM

signatures of cryptojacking malware for its detection. Similarly, Ruth et al. [16] also optimized the detection of web-based cryptojacking malware. They studied 138 million websites in three top-level domains and analyzed the malicious websites. They developed a fingerprinting methodology that was six times more efficient than the publicly available blacklisting techniques. According to them, the Coinhive is the platform used for illegal cryptomining, used in mining more than 1290 Moneros [17] back in 2018. Both of the above approaches are static and deal with the detection of cryptojacking attacks only. These approaches do not prevent cryptojacking malware after its detection. Kelton et al. [18] also presented an in-browser tool called CoinSpy based on deep learning. It is a cryptojacking classifier for the detection of cryptomining activities within a web page. They combined several alert signals from cryptomining scripts run within a web page. They also analyzed both injected websites and websites in the wild. Their approach was robust compared to the state of the art and achieved a 97% detection accuracy.

## 2.2. Dynamic Approaches

Naseem et al. [19] presented a dynamic web-based cryptojacking detection approach called Minos. They used WASM-based samples collected from the malware dataset of PublicWWW and focused on the image frames of the malicious samples using a Convolutional Neural Network (CNN). Their detection technique achieved an accuracy of 98.97% while using only 4% of the CPU, making this approach very convenient to run on any platform.

Rauchberger et al. [20] presented a framework for detecting and analyzing in-browser cryptojacking malware, named MiningHunter. This framework can detect malware even if implementing an obfuscation technique on it. It was used on Alexa's top 1 million websites and gathered 13 million unique websites containing JavaScript embedded in them. They used a total size of 246 GB of websites. They found that 3178 websites had cryptojacking malware embedded in them. They classified these cryptojacking attacks based on matching and provided in-depth details of their effect on the internet.

Similarly, Munoz et al. [21] developed machine learning techniques and analyzed the NetFlow and IPFIX features. They used the decision tree algorithm and were able to detect cryptocurrency miners. Their detection technique is cost-effective as it requires no payload to analyze a packet. Overall, this dynamic approach achieved a recall value of 90% while using the network traffic studied via the Stratum protocol as a dataset of six cryptocurrencies. Musch et al. [22] presented an in-browser cryptojacking attack and examined its three phases. They also used Alexa's top 1 million websites as a dataset for their study. They concluded that 0.2% of websites are affected by cryptojacking malware. They also studied the insights of cryptojacking malware (e.g., specific code features and how much revenue they may generate) and their countermeasures.

Liu et al. [23] also presented a detection technique for an in-browser cryptojacking malware. They used a dataset of 1159 snapshots of memory of the running browsers that modified the browser kernel code of Chrome. They used a Recurrent Neural Network (RNN) model and analyzed the features of Heap snapshots and the code stack to detect in-browser cryptojacking attacks. They achieved a 95% precision rate and a 93% recall value. Caprolu et al. [24] analyzed the network features and achieved the true positive rate (TPR) of 92%. Similarly, Kelton et al. [18] used CPU memory features from the Alexa 100k dataset by applying a neural network and achieved an accuracy of 97.9%.

Gangwal et al. [25] used the SVM approach and analyzed the features of hardware cache events. They achieved 97% precision in cryptojacking detection. Similarly, Tahir et al. [26] applied Random Forest by analyzing the features of HPC values and achieved a 100% precision rate for detecting cryptojacking attacks. Konoth et al. [27] detected the cryptojacking malware based on CPU cache events. They analyzed the Alexa 1 Million dataset for cryptojacking detection.

Kharraz et al. [28] detected cryptojacking malware and analyzed the JavaScript execution and its compilation time. They used the Alex 600,000 dataset for analysis purposes. Similarly, Hong et al. [29] presented a work that detects cryptojacking malware based

on hash features. They used the Alexa 100,000 dataset and achieved a 100% TPR. Wang et al. [30] presented a technique that dynamically detects malware based on WASM instructions. They used the Alexa 500 dataset for analysis and achieved 98% F1 scores. The above study shows that most of the works from the literature focus only on the detection technique and propose no prevention techniques. Sivaraju [31] also presented a new Cap-Jack method using CapsNet technology and identified illicit Bitcoin cryptomining activity in a browser. His approach can detect malware and fraudulent miners efficiently and heuristically using system behavior or even in a situation where several apps are active simultaneously. Ying et al. [32] presented a novel hardware-based cryptojacking detection technique called CJSpector. They exploited hardware features, e.g., Intel Processor traces and virtualization technology. Then, they used CJSpector based on the library functionality and control flow information received from the processor. They used a Recurrent Neural Network (RNN) to obtain relevant features from the optimized flow control information. Their approach achieved average accuracy, recall, and F1 scores of 98.04%, 96.88%, and 97.92%, respectively, for the detection of cryptomining websites.

### 2.3. Hybrid Approaches

Suarez et al. [33] proposed a detection technique for in-browser cryptojacking malware. They merged host and network-based features and selected 18 unique features from the dataset of 8000 benign websites and the Alexa dataset with 8156 malicious samples, respectively. The Alexa dataset extracted all the malicious samples from different mining service providers. Then, they applied Z-score normalization and autoencoders for dimension reduction and data compression in the preprocessing phase. The training data passed through a deep, dense neural network had an input layer, three hidden layers, and one output layer using the sigmoid activation function. The other layers used the leaky ReLU as the activation function. Their approach has a recall score of 99.25% and has a short training time.

Mani et al. [34] applied a deep neural network, Long Short-Term Memory (LSTM) [35], that used the performance counters of Windows as the data. They trained their model on 1200 samples with the HPU and CPU usage as the key features and achieved a precision value of 96% and a recall score of 97%. Yulianto et al. [10] presented a taint analysis to prevent cryptojacking attacks. Their approach performed two detection checks. They first measured the CPU usage of a website. If the CPU usage was more than 99%, they started the second check. In this check, they compared the scripts run in the background of a website with already blacklisted websites that contain traces of malicious scripts from the AdBlock-no-coin-list. They added the malicious website to the log of malicious scripts. They sent a notification to the user after the second check. Although, this technique deploys double detection, using only 99% as a threshold which is not idle as the cryptojacking malware and getting more and more complex which is capable of throttling the CPU usage. However, only a notification is not sufficient to prevent malware.

**Table 1.** Summary of the state of the art on in-browser cryptojacking malware.

Ref. No	Approach	Dataset	Features	Prevention Method	Result	
[15]	Static	Alexa: 33k	JS API consumption of resources	N/A	TPR: 95.95%	
[16]		Alexa 1 Million	WASM signatures		N/A	
[19]		PublicWWW	Images frames of cryptojacking		Accuracy: 98.97 %	
[20]		Alexa 1 Million	Web socket training		N/A	
[21]		Network Traffic (Stratum)	Network's metadata		Recall: 91 %	
[22]		Alexa 1 Million	CPU usage		N/A	
[23]		Memory of Browser (1160 snapshots)	Stack features, heap snapshots		Precision: 95%	
[24]	N/A	Network features	TPR: 92%			
[18]	Dynamic	Alexa: 100k	CPU memory		N/A	Accuracy: 97%
[25]		N/A	Hardware cache events			Precision: 97.9%
[26]		Manually created dataset (420 instances)	HPC values			Precision: 100%
[27]		Alexa: 1 M	CPU and WASM based			N/A
[28]		Alexa: 600,000	JavaScript execution/compilation			TPR: 97.9%
[29]		Alexa: 100,000	Hashes based			TPR: 100%
[30]		Alexa: 500	WASM	F1 score: 98%		
[33]	Hybrid	Alexa 8000, 8156 samples from Coinhive, etc.	Network traffic, CPU speed, subprocesses	F1 score: 99.25%		
[34]		1200 samples	CPU features and HPC	Precision: 96%		
[10]		PublicWWW	CPU usage, code analysis	Notification	N/A	
[11]		In-browser cryptojacking samples	Blacklist behavior	Kills the process	N/A	
[12]		Alexa 1 M	Code analysis + CPU usage	Suspension of process	FNR 1.83% FPR 0%	

Razali and Shariff [11] proposed two ways for the detection of cryptojacking malware. First, they checked whether the URL of the visited website is present in the blacklisted database. If not, then they analyzed the dynamic behavior and detected potential mining behavior. They ran the application as a web extension with a user-friendly interface. They detected inline cryptojacking and proxy networks. However, they required no special permissions to work, which might be a potential risk. Moreover, their approach lacks to auto-add the detected malicious website to the blacklisted domain.

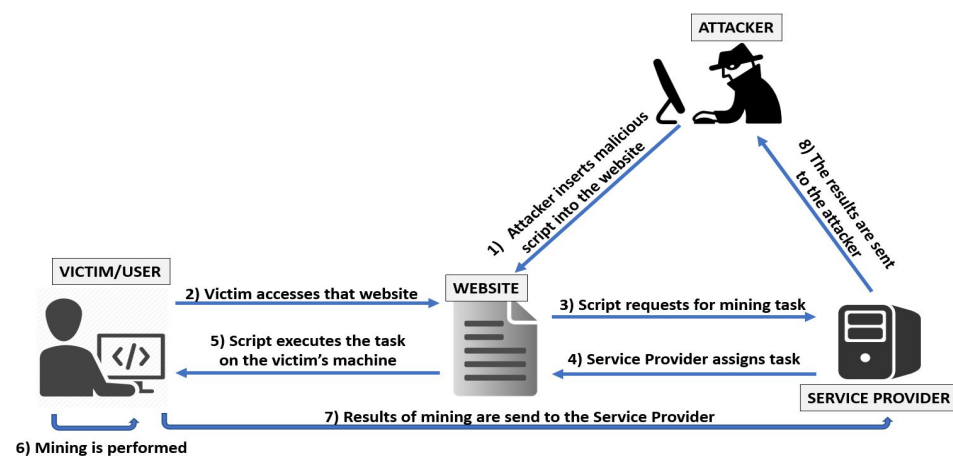
Bian et al. proposed MineThrottle [12], a technique for detecting and preventing cryptojacking malware. Their approach operates in three modes: interpreting, detection, and defense. In the interpreting mode, it extracts mining-related code blocks. In the detection mode, it detects the mining code's speed and compares it with the average speed. In the defense mode, it suspends the detected malicious code for a defined time interval. MineThrottle uses block-level profiling to reduce the complexity and can work in real time, but it only detects and prevents web-assembly (WASM) scripts. Cryptojacking malware



that uses alternative programming languages, such as JavaScript obfuscation techniques, can easily bypass their detection system. For example, Coinhive [36], a mining-service provider frequently used for cryptojacking attacks, can not be detected by MineThrottle.

### 3. Proposed Approach

The complete lifecycle of this malware is shown in Figure 1. The attacker injects the malicious cryptojacking malware script into a website. Either the website is owned by the attacker or a website owned by someone else is compromised so that the payload can be embedded into it.



**Figure 1.** Lifecycle of In-browser Cryptojacking.

The user requests that the malicious website and the website, along with the script, be loaded on the client side, i.e., the user's laptop, desktop, etc. The script gets executed and communicates with a mining service provider. The script will ask for a mining task. The service provider will assign it a mining task to perform cryptomining for some specified cryptocurrencies such as Monero, etc. The script will execute that task on the client side. Cryptomining will start at the user's end using the user's computational resources. The results of the mining process will be sent to the service provider. Finally, these results will be forwarded to the attacker who created the mining script. In this way, the attacker could use the victim's resources without the consent and knowledge of that victim.

Figure 2 shows the architectural diagram of our proposed approach. In addition to the dynamic approach of extracting features such as CPU behavior, this methodology introduces a static malware analysis technique, thus deploying a hybrid defensive mechanism. When the user visits any website, before conducting the dynamic analysis of that website, this approach compares the URL of that website with an existing blacklist containing the URLs of already detected in-browser cryptojacking malware. This technique will check whether that website's URL is in that blacklist. If yes, the website will be blocked immediately, thus saving a lot of computational resources and time as there is no need to conduct the dynamic analysis. On the other hand, if the website's URL is not present, then static code analysis will be conducted, and the URL will be blocked if detected as malicious; else, the dynamic analysis will be performed. The interpreting mode will identify malicious code blocks, the detection mode will measure the CPU time of that code block and compare it against the average CPU time of cryptojacking malware, and finally, the defense mode will suspend it if found to be malicious.

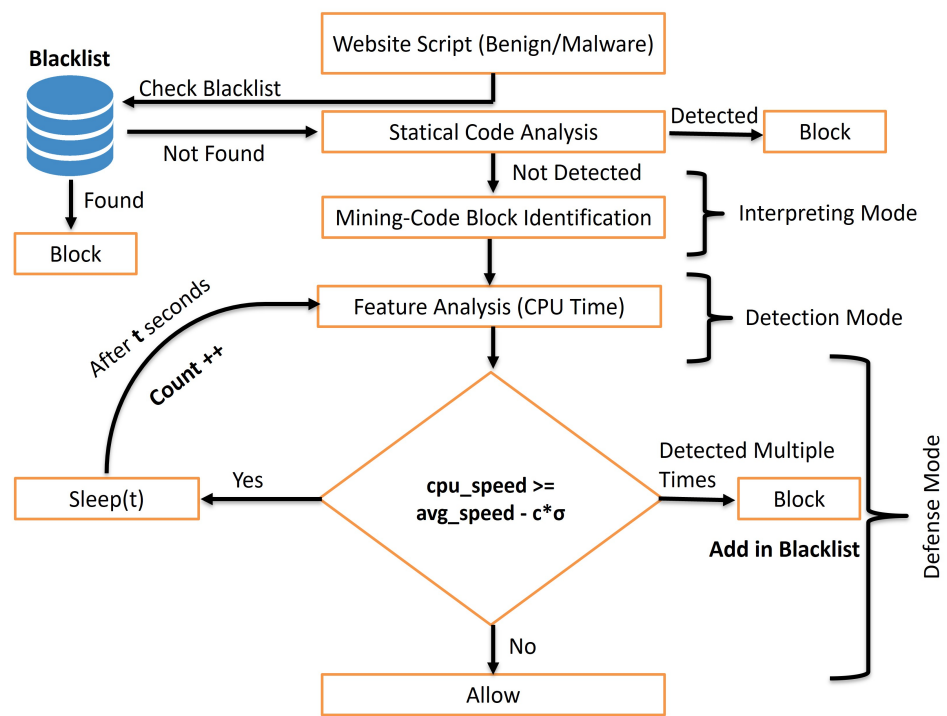


Figure 2. Architectural diagram of proposed approach.

Following is a brief description of the various modules of our proposed approach. **Blacklist:** The blacklist used for detecting cryptojacking malware is used by Tekiner et al. [14] in their research. It consists of the URLs of malicious domains/websites which perform cryptojacking. The corresponding public service provider was mentioned along with their domain name. Moreover, the associated keywords were provided to find the publicly known service provider and their user domains in PublicWWW. It should be noted that it may be possible for some malicious domains to utilize more than one service provider for the cryptojacking attack. Table 2 shows some malicious domains, their service providers, and associated keywords. In general, 14 unique publicly known service providers were used to generate this blacklist. Some famous service providers are service providers such as Coinhive and Cryptoloot. Other service providers are less frequently used but still active. Table 3 shows all the unique service providers and their associated keywords. The dataset presented by Tekiner et al. [14] used the blacklists of previous studies. As some of these blacklists were not regularly updated, the dataset was verified and filtered so that the final dataset consists of 1273 malicious cryptojacking-infected websites. The blacklist contains the URLs of those infected websites. The last time these websites were verified to contain cryptojacking malware was on 24 July 2022.

Table 2. Example of Some Malicious Domains.

S. No	URL	Service Provider	Keyword
0	<a href="http://rugbysearch.co.za">rugbysearch.co.za</a>	jsecoin	<a href="http://load.jsecoin.com">load.jsecoin.com</a>
1	<a href="http://czh72.com">czh72.com</a>	nerohut	<a href="http://nerohut.com/srv">nerohut.com/srv</a>
2	<a href="http://raffey-cassidy.com">raffey-cassidy.com</a>	jsecoin	<a href="http://load.jsecoin.com">load.jsecoin.com</a>
3	<a href="http://school-shop.su">school-shop.su</a>	coinhive	<a href="http://coinhive.min.js">coinhive.min.js</a>
4	<a href="http://247iphone.co.uk">247iphone.co.uk</a>	authedmine	<a href="http://authedmine.min.js">authedmine.min.js</a>

Table 2. Cont.

S. No	URL	Service Provider	Keyword
5	<a href="https://myweedmarket.com">myweedmarket.com</a>	coinhive	coinhive.min.js
6	<a href="https://viralrugby.com">viralrugby.com</a>	coinimp	_client.start
7	<a href="https://mistressalanaaradia.com">mistressalanaaradia.com</a>	coinhive	coinhive.min.js
8	<a href="https://greenheartoc.com">greenheartoc.com</a>	coinhive	coinhive.min.js
9	<a href="https://intellegration.com">intellegration.com</a>	coinimp	_client.start
10	<a href="https://my-shopping-list.de">my-shopping-list.de</a>	authedmine	authedmine.min.js
11	<a href="https://arcadianlandscape.com">arcadianlandscape.com</a>	coinimp	_client.start
12	<a href="https://ifixxxx.com">ifixxxx.com</a>	coinhive	coinhive.min.js
13	<a href="https://sto-avtomix.ru">sto-avtomix.ru</a>	monerise	monerise_payment_address
14	<a href="https://dnd5spells.rpgist.net">dnd5spells.rpgist.net</a>	coinimp	_client.start
15	<a href="https://tabforcancer.com">tabforcancer.com</a>	coinhive	coinhive.min.js
16	<a href="https://onkoliki.com">onkoliki.com</a>	coinhive	coinhive.min.js
17	<a href="https://tildrakizumab.de">tildrakizumab.de</a>	coinhive	coinhive.min.js
18	<a href="https://9-journal.com">9-journal.com</a>	coinhive	coinhive.min.js
19	<a href="https://niftybuzz.com">niftybuzz.com</a>	jsecoin	load.jsecoin.com
20	<a href="https://fhkwindowsanddoor.com">fhkwindowsanddoor.com</a>	browsermine	bmst.pw

Table 3. Unique Public Service Providers.

S. No	Service Providers	Keywords
1	coinimp	_client.start
2	coinhive	coinhive.min.js
3	jsecoin	load.jsecoin.com
4	cryptoloot	CRLT.Anonymous(
5	webminepool	WMP.Anonymous(
6	browsermine	bmst.pw
7	wpmonerominer	wp-monero-miner
8	nerohut	nerohut.com/srv
9	webminerpool	webmr.js
10	coinhave	cdn.minescripts.info
11	deepminer	deepMiner.Anonymous
12	monerise	monerise_payment_address
13	webmine	webmine.cz
14	coinnebula	CoinNebula

**Statical Code Analysis:** In this phase, the source code of the input website is examined to find out if that website communicates with any of the above 14 mentioned unique public mining service providers. If that website turns out to be malicious, it will be blocked from being loaded on the user's machine, else the dynamic analysis is carried out as explained next.

**Mining-Code Blocks Identification:** The code blocks are identified, which contain mining-related instructions. In this way, only selected code blocks are extracted for further analysis in the second mode. The entire code is not used in the detection mode because overhead needs to be excluded. If the entire website code was analyzed, processing time would increase heavily, making this protection mechanism not a real-time defensive technique.



Thus, it is necessary to keep the processing time as limited as possible. The selected code blocks are identified as possible mining scripts if they have some mining-related properties, such as calculating hashes, etc.

**Feature Analysis:** The identified code blocks are measured concerning CPU time. Then, they are compared with the average time calculated from the malicious sample. Different mining service providers are utilized to calculate the baseline value. They are termed malicious if they pass a specified threshold from that average baseline value. The formula used for decision is:

$$cpu\_speed \geq avg\_speed - c * \sigma \quad (1)$$

where *cpu\_speed* is the calculated CPU time of a code block, *avg\_speed* denotes the average CPU time of cryptojacking malware, the last symbol denotes the variance in the malicious cryptojacking samples, and *c* is an arbitrary constant for which different values are used in this research paper. Thus, if the above equation is true in terms of CPU time for a code block, it is termed malicious; otherwise, it is considered non-malicious.

**Defense Phase:** The malicious code blocks are suspended for some time interval and then enter the detection mode again. The `sleep()` function is called in that code block to make it disabled for some time to continue its execution. According to the research paper, the sleep interval used was 10 s. After that interval, that code block faces the detection mode again, and the cycle continues. This way, cryptojacking malware will face too many delays to mine a block and get the reward. Hence, cryptomining becomes inefficient.

#### 4. Implementation Details

In this study, the experimental procedures were performed utilizing a Dell Latitude E5450 laptop equipped with an Intel Core i5-5300U CPU clocked at 2.30 GHz, 8 GB of RAM, and a storage capacity of 400 GB. The experiments were conducted on a Windows 10 Pro operating system. The Chromium Depot Tools [37] were utilized to manage and maintain the Chromium codebase. The toolset, which includes scripts for building, testing, and deploying Chromium and tools for managing dependencies and code reviews, was used to download and build the Chromium source code and execute automated tests on the codebase. The Chromium Depot Tools demonstrated to be a robust and versatile toolset for managing and developing the Chromium project, allowing for streamlined and efficient development and helping to ensure that the operations in the malware safety lab were conducted without limitations on resources.

##### 4.1. Datasets

A dataset consisting of 1000 instances was used in this paper for the experiments. Of these 1000 samples, 30 are malware, and 970 are non-malware. These samples were collected from Alexa [38] and using PublicWWW [39] service. The second most common source is PublicWWW [39], a tool that finds keywords, signatures, etc., in websites' HTML, CSS, or JavaScript code.

##### 4.2. Evaluation Metrics

The selection of evaluation metrics is an important aspect of evaluating the performance of the proposed approach. In this study, we have used four commonly used evaluation metrics, true positive (TP), false negative (FN), false positive (FP), and true negative (TN), to evaluate the performance of the approach. True positive (TP) refers to the correct classification of malware samples, and false negative (FN) refers to the incorrect classification of malware samples. False positive (FP) refers to the incorrect classification of non-malware samples as malware, and true negative (TN) refers to the correct classification of non-malware samples as non-malware. These metrics are commonly used in the literature [37] for evaluating the performance of malware detection systems, as they provide a comprehensive view of the system's performance. They are widely accepted as they are easy to interpret and can be used to calculate other evaluation metrics, such as precision, recall, F1-score, and accuracy.

(1) Accuracy:

$$\frac{TP + TN}{TP + TN + FP + FN} \quad (2)$$

(2) False Positive Rate:

$$\frac{FP}{TN + FP} \quad (3)$$

(3) False Negative Rate:

$$\frac{FN}{TP + FN} \quad (4)$$

## 5. Experimental Evaluation

Using the dataset of 1000 websites having 30 malicious and 970 non-malicious inputs, the proposed approach was followed, and the experimental results are reported in Figure 3. It shows the confusion matrix obtained for the proposed methodology. The proposed approach classifies an input sample as malicious if its URL is found in the blacklist, the static code analysis detects any communication of that website with the public service providers, or Equation (1) is satisfied; otherwise, it is predicted as a non-malicious input. Out of 30 malicious inputs, 26 were detected as TP, while 4 were incorrectly classified as FN, whereas regarding the 970 non-malicious inputs, all the inputs were correctly detected as non-malicious, and no FP was detected.

PREDICTED VALUES	TRUE	26	0
	FALSE	4	970
		TRUE	FALSE
		ACTUAL VALUES	

Figure 3. Confusion Matrix.

By analyzing the above output, it can be seen that an accuracy rate of 99.6% was achieved. At the same time, the false negative rate is 13.3%. Table 4 shows the evaluation metrics with their corresponding values.

Table 4. Performance Evaluation of Proposed Approach.

S. No	Metrics	Values
1	Accuracy	99.6%
2	False Positive Rate	0%
3	False Negative Rate	13.3%

### 5.1. Performance Evaluation

In addition to the dynamic analysis of calculating the CPU usage, the proposed approach has integrated two static analysis techniques: the blacklisting technique and the signature-based detection technique. As we have seen in the previous sections, high accuracy and low false negative rates were achieved. On the other hand, integrating multiple techniques increases the execution time. This section will calculate and compare the performance evaluation of the static analysis with the dynamic analysis.

For experimental purposes, the execution time (in seconds) required for the static and dynamic techniques on five sample websites has been presented. If the in-browser cryptojacking malware is identified at the first defense layer (blacklisting), the second and third defenses will not be run. If the second technique (signature-based detection) identifies the malware, the third defense technique (dynamic analysis) will not be executed. Out of five websites, two contained WASM code. We have chosen the samples so that all three defense techniques will be executed to calculate the performance overhead for the worst case. It can be seen that the blacklisting technique has the lowest execution time. The signature-based detection takes relatively more time than the blacklisting technique, while the dynamic analysis takes the most time. It is because a dynamic analysis requires the malware to be executed for some time. In contrast, a static analysis is performed without executing the malware. Thus, a static analysis requires less time than dynamic approaches. The average increase in the execution of the state-of-the-art work by integrating both static approaches is 7.01%. The maximum increase is 10.7%. Figure 4 shows the execution times in seconds of the static and dynamic analysis, while Table 5 also shows the percentage increase in the execution time. Figure 5 shows how much percentage of the total execution time both the static and dynamic techniques use.

**Table 5.** Performance Evaluation (in seconds).

S. No	Website	WASM/ Non-WASM	Blacklisting Technique	Signature- Based Technique	Total Static Analysis	Dynamic Analysis	Percentage Increase
1	<a href="http://beerthievery.com">http://beerthievery.com</a>	Non-WASM	0.0023	0.04626	0.04884	1.042	4.68%
2	<a href="http://www.rotoglow.com/">http://www.rotoglow.com/</a>	Non-WASM	0.00161	0.10990	0.11167	1.034	10.7%
3	<a href="https://www.dailypaws.com/cats-kittens/cat-names/most-popular-cat-names-2021">https://www.dailypaws.com/cats-kittens/cat-names/most-popular-cat-names-2021</a>	Non-WASM	0.001683	0.06111	0.06384	2.714	2.35%
4	<a href="https://wasm4.org/play/lingword/">https://wasm4.org/play/lingword/</a>	WASM	0.001579	0.08252	0.08528	1.176	7.25%
5	<a href="https://secure.imvu.com/">https://secure.imvu.com/</a>	WASM	0.001496	0.13784	0.13984	1.379	10.1%

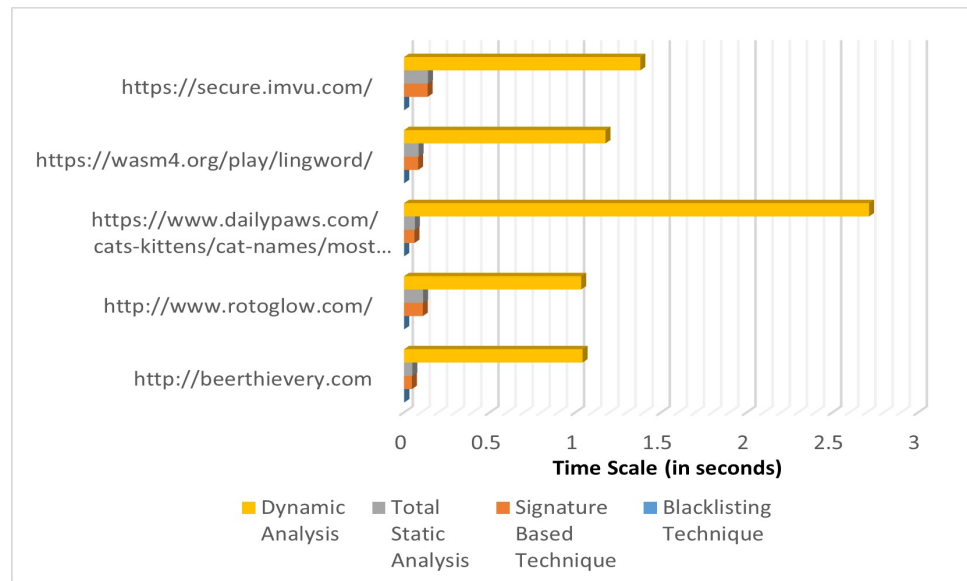


Figure 4. Performance Comparison between Static and Dynamic Approaches.

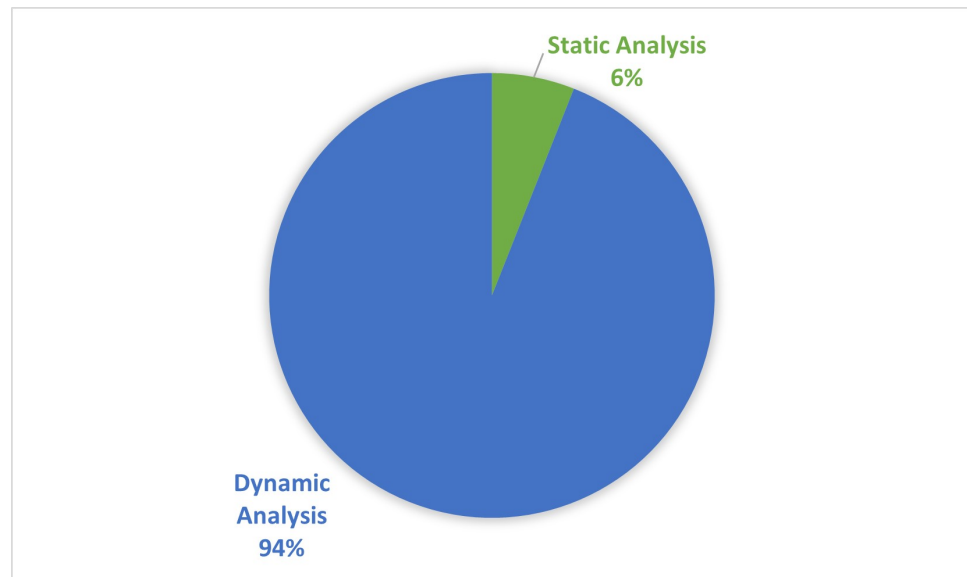


Figure 5. Execution Time in Percentage for Static and Dynamic Approaches.

5.2. Discussion

The proposed methodology is supposed to overcome the limitation of the state-of-the-art approach, which in this is to extend the detection and prevention of in-browser cryptojacking attacks from WASM modules only to both WASM and non-WASM code instructions. By looking at Table 4, it can be analyzed that the proposed solution was able to detect malicious websites better than the previous method presented by Bian et al. [12]. The accuracy has improved from 97.6% to 99.6%. Moreover, the false negative rate has reduced from 60% to 10%. The experiments were also performed on a larger dataset with 1000 instances containing 30 malicious instances to make the results more reliable. The state-of-the-art approach achieved an accuracy of 97.5%, while the proposed approach achieved an accuracy of 99.4%. All these evaluation metrics show that this proposed methodology is more accurate while detecting malicious samples with a maximum performance overhead of 10.7%. This is because the proposed methodology can detect non-WASM service providers’ scripts and detect WASM code blocks only. Thus, the results confirm that the research gap in the state-of-the-art work has been solved efficiently. The experiments show that the proposed approach can efficiently detect and prevent in-browser

cryptojacking attacks in real time. It has achieved an accuracy of 99.6% with a low false negative result of only 13.3%. Moreover, integrating the static and dynamic analyses has increased the performance overhead by only 10.7% for the worst scenario. All these metrics show that the proposed approach is a suitable defense against in-browser cryptojacking attacks. Below is a comparison of this approach with the previous works.

#### Comparison with the State of the Art

As it can be seen from Table 1, many papers do focus on the detection of in-browser cryptojacking attacks only. Work presented by Tahir et al. [26] has achieved a precision rate of 100% but is focused only on detection. Other hybrid approaches such as that of Suarez et al. [33] achieved an F1 score of 99.25% but still lack a prevention procedure. The work presented by Yulianto et al. [10] focuses on preventing cryptojacking attacks but uses only 100 input samples. Moreover, its prevention technique is limited to only notifying the user. Another hybrid preventive technique is that of Razali and Shariff [11]. It successfully blocks the malicious input, but no results were included in their paper, and thus no comparison can be made. The work presented by Bian et al. [12] is the closest work to ours, and thus, we implemented their work and used the same dataset as used for our work. It turned out that for the same dataset, [12] achieved an accuracy of 97%, while our work achieved an accuracy of 99.6%. Thus, our work out-stands the state-of-the-art work in preventing in-browser cryptojacking malware by 2.6%. This is because the proposed methodology can detect non-WASM service providers' scripts along with WASM-based cryptojacking scripts, while [12] can detect WASM code blocks only. Thus, the results confirm that the research gap in the state-of-the-art work has been solved efficiently.

#### 6. Conclusions and Future Work

Hackers have been using in-browser cryptojacking malware for mining crypto on the victim's computer (of course, without consent and knowledge). Research has shown to be limited to its detection by using static and dynamic malware analysis techniques only; however, none of the studies report detection and prevention in the way we propose. In this paper, we have presented a hybrid approach to detect and prevent installing cryptojacking malware. Further, our approach advances the state of the art in terms of accuracy as well: we report an overall accuracy of 99.6%. Unlike the existing approaches, which focus mainly on WASM-based detection, our approach is equally useful in both WASM and non-WASM cryptojacking attacks. The proposed methodology aims to improve the detection and prevention of in-browser cryptojacking attacks by including both WASM and non-WASM code instructions. The results of the experiments show that this approach is more accurate than the previous state-of-the-art approach, with an accuracy of 99.6% and a false negative rate of only 10%. Additionally, the experiments were performed on a larger dataset, making the results more reliable. The proposed approach has also shown a low-performance overhead of 10.7%. Overall, the proposed approach is a suitable defense against in-browser cryptojacking attacks.

The proposed methodology has improved the detection accuracy of the state-of-the-art work. Using multiple behavior parameters in a dynamic analysis, such as the network behavior and CPU time, could potentially improve further the detection accuracy. This can be a future direction for this type of research. As the malicious script has to communicate with the mining service provider, different network behaviors could be studied and included in the analysis to yield better results.

**Author Contributions:** M.H.K.A.: Conceptualization, Data Curation, Software, Writing—Original Draft, Investigation, Validation, and Visualization; S.U.: Methodology, Formal Analysis, Resources, Writing—Original Draft, Validation, and Investigation; T.A.: Writing—Original Draft, Writing—Review and Editing, Funding Acquisition, Validation, and Visualization; A.B.: Methodology, Supervision, Writing—Original Draft, Project Administration, and Visualization. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Available upon request.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Bus. Rev.* **2008**, 21260.
2. Sakas, D.P.; Giannakopoulos, N.T.; Reklitis, D.P.; Dasaklis, T.K. The Effects of Cryptocurrency Trading Websites on Airlines' Advertisement Campaigns. *J. Theor. Appl. Electron. Commer. Res.* **2021**, *16*, 3099–3119. [[CrossRef](#)]
3. Number of Cryptocoins. Available online: <https://coinmarketcap.com/> (accessed on 31 May 2022).
4. Dospinescu, O.; Caramangiu, M.E. The Key Success Factors for an M-Learning Cryptocurrency Application. *Inform. Econ.* **2018**, *22*, 14–24. [[CrossRef](#)]
5. Mestiri, H.; Barraji, I.; Alsir Mohamed, A.; Machhout, M. An efficient AES 32-bit architecture resistant to fault attacks. *Comput. Mater. Contin.* **2022**, *70*, 3667–3683. [[CrossRef](#)]
6. Saad, M.; Khormali, A.; Mohaisen, A. Dine and dash: Static, dynamic, and economic analysis of in-browser cryptojacking. In Proceedings of the APWG Symposium on Electronic Crime Research (eCrime), Pittsburgh, PA, USA, 13–15 November 2019; pp. 1–12.
7. Pastrana, S.; Suarez-Tangil, G. A first look at the crypto-mining malware ecosystem: A decade of unrestricted wealth. In Proceedings of the Internet Measurement Conference, Amsterdam, The Netherlands, 21–23 October 2019; pp. 73–86.
8. Jayasinghe, K.; Poravi, G. A survey of attack instances of cryptojacking targeting cloud infrastructure. In Proceedings of the 2nd Asia Pacific Information Technology Conference, Bali Island, Indonesia, 17–19 January 2020; pp. 100–107.
9. Darabian, H.; Homayounoot, S.; Dehghantanha, A.; Hashemi, S.; Karimipour, H.; Parizi, R.M.; Choo, K.K.R. Detecting cryptomining malware: A deep learning approach for static and dynamic analysis. *J. Grid Comput.* **2020**, *18*, 293–303. [[CrossRef](#)]
10. Yulianto, A.D.; Sukarno, P.; Warrdana, A.A.; Makky, M.A. Mitigation of Cryptojacking Attacks Using Taint Analysis. In Proceedings of the 4th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE), Yogyakarta, Indonesia, 20–21 November 2019; pp. 234–238. [[CrossRef](#)]
11. Razali, M.A.; Mohd Shariff, S. CMBlock: In-Browser Detection and Prevention Cryptojacking Tool Using Blacklist and Behavior-Based Detection Method. In Proceedings of the Advances in Visual Informatics, Bangi, Malaysia, 19–21 November 2019; Badioze Zaman, H., Smeaton, A.F., Shih, T.K., Velastin, S., Terutoshi, T., Mohamad Ali, N., Ahmad, M.N., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 404–414.
12. Bian, W.; Meng, W.; Zhang, M. MineThrottle: Defending against Wasm In-Browser Cryptojacking. In Proceedings of the Web Conference, WWW '20, Taipei, Taiwan, 20–24 April 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 3112–3118. [[CrossRef](#)]
13. WebAssembly-Language. Available online: <https://webassembly.org/> (accessed on 31 May 2022).
14. Tekiner, E.; Acar, A.; Uluagac, A.S.; Kirda, E.; Selcuk, A.A. SoK: Cryptojacking Malware, 2021. In Proceedings of the 2021 IEEE European Symposium on Security and Privacy (EuroS&P), Vienna, Austria, 6–20 September 2021.
15. Rodriguez, J.D.P.; Posegga, J. RAPID: Resource and API-Based Detection Against In-Browser Miners. In Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC '18, San Juan, PR, USA, 3–7 December 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 313–326. [[CrossRef](#)]
16. R uth, J.; Zimmermann, T.; Wolsing, K.; Hohlfeld, O. Digging into Browser-Based Crypto Mining. In Proceedings of the Internet Measurement Conference, IMC '18, Boston, MA, USA, 31 October–2 November 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 70–76. [[CrossRef](#)]
17. Monero. Available online: <https://www.getmonero.org/> (accessed on 31 May 2022).
18. Kelton, C.; Balasubramanian, A.; Raghavendra, R.; Srivatsa, M. Browser-Based Deep Behavioral Detection of Web Cryptomining with CoinSpy. In Proceedings of the 27th Annual Network and Distributed System Security Symposium, NDSS, San Diego, CA, USA, 23–26 February 2020; pp. 23–26.
19. Naseem, F.; Aris, A.; Babun, L.; Tekiner, E.; Uluagac, S. MINOS: A lightweight real-time cryptojacking detection system. In Proceedings of the 28th Annual Network and Distributed System Security Symposium, NDSS, Virtual, 21–25 February 2021.
20. Rauchberger, J.; Schrittwieser, S.; Dam, T.; Luh, R.; Buhov, D.; P tzelberger, G.; Kim, H. The Other Side of the Coin: A Framework for Detecting and Analyzing Web-Based Cryptocurrency Mining Campaigns. In Proceedings of the 13th International Conference on Availability, Reliability and Security, ARES, Hamburg, Germany, 27–30 August 2018; Association for Computing Machinery: New York, NY, USA, 2018. [[CrossRef](#)]
21. i Mu oz, J.Z.; Su arez-Varela, J.; Barlet-Ros, P. Detecting cryptocurrency miners with NetFlow/IPFIX network measurements. In Proceedings of the IEEE International Symposium on Measurements & Networking (M&N), Catania, Italy, 8–10 July 2019; pp. 1–6.



22. Musch, M.; Wressnegger, C.; Johns, M.; Rieck, K. Thieves in the Browser: Web-Based Cryptojacking in the Wild. In Proceedings of the 14th International Conference on Availability, Reliability and Security, ARES '19, Canterbury, UK, 26–29 August 2019; Association for Computing Machinery: New York, NY, USA, 2019. [CrossRef]
23. Liu, J.; Zhao, Z.; Cui, X.; Wang, Z.; Liu, Q. A Novel Approach for Detecting Browser-Based Silent Miner. In Proceedings of the IEEE Third International Conference on Data Science in Cyberspace (DSC), Guangzhou, China, 18–21 June 2018; pp. 490–497.
24. Caprolu, M.; Raponi, S.; Oligeri, G.; Pietro, R.D. Crypto Mining Makes Noise. *arXiv* **2019**, arXiv :1910.09272.
25. Gangwal, A.; Piazzetta, S.G.; Lain, G.; Conti, M. Detecting Covert Cryptomining using HPC. In Proceedings of the Cryptology and Network Security: 19th International Conference, CANS 2020, Vienna, Austria, 14–16 December 2020; pp. 344–364.
26. Tahir, R.; Durrani, S.; Ahmed, F.; Saeed, H.; Zaffar, F.; Ilyas, S. The Browsers Strike Back: Countering Cryptojacking and Parasitic Miners on the Web. In Proceedings of the IEEE INFOCOM 2019—IEEE Conference on Computer Communications, Paris, France, 29 April–2 May 2019; pp. 703–711. [CrossRef]
27. Konoth, R.K.; Vineti, E.; Moonsamy, V.; Lindorfer, M.; Kruegel, C.; Bos, H.; Vigna, G. MineSweeper: An In-Depth Look into Drive-by Cryptocurrency Mining and Its Defense. In Proceedings of the CCS '18: 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15–19 October 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 1714–1730. [CrossRef]
28. Kharraz, A.; Ma, Z.; Murley, P.; Lever, C.; Mason, J.; Miller, A.; Borisov, N.; Antonakakis, M.; Bailey, M. Outguard: Detecting In-Browser Covert Cryptocurrency Mining in the Wild. In Proceedings of the The World Wide Web Conference, WWW '19, San Francisco, CA, USA, 13–17 May 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 840–852. [CrossRef]
29. Hong, G.; Yang, Z.; Yang, S.; Zhang, L.; Nan, Y.; Zhang, Z.; Yang, M.; Zhang, Y.; Qian, Z.; Duan, H. How You Get Shot in the Back: A Systematical Study about Cryptojacking in the Real World. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, CCS '18, Toronto, ON, Canada, 15–19 October 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 1701–1713. [CrossRef]
30. Wang, W.; Ferrell, B.; Xu, X.; Hamlen, K.W.; Hao, S. SEISMIC: SEcure In-lined Script Monitors for Interrupting Cryptojacks. In Proceedings of the Computer Security, Barcelona, Spain, 3–7 September 2018; Lopez, J., Zhou, J., Soriano, M., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 122–142.
31. Sivaraju, S. An Insight into Deep Learning based Cryptojacking Detection Model. *J. Trends Comput. Sci. Smart Technol.* **2022**, *4*, 175–184. [CrossRef]
32. Ying, Q.; Yu, Y.; Tian, D.; Jia, X.; Ma, R.; Hu, C. CJSpector: A Novel Cryptojacking Detection Method Using Hardware Trace and Deep Learning. *J. Grid Comput.* **2022**, *20*, 31. [CrossRef]
33. Hernandez-Suarez, A.; Sanchez-Perez, G.; Toscano-Medina, L.K.; Olivares-Mercado, J.; Portillo-Portilo, J.; Avalos, J.G.; García Vilalba, L.J. Detecting Cryptojacking Web Threats: An Approach with Autoencoders and Deep Dense Neural Networks. *Appl. Sci.* **2022**, *12*, 3234. [CrossRef]
34. Mani, G.; Pasumarti, V.; Bhargava, B.; Vora, F.; MacDonald, J.; King, J.; Kobes, J. DeCrypto Pro: Deep Learning Based Cryptomining Malware Detection Using Performance Counters. In Proceedings of the IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS), Washington, DC, USA, 17–21 August 2020; IEEE Computer Society: Los Alamitos, CA, USA, 2020; pp. 109–118. [CrossRef]
35. Long Short-Term Memory. Available online: [https://en.wikipedia.org/wiki/Long\\_short-term\\_memory](https://en.wikipedia.org/wiki/Long_short-term_memory) (accessed on 31 May 2022).
36. Coinhive. Available online: <https://krebsonsecurity.com/2018/03/who-and-what-is-coinhive/> (accessed on 31 May 2022).
37. Metrics. Available online: <https://onlineconfusionmatrix.com/> (accessed on 31 May 2022).
38. Alexa. Available online: <https://www.alexa.com/> (accessed on 31 May 2022).
39. PublicWWW. Available online: <https://publicwww.com/> (accessed on 31 May 2022).

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.