MineDetector: JavaScript Browser-side Cryptomining Detection using Static Methods

Peiran Wang*, Yuqiang Sun*, Cheng Huang*†§, Yutong Du*, Genpei Liang*, Gang Long ‡
*School of Cyber Science and Engineering, Sichuan University, Chengdu, China
†Guangxi Key Laboratory of Cryptography and Information Security, Guilin, China
‡Faculty of Computing, Harbin Institute of Technology, Harbin, China
§Corresponding author, Email: opcodesec@gmail.com

***Abstract*—Because of the rise of the Monroe coin, many JavaScript files with embedded malicious code are used to mine cryptocurrency using the computing power of the browser client. This kind of script does not have any obvious behaviors when it is running, so it is difficult for common users to witness them easily. This feature could lead the browser side cryptocurrency mining abused without the user's permission. Traditional browser security strategies focus on information disclosure and malicious code execution, but not suitable for such scenes. Thus, we present a novel detection method named MineDetector using a machine learning algorithm and static features for automatically detecting browser-side cryptojacking scripts on the websites. MineDetector extracts five static feature groups available from the abstract syntax tree and text of codes and combines them using the machine learning method to build a powerful cryptojacking classifier. In the real experiment, MineDetector achieves the accuracy of 99.41% and the recall of 93.55% and has better performance in time comparing with present dynamic methods. We also made our work user-friendly by developing a browser extension that is click-to-run on the Chrome browser.**

I. Introduction

Situation. For a long time, online advertising has been an effective way for website owners to transfer their popularity to profit, rather than a mechanism that was unbearable for most visitors. However, visitors are still dissatisfied with the existence of web advertising because it worsens their reading experience. The implement of online advertising brings a significant descent to the websites' popularity and the income of websites. As a revenue-generating mechanism to take place of advertising, browser-side cryptocurrency [1] mining gradually becomes the best choice for website owners. In the past, cryptocurrencies, such as bitcoin, were unfriendly to miners with the only CPU. At the beginning of cryptocurrency mining, CPU miners take the majority part of cryptocurrency miners. Yet with the first Bitcoin block mined on a GPU by a miner named ArtForz, the era of CPU miners died. Programmable FPGA chips brought custom-built circuits specially designed for cryptocurrency mining. They have spawned a lot of mining-oriented hardware. Then by mid-2012, some companies started selling ASICs. Those ASICs are specially designed for cryptomining. From then on, mining-oriented ASICs have held the throne of cryptocurrencies for several years until Monero(XMR). The base of the Monero, CryptoNight algorithm is I/O intensive: it stores a set of bytes and requires frequent reads and writes. CryptoNight requires 2MB, which is slightly smaller than the L3 cache of modern processers, to store temporary results. Because of the low latency of the L3 cache, this algorithm performs better on CPUs than FPGAs or ASICs [2].

Problem Description. Several browser-side cryptocurrency mining script families based on JavaScript, such as CoinHive, Crypto-Loot, and JSECoin, have become efficient profit tools for website owners. To use Coinhive [3], website owners just need to copy several lines of code that launch Coinhive and paste those lines on their websites. In the process, no complicated operations are taken. Also, users may not be aware of the existence of mining scripts because the scripts are not shown in the user interface as online advertisements do. Due to the simplicity and the invisibility of browser-side cryptocurrency mining, it is seriously abused. Many websites stealthily run cryptocurrency mining scripts without the consent of users. Some researchers call this phenomenon cryptojacking. Users entering this kind of website will unconsciously offer their own computers' computing power to assist the mining. What is worse, cryptojacking scripts will just do extremely complicated computation which consumes a lot of computing powers. But no suspicious behavior like reading system data, stealing cookies will be conducted in the meantime. Thus, most web browsers do not have special protection mechanism against it.

Nowadays there are some browser-side mining detection frameworks [4], such as OutGuard and CoinSpy. However, most of these methods are based on dynamic feature detection. Those dynamic features include the pattern of stack calls and CPU usage. However, the existing methods have shortcomings in time cost and misreport.

Our Framework. We proposed a framework called MineDetector to automatically detect cryptojacking in websites. The design of MineDetector is based on several static features found within JavaScript mining families. We systematically analyzed the existing JavaScript cryptocurrencies mining families' codes and extract features based on their keywords, API calls, and the structure of their abstract syntax tree. Based on these features, we built MineDetector using a machine learning algorithm.

Contributions. With this paper we make the following contributions:

- Designed and implemented MineDetector based on static analysis, an approach for automatically crawling web

pages, extracting JavaScript codes, and judging if these pages are mining web pages.
- Made MineDetector user-friendly by providing a one-click Chrome extension. MineDetector browser extension will collect scripts from the website been visited and send the scripts to the MineDetector server for analysis.
- Investigated today's JavaScript cryptocurrencies mining and compared our detection method with them.

II. Related Work

A. Cryptojacking Harm

Existing research work has been done to discuss the current situation of browser-side cryptojacking: how much one website can earn with cryptojacking, how many websites are involved in cryptojacking [5][6].

Most cryptojacking scripts discovered were configured to utilize the website users' 25% computing power. The threshold of 25% was to prevent users from discovering the presence of the scripts. Though in the early days, some scripts using 100% computing power of users' CPU are reported, which could be easily detected. Cryptocurrency scripts will consume users' computation power and occupy the processor at a high level. This will seriously influence the daily use of the users' computers. Also, some hardware and economic losses may be brought by the abuse of cryptojacking as well [7]. Former reports have reported that there had been an 8500% increase in cryptojacking online websites [8].

B. Cryptojacking Detection

As for detection towards cryptojacking, some ancient methods which conduct detection by existing public blacklist [5] have been done. For instance, Google developed the MinerBlock Chrome extension to prevent cryptojacking in the Google browser. Yet these methods are too simple to be bypassed and crude: blacklists require daily maintenance or they'll quickly lose track of newly rising cryptojacking websites; pattern matching is easy to bypass with some easy confusion of JavaScript cryptocurrencies codes.

The further study looked at the unique behavior of those cryptojacking scripts. Behavioral features have been used to provide great malware detection methods across a variety of fields [9][10]. Konoth developed a framework called Minesweeper. It could monitor the pattern of reading and writing operations of CPU L1 cache as a special behavior signal for cryptojacking. This is because the memory-hard PoW(proof of work) algorithm could bring special known frequent access rates to these caches [11]. While this method requires detection scripts to monitor the low-level behaviors of computer storage and consumes a lot of time when executing massive detection.

CMTracker [12] and Outguard [13] first implemented behavioral analysis as the basis of their cryptojacking detection. CMTracker systematically proposed the detailed behavior features of JavaScript cryptojacking scripts including several Web worker processes, the dynamic behavior of Web Assembly [14], the number of connections established by WebSocket, and so on. Based on these features, CMTracker built its machine learning model using the support vector machine algorithm. Outguard takes the pattern of stack calls and the call of a hash algorithm for detection. And OutGuard developed a scoring mechanism over the two features to judge if a web page is a cryptocurrencies page. But the price of time cost is rather high for they need to wait 5 seconds in analyzing every web page. The time cost can be accepted under the circumstance that a single user is not sensitive about the response time when visiting websites. But in a situation that requires massive script analysis, the time cost of them can not be accepted.

CoinSpy [12] focused on how cryptojacking scripts influenced the resources of a computer. The emphasis of its study lies in the fact that cryptojacking required a large scale of CPU, storage, and network resources devoted to hash computing. CoinSpy collected samples of CPU, memory, and network behaviors and represented them as a time series. The time series was used to learn both individual and correlated patterns that point to cryptojacking, all from within the browser [15].

C. Web Malware Detection Methods

In the past, Web security defenses mainly focus on detecting malware with some static signatures such as the number of IP addresses or domains existing in known blacklists, or DOM read operations and write operations [16][17]. Though these simple methods have been proved to be quite efficient, they only consume little computing power. Current research has also proved that they are also insensitive to bypass techniques such as code obfuscation [18].

In contrast to the static methods we mentioned above, dynamic defenses focus on the behavioral features of web malware. B. L. Paruj Ratanaworabhan [19] used access patterns to detect code injection attacks. TCP traffic patterns were also applied by F. Tegeler [20] to detect botnets in the network. These defense methods have applied various behavioral patterns in response to evasive techniques, but consume more computing power at the same time.

D. Abstract Syntax Tree

Abstract syntax tree(AST) is an abstract pattern of representation. It represents the syntax structure of programming language in a tree-like format. Every node in the tree represents a structure of the source codes. It is usually used as a medium between source codes and executable files. When being compiled into executable files, source codes are first built into a parse tree which will be turned into an abstract syntax tree next. Meanwhile, AST has also brought some opportunities for researchers. Jianglang Feng [21] developed an effective plagiarism detection algorithm based on the AST which can detect plagiarism in case of changing the names of variables in the codes or even reordering the sequences of the codes and so on. Furthermore, Jingling Zhao [22] proposed a more effective plagiarism detection technique based on AST by computing the hash values of every node of the

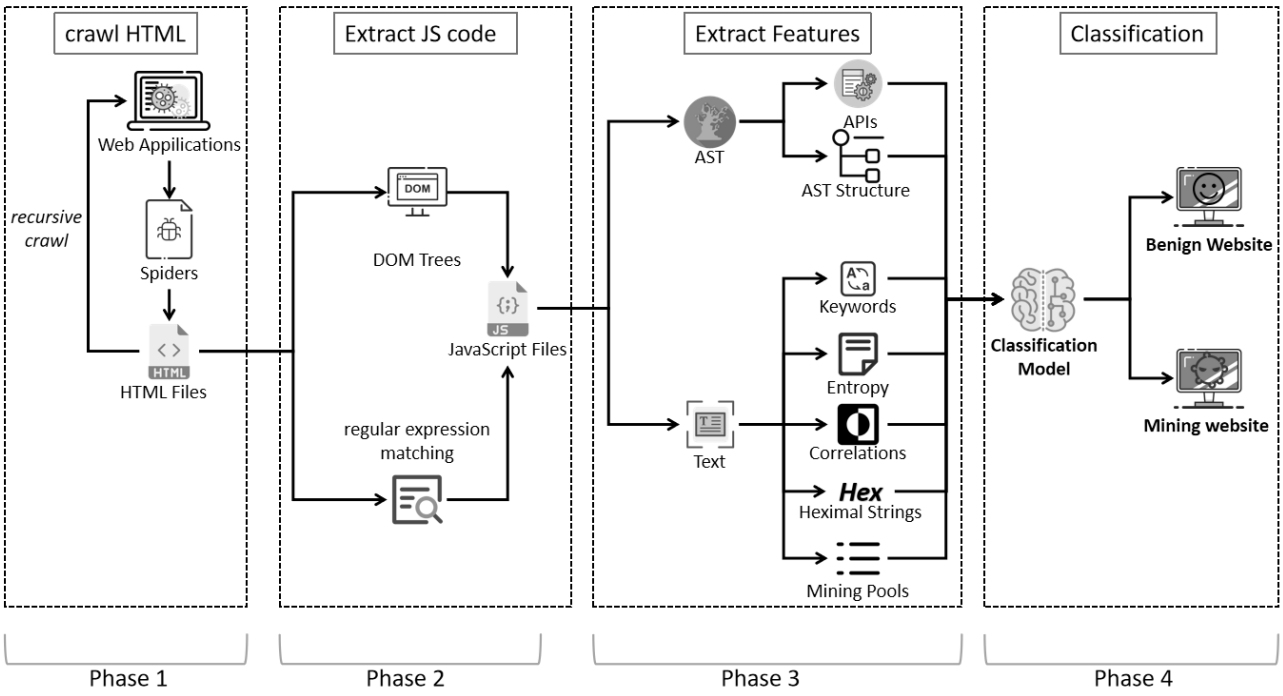

Fig. 1. The detail structure of proposed system named MineDetector

abstract syntax tree of the program and comparing them with plagiarized programs.

Besides plagiarism detection, AST can be used in more research fields. Tuan-Hung Pham [23] proposed an effective approach to study whether the implementation of programs is in corresponds with their security policy based on the abstract syntax tree of their source codes. Xin Wang [24] developed a reverse engineering technique using an abstract syntax tree. Xin Wang proposed a framework to build the corresponding UML model elements by traversing every node in an abstract syntax tree.

Current research based on AST has revealed the truth that abstract syntax tree is a great instrument for us to capture the semantics of source codes.

III. Framework

In this section, we will introduce the framework of MineDetector. This paper proposes to detect cryptojacking according to static features extracted from JavaScript source codes. MineDetector is composed of several components, as illustrated in Figure 1. First, the crawler will get the target web pages and store them in local storage from websites in a recursive method. The recursive crawling method can avoid missing redirected HTML pages. In the second phase, using the DOM tree and regular expression matching, MineDetector extracts JavaScript codes directly from web HTML pages crawled last time. Then MineDetector transfers JavaScript codes into abstract syntax tree format to extract AST structure features. Those features are used to measure if the structure of the target script is like the structure of mining scripts. Application programming interface calls information is also extracted via abstract syntax tree. And scripts are transformed into text format to extract keywords information, the existence of mining pool address, the entropy of information, the similarity of code text, and the portion of hexadecimal strings in the text as well. At last with the features extracted MineDetector uses a random forest algorithm to build our model.

A. Data Collection and Preprocessing

In the process of collecting data, we collect different families of JavaScript cryptojacking codes including Coinhive, PPoi, webmine, JSECoin, which differ in API calls and AST structure. With the closure of the Coinhive service, it is impossible to directly access Coinhive's official website to get its source codes. However, we can find the history version of mining scripts' official website on the Wayback Machine (web.archieve.org) [25]. Also, we can collect JavaScript samples of the websites used to conduct cryptojacking as well. Besides, we also collect a few samples from VirusShare [26], a website to share newly discovered viruses and malicious codes.

The crawler gets the target web pages by crawling the original HTML pages from the websites and stores the web pages in local storage. Then MineDetector extracts JavaScript code from the web pages stored in local storage. The extraction is performed by extracting the content in script labels in DOM trees and uses regular expression matching.

After collecting enough samples, we need to preprocess the scripts for feature extraction. It is not a good idea to just extract raw HTML pages for HTML may greatly interrupt our feature

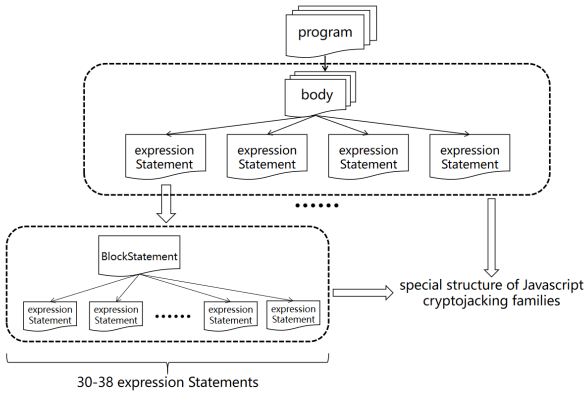

Fig. 2. The AST structure of JavaScript cryptojacking families

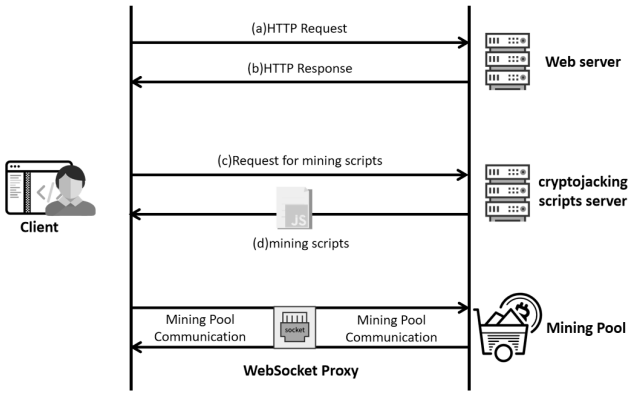

Fig. 3. Communication steps for Browser-side Cryptomining (eg. Coinhive)

extraction. Also, during our sample collection, we found that few websites had used redirection to avoid direct detection over initial HTML pages. So, we developed a recursive crawler to crawl web HTML pages from online websites and extract JavaScript codes from them. Then we converted JavaScript codes into abstract syntax trees and text preparing for features extraction afterward.

B. AST Structure Feature Extraction

During our analysis of the AST structure of various JavaScript cryptojacking scripts, we found some special structures of these scripts as shown in Figure 2.

Among all the JavaScript cryptojacking families we had found, Coinhive, PPoi, CryptoWebMiner, DeepMiner, and several other families shared the same AST structure which made us easier to detect them. What is more, because of our searching, all the websites we had found which used to be cryptojacking websites conducted cryptojacking using existing open-source JavaScript cryptojacking families. And there was not a single one using the scripts developed by themselves. The reason might be quite simple. Most website owners could not develop their cryptojacking scripts. As a result, it was much simpler to just use the ready-made scripts developed by several organizations.

When extracting AST structure features from the script, MineDetector will first build the abstract syntax tree of the scripts. Then MineDetector will traverse the whole abstract syntax tree to check the degree of similarity between the AST structure of the target script and cryptojacking families scripts based on a depth-first traverse. MineDetector will judge mainly on the number of expression statements based on the unique number of cryptojacking families' expression statements. The number should range from 30 to 38 for it is the number of the object's functions in the cryptojacking families.

C. API Call Feature Extraction

Coinhive's scripts work as the process shown in Figure 3 above: Firstly user's host sends an HTTP Request to the Web server to get the web page. Then the Web server sends HTTP Response with a web page as a reply to the request.

TABLE I
THE API CALL FEATURES EXPLANATION

API	Illustration
navigator	*navigator* function used to navigate on the web pages.
hasWASMSupport	*hasWASMSupport* function is used to check whether the webassembly can be used or not.
anonymous	*anonymous* function is used to receive the address of mining pool
prototype	*prototype* is the attribute name of object *coinhive* in mining scripts.
start	*start* function is used to start the mining process of mining scripts
CRLT	*CRLT* is used as the name of an object in the mining scripts
miner	*miner* frequently appears in the scripts as the name of an object

After that, the short link of Coinhive hidden in the web page replied from the Web server will ask for the user's host to request for mining scripts stored in the Coinhive scripts server. After the requirement is satisfied, the mining scripts of Coinhive establishes a connection with the mining pool using WebSocket and start hashing computing [27].

Observing the process, it is not complicated to find the key procedure that mining scripts must do. First, to establish communication with the mining pool, mining scripts must use WebSocket. Then, the hashing computing task requires mining scripts to use WebAssembly and hashing algorithms.

MineDetector will traverse the abstract syntax tree of the target script and extract all the application programming interfaces into a list from the tree. Then MineDetector will traverse the list and identify the existence of target application programming interfaces shown in Table I.

D. Keywords Feature Extraction

Besides the two groups of features we define above, keywords features are also remarkable features to help us detect cryptojacking scripts. Some keywords such as *crypto*, *throttle*, *coinhive*, *miner* are usually used by mining scripts for they are usually the attributes name or objects taken by the scripts.

TABLE II
THE KEYWORDS FEATURE EXPLANATION.

Keywords	Illustration
coinhive	*coinhive* is the object name used in Coinhive scripts.
throttle	*throttle* is the parameter coinhive uses to set the unilization ratio of the CPU.
prototype	*prototype* is the attribute name of coinhive object in the mining scripts.
miner	*miner* frequently appears in the scripts as the name of an object.
hash	coinhive needs to run hash algorithm to get cryptocurrencies thus making *hash* an usual string appearing in the scripts.
cryptonight	*cryptonight* is the hash algorithm applied by many cryptocurrencies.
anonymous	*anonymous* is the function name which is used to receive the address of mining pool.
cloudcoins	some scripts will use *cloudcoins* as the substitute of coinhive.

It is quite hard for attackers to change the original code, for the scripts are too complicated. The keywords are presented in Table II.

E. Mining Pool Features Extraction

When deploying mining scripts on the website, the attacker should set the address of the mining pools for the mining process. Most mining pool addresses were collected by us, and the existence of the mining pool addresses was also treated as a feature. We used regular expressions to find the existing mining pool addresses in the script. This was conducted by comparing addresses in the blacklist of mining pool addresses with addresses in the script.

F. Other Features Extraction

During our observation among the script samples collected, we've found that the users of cryptojacking scripts lacked the common sense to hide the cryptojacking scripts from being discovered by the users. As a result, we can calculate coincidence by calculating the equation below. In the equation below, $Count$ is the number of characters in the text and $count_i$ represents the number of ith characters of ASCII characters in the text.

$$correlation = \frac{\sum_i^{256} count_i \times (count_i - 1)}{Count \times (Count - 1)} \quad (1)$$

Meanwhile, some users of cryptojacking scripts take advantage of the encoding algorithm to encode the cryptojacking scripts. As a response, we can use the portion of heximal strings in the text as a feature. When the cryptojacking scripts are encoded, the portion of heximal strings will rise for characters in the script text are transferred into heximal strings.

What is more, entropy can be a strong feature as well. With the rise of the portion of heximal strings, the entropy of the scripts' text will rise as well.

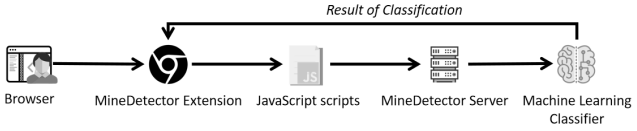

Fig. 4. The detail process of MineDetector chrome extension

G. Browser Extension

We deploy the trained model of MineDetector as a browser extension [28] to make MineDetector user-friendly. It just requires users to click once when visiting websites. The extension in the browser will collect the scripts from websites via browser extension APIs. And then the extension will send the JavaScript code it collects from the user's browser to the server. The classification model in the server will detect the potential cryptomining code by extracting features and use a machine learning model to predict. The MineDetector model is deployed in the server. Without deploying it within the user's browser, it can save the user's computing power. The structure of the MineDetector Chrome extension is shown in Figure 4.

IV. EXPERIMENT AND EVALUATION

In this section, we will introduce the detailed composition of the dataset, the experiment details, the experiment results, and the comparison with other cryptojacking detection methods.

A. Dataset

There are two datasets for our experiments. One is the JavaScript cryptojacking dataset storing 130 browser-side JavaScript cryptomining scripts we collected from virusshare.com and Wayback machine [25]. And the other one is the normal JavaScript dataset of 2,668 scripts which we collected from BootCDN.

B. Machine Learning Model

The theory of random forest was first proposed by Tianqin He in 1995 [29]. And Leo Breiman [30] officially proposed the algorithm of random forest in his paper which described a method to combine Breiman's bagging idea with a random selection of features proposed by Ho to construct the forest of decision trees. Decision tree [31] is a popular machine learning method in a binary classification task. Decision trees that grow very deeply to learn the patterns of the training set may overfit, and become unable to fit the data in reality. Under the circumstance of the drawback of decision trees, random forests are proposed as a good method to average multiple deep decision trees. The different decision trees of the random forests will be trained on different parts of the training set provided by researchers. The small increasing expense of the bias and interpretability that comes with random forests can be ignored when comparing with the extraordinary improvement of the performance of the final model. We can regard random forests as a set of decision trees. Random forests will take the teamwork of all the decision trees, therefore improving the performance of the whole model.

C. Experiment design

We compare the Random Forests model with several other machine learning algorithms including SVM(Supported Vector Machine), Naive Bayes, and Logistic Regression which are commonly used in the binary classification tasks, and find the one with the best performance. All the experiments were performed on a PC machine with an i7-8750H processor and 16GB of memory. All the machine learning models are implemented by scikit-learn. Before using the raw data to build the model, we did a standardization to make all the data appropriate for the machine learning models. We used some python libraries including esprima(to formulate abstract syntax tree), re(to use regular expressions to extract features), pandas(to build features matrix), scikit-learn(to formulate machine learning model), and seaborn(to plot graph).

Because the dataset used for the experiment is unbalanced(only 5% of the whole samples are cryptojacking scripts, and the rest are normal samples), we choose Accuracy, Recall, Precision, F1-Score and the Receiver Operating Characteristics Curves (ROC) and Area-Under-Curve (AUC) measure to evaluate the proposed method and to measure the performance of each machine learning algorithms. Besides, we can visualize the relation between TPR and FPR of a classifier. These indicators can be expressed as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

$$Recall = \frac{TP}{TP + TN} \tag{3}$$

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{5}$$

$$AUC = \frac{\sum_{iepositiveclass} rank_i - \frac{M \times (1+M)}{2}}{M \times N} \tag{6}$$

Where M is the number of cryptojacking samples and N is the number of normal samples. The score indicates the probability that each test sample belongs to a cryptojacking sample, while rank is a positive sample set sorted in descending order based on score. The higher these indicators are, the better the algorithm performs on the problem.

D. Experiment Result

We explored the best algorithm that suited the circumstance by comparing the results of different algorithms used for binary classification tasks which are Logistic Regression, Naive Bayes, Random Forests, and SVM. And the results of the experiments are shown below:

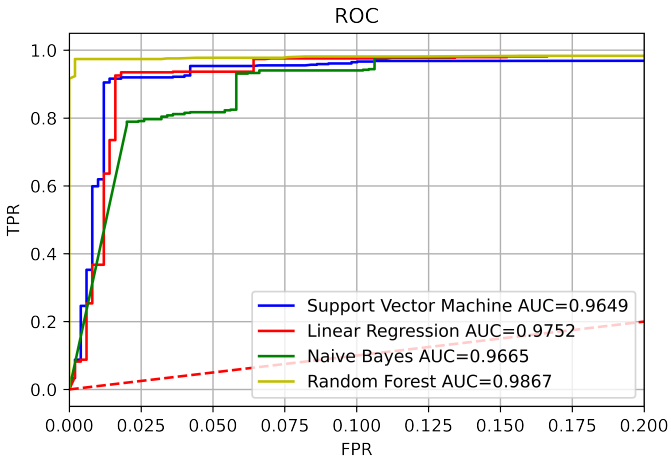

Fig. 5. Detail ROC of proposed algorithms

Among all the machine learning algorithms we tried, the random forest is the one which performs best in accuracy, recall, precision, and F_1-score, therefore making random forest the most suitable algorithm for MineDetector.

TABLE III
THE RESULTS OF DIFFERENT ALGORITHMS WITH MINEDETECTOR

Classification Algorithm	Evaluation Criteria			
	Accuracy	*Recall*	*Precision*	*F_1-score*
Logistic Regression	99.27%	93.47%	96.43%	91.53%
Naive Bayes	98.63%	85.41%	95.65%	81.48%
Supported Vector Machine	99.12%	91.86%	96.30%	89.66%
Random Forest	**99.41%**	**93.55%**	**99.57%**	**93.10%**

E. Comparisons with Previous Methods

Besides our method, there are existing browser-side cryptojacking detection methods [12][13] done by former researchers. Existing dynamic methods do have the advantages of high accuracy and difficulty to bypass but also have some drawbacks in efficiency. Previous research has shown that it'll take about 3 seconds for CMTracker [12] to detect a cryptocurrency script in a website for they are supposed to collect the dynamic features during running time. While it takes MineDetector only 22 microseconds to accomplish the same work.

V. DISCUSSION AND LIMITATIONS

In this paper, we proposed a JavaScript cryptojacking scripts detection method based on several static features extracted from the scripts. We developed our detection model called MineDetector using the method we mentioned above.

MineDetector can judge whether the test script is a cryptojacking script or not in a few microseconds. This is much faster than previous dynamic methods. But it might also be limited by several aspects.

The AST structure features detection can be easily bypassed if attackers know about the detection methods we used. They can change the AST structure of the cryptojacking provided

by the opening sources organizations to bypass the detection. But the API of mining scripts used can not be changed, which suggests that our other features are still valid.

VI. Conclusion

In this paper, we investigate the current situation of cryptojacking in a browser. We assessed the efficiency of several current cryptojacking detection methods which used dynamic methods or static methods to detect cryptojacking in a browser. Based on our research, we proposed a static method based on AST structure and API call information and developed our model, MineDetector. MineDetector can achieve an accuracy of 99.41% and a recall of 93.55% on the datasets of 130 malicious samples and 2,668 benign samples. It can handle the quick response to the cryptojacking within several microseconds and make up for the shortage of existing dynamic detection methods. Furthermore, we develop a Chrome extension using MineDetector to detect the mining scripts in the website [28]. We are planning to import deep learning to fix the problem in the future.

VII. Acknowledgment

This document is the results of the research project funded by the National Natural Science Foundation of China (No.61902265), and Sichuan Science and Technology Program (No.2020YFG0047, No.2020YFG0076) and Guangxi Key Laboratory of Cryptography and Information Security (No.GCIS201921).

References

[1] Marius Musch, Christian Wressnegger, Martin Johns, and Konrad Rieck. Web-based cryptojacking in the wild. *arXiv preprint arXiv:1808.09474*, 2018.

[2] Fergal Reid and Martin Harrigan. An analysis of anonymity in the bitcoin system. In *Security and privacy in social networks*, pages 197–223. Springer, 2013.

[3] Brian Krebs. Who and what is coinhive. *Website https://krebsonsecurity.com/2018/03/who-and-what-is-coinhive*, 2018.

[4] Victor Marchetto et al. An investigation of cryptojacking: Malware analysis and defense strategies. *Journal of Strategic Innovation and Sustainability*, 14(1):66–80, 2019.

[5] Shayan Eskandari, Andreas Leoutsarakos, Troy Mursch, and Jeremy Clark. A first look at browser-based cryptojacking. In *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 58–66. IEEE, 2018.

[6] Jan Rüth, Torsten Zimmermann, Konrad Wolsing, and Oliver Hohlfeld. Digging into browser-based crypto mining. In *Proceedings of the Internet Measurement Conference 2018*, pages 70–76, 2018.

[7] Muhammad Saad, Aminollah Khormali, and Aziz Mohaisen. Dine and dash: Static, dynamic, and economic analysis of in-browser cryptojacking. In *2019 APWG Symposium on Electronic Crime Research (eCrime)*, pages 1–12. IEEE, 2019.

[8] Troy Mursch. Cryptojacking malware coinhive found on 30,000+ websites, 2017.

[9] Michael Bailey, Jon Oberheide, Jon Andersen, Z Morley Mao, Farnam Jahanian, and Jose Nazario. Automated classification and analysis of internet malware. In *International Workshop on Recent Advances in Intrusion Detection*, pages 178–197. Springer, 2007.

[10] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore Stolfo. On the feasibility of online malware detection with performance counters. *ACM SIGARCH Computer Architecture News*, 41(3):559–570, 2013.

[11] Radhesh Krishnan Konoth, Emanuele Vineti, Veelasha Moonsamy, Martina Lindorfer, Christopher Kruegel, Herbert Bos, and Giovanni Vigna. Minesweeper: An in-depth look into drive-by cryptocurrency mining and its defense. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1714–1730, 2018.

[12] Geng Hong, Zhemin Yang, Sen Yang, Lei Zhang, Yuhong Nan, Zhibo Zhang, Min Yang, Yuan Zhang, Zhiyun Qian, and Haixin Duan. How you get shot in the back: A systematical study about cryptojacking in the real world. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1701–1713, 2018.

[13] Amin Kharraz, Zane Ma, Paul Murley, Charles Lever, Joshua Mason, Andrew Miller, Nikita Borisov, Manos Antonakakis, and Michael Bailey. Outguard: Detecting in-browser covert cryptocurrency mining in the wild. In *The World Wide Web Conference*, pages 840–852, 2019.

[14] R Neumann and A Toro. In-browser mining: Coinhive and webassembly. *Forcepoint Security Labs*, 2018.

[15] Conor Kelton, Aruna Balasubramanian, Ramya Raghavendra, and Mudhakar Srivatsa. Browser-based deep behavioral detection of web cryptomining with coinspy. In *Workshop on Measurements, Attacks, and Defenses for the Web*, 2020.

[16] Kevin Borgolte, Christopher Kruegel, and Giovanni Vigna. Delta: automatic identification of unknown web-based infection campaigns. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 109–120, 2013.

[17] Davide Canali, Marco Cova, Giovanni Vigna, and Christopher Kruegel. Prophiler: a fast filter for the large-scale detection of malicious web pages. In *Proceedings of the 20th international conference on World wide web*, pages 197–206, 2011.

[18] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of static analysis for malware detection. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pages 421–430. IEEE, 2007.

[19] Paruj Ratanaworabhan, V Benjamin Livshits, and Benjamin G Zorn. Nozzle: A defense against heap-spraying code injection attacks. In *USENIX security symposium*, pages 169–186, 2009.

[20] Florian Tegeler, Xiaoming Fu, Giovanni Vigna, and Christopher Kruegel. Botfinder: Finding bots in network traffic without deep packet inspection. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 349–360, 2012.

[21] Jianglang Feng, Baojiang Cui, and Kunfeng Xia. A code comparison algorithm based on ast for plagiarism detection. In *2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies*, pages 393–397. IEEE, 2013.

[22] Jingling Zhao, Kunfeng Xia, Yilun Fu, and Baojiang Cui. An ast-based code plagiarism detection algorithm. In *2015 10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA)*, pages 178–182. IEEE, 2015.

[23] Tuan-Hung Pham, Ninh-Thuan Truong, and Viet-Ha Nguyen. Analyzing rbac security policy of implementation using ast. In *2009 International Conference on Knowledge and Systems Engineering*, pages 215–219. IEEE, 2009.

[24] Xin Wang and Xiaojie Yuan. Towards an ast-based approach to reverse engineering. In *2006 Canadian Conference on Electrical and Computer Engineering*, pages 422–425. IEEE, 2006.

[25] Wayback machine. Website, 2020. https://archive.org/web/.

[26] J-Michael Roberts. Virus share.(2011). *URL https://virusshare. com*, 2011.

[27] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th {USENIX} symposium on networked systems design and implementation ({NSDI} 16)*, pages 45–59, 2016.

[28] Minedetector. Website, 2020. https://www.github.com/WhileBug/MineDetector.

[29] Tin Kam Ho. Classification technique using random decision forests, July 27 1999. US Patent 5,930,392.

[30] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[31] Scikit-Learn. Decision trees. Website. https://scikit-learn.org/stable/modules/tree.html Accessed June 4, 2020.