



National
College of
Ireland

Machine Learning Approaches to Detect Browser-Based Cryptomining

MSc Internship
MSc in Cyber Security

Sherwin Norman Xavier

Student ID: 19126662

School of Computing
National College of Ireland

Supervisor: Prof. Vikas Sahni

**National College of Ireland
Project Submission Sheet
School of Computing**



Student Name:	Sherwin Norman Xavier
Student ID:	19126662
Programme:	MSc in Cyber Security
Year:	2020
Module:	MSc Internship
Supervisor:	Prof. Vikas Sahni
Submission Due Date:	17/08/2020
Project Title:	Machine Learning Approaches to Detect Browser-Based Cryptomining
Word Count:	5656
Page Count:	17

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

I agree to an electronic copy of my thesis being made publicly available on NORMA the National College of Ireland's Institutional Repository for consultation.

Signature:	
Date:	16th August 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	
You must ensure that you retain a HARD COPY of the project , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Machine Learning Approaches to Detect Browser-Based Cryptomining

Sherwin Norman Xavier
19126662

Abstract

Modern browsers provide support to multiple APIs that make use of JavaScript leading to an increase in developing useful web applications but also malicious activities. One such malicious activity is browser-based cryptomining. Browser-based cryptomining activities are a way to hijack a user's system without any permission from the end-user. This is a result of a rise in the popularity of cryptocurrencies that support mining activities requiring a CPU. This study proposes two approaches, static analysis and dynamic analysis to detect browser-based cryptomining activities. The static analysis uses the complexity features of a JavaScript code to design, implement and evaluate three unsupervised machine learning models while the dynamic analysis uses the different performance parameters of a system to design, implement and evaluate three supervised machine learning models. Ultimately, One-Class SVM anomaly detection model performed well for the static analysis with an accuracy of 78.9% while KNN classification model performed well for the dynamic analysis with an accuracy of 98.8%. Matthew's Correlation Coefficient statistical test results supported the results of this study.

Keywords – Cryptojacking, Browser, Machine Learning, Static Analysis, Dynamic Analysis.

1 Introduction

In recent times the popularity of cryptocurrencies has risen enormously leading to a significant rise in the demand for mining. This demand keeps rising as it is the basis for the cryptocurrency to stay in position. The increasing number of cryptocurrencies have led to several financial crimes where criminals misuse cryptocurrency for ransom [1]. The demand for mining has also given rise to a new form of malicious activity called cryptojacking wherein the cybercriminal mines cryptocurrency using a victim's system resources. To date, cryptojacking has been reported in the form of botnets [2,3] as well as in cloud infrastructures.

For an individual to begin the mining process they would have to invest in both hardware as well as in cooling equipment. The individual would also have to incur high electricity costs for the hardware to run smoothly. Thus, to avoid these extra costs to mine cryptocurrency the attackers made unauthorised use of another's system resources which could either be a computer or mobile phone. This could even lead to the victim suffering financial impact due to the high electricity costs and possible damage to the system's hardware if mining is done on an unsuitable device.

There have been technologies developed to detect such cryptomining activities. But as time progressed attackers developed ways to avoid these detection techniques, one of which is for the attacker to stop using the victim's system while the victim uses it.¹

¹ <https://www.cyberthreatalliance.org/wp-content/uploads/2018/09/CTA-Illicit-CryptoMining-Whitepaper.pdf>

Another method is to make use of only valid processes of the operating system for the mining process.

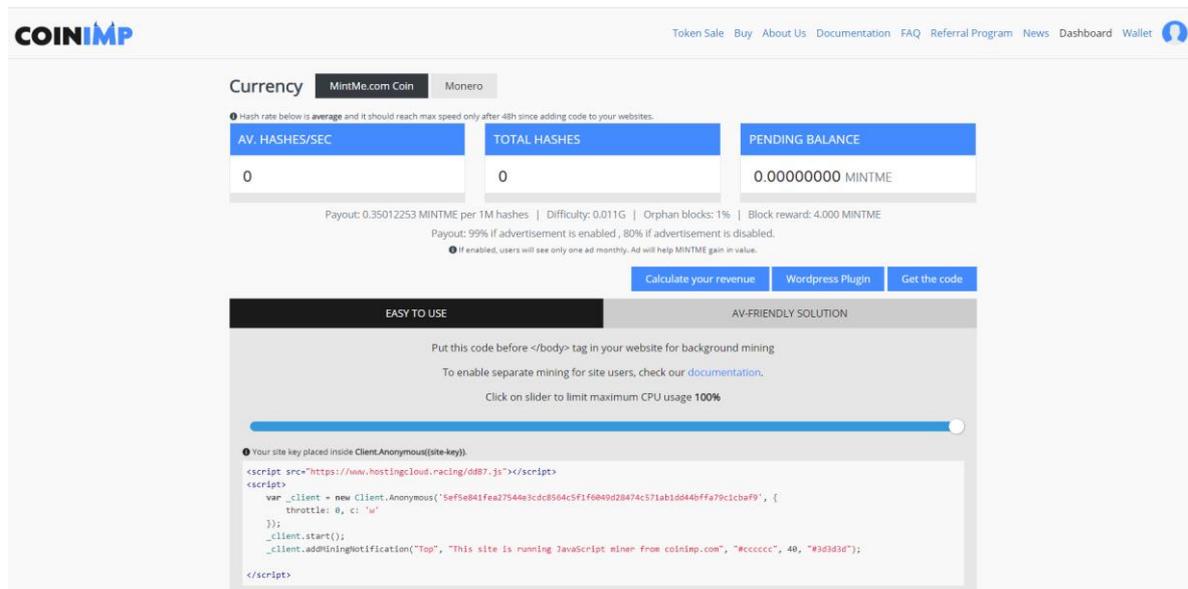


Figure 1: CoinImp providing mining JavaScript

Another worrying factor is services like CoinImp and Crypto-Loot provide cryptocurrency mining by distributing the JavaScript code for mining purposes to add in a web page. This helps the website to make a profit by mining cryptocurrency using their visitors' system resources. The aforementioned reasons make it necessary to develop up-to-date methods to detect unauthorised cryptomining activities.

Research Question

- i. **Can machine learning help to detect browser-based cryptomining activities?**
 - To test different supervised and unsupervised machine learning models to achieve accurate results in detecting such attacks.
- ii. **Can JavaScript complexity features help in detecting cryptojacking script?**
 - To test the dynamic characteristics of a malicious JavaScript instead of the static characteristics which is generally taken into consideration for detecting a cryptojacking script but this method cannot detect a cryptojacking script if it is obfuscated.
- iii. **Do the performances of memory, processes and filesystems help in detecting cryptomining activities?**
 - Usually, only the CPU usage of a system is considered while determining if the system is cryptomining but in recent times the attackers have begun using scripts that allow them to reduce the throttle level to avoid such detection systems.

The remainder of this report is organised as follows: Section 2 describes the background for this study. Section 3 describes a critical review of similar works related to the study. Section 4 describes the methodology adopted for this study along with the features used to design the models for the two different approaches. Section 5 describes the framework followed for the two approaches. Section 6 gives a brief description of the tools and methods used for the two approaches. Section 7 describes the evaluation of the various models used for the two approaches along with a critical review of the experiments carried out. The conclusions for the study and future work are described in Section 8.

2 Background

This section reviews the basis for this research by also including an introduction about cryptojacking, the mining process in browsers and its legality.

2.1 Cryptojacking

In cryptojacking, the attackers make use of two methods to mine cryptocurrency for gaining unapproved use of the target system. One method involves loading the cryptomining code as a stand-alone binary on the target machine with access to the machines operating system and hardware. The focus of this study is on another method of cryptojacking which occurs when a user loads onto an infected website through their browser, allowing the website to execute the malicious cryptomining script onto the system. In both cases, the target continues using the machine unaware of the mining code working in the background.

Continuous hashing due to mining leads to abuse of the machine. Reduced computing resources for other tasks is a clear indication of system abuse. A minimal rise in electricity bills due to the processor using greater power is also an indication of continuous mining. The cooling expense also increases due to the excess heat generated by the processor. Constant mining-induced stress lowers the life span of the target machine. Hence, early detection of cryptojacking becomes imperative.

2.2 Browser-based Mining

A JavaScript code injected into a web page allows access to a target device, enabling the abuse of device resources to mine cryptocurrency. Browser-based mining occurs when the JavaScript code runs via an infected website open on a browser. This makes browser-based mining platform-independent, as web pages can run on any JavaScript-enabled browser and is accessible on most host devices such as mobile phones, tablets, computers, etc. due to JavaScript's popularity as a default language on common web browsers. In-browser cryptomining reduces the use of custom hardware required initially for mining purposes as all users accessing the infected website automatically become targets of cryptomining contributing to the ideal processing power required.

2.3 Cryptojacking Workflow

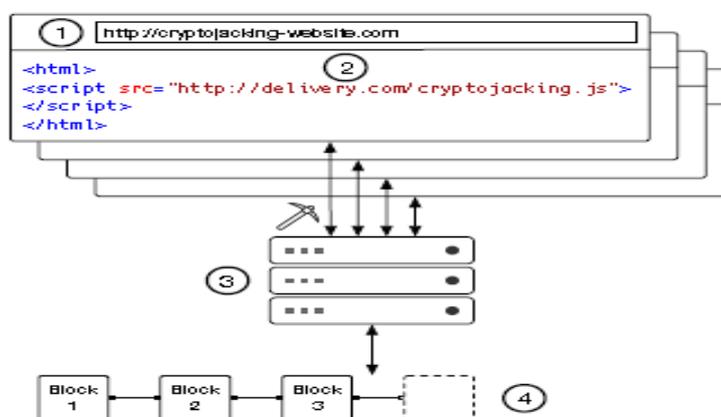


Figure 2: Cryptojacking workflow [4]

Cryptonight a new proof-of-work algorithm reduced drastically the merits of custom hardware. Cryptonight is a memory-bound protocol which depends on random access to slow memory, with every new block depending on the outcome of the previous block. This development created cryptojacking, helping mine cryptocurrency. The cryptojacking workflow shown in Figure 2. involves several elements. Cryptojacking websites loaded on a user's JavaScript-enabled web browser run the cryptojacking JavaScript code either because of a design error or an attacker injected malicious script. The cryptojacking code done on a domain similar to the original website will be delivered from a delivery server.

Mining jobs are received once the cryptojacking code connects to the mining pool proxy on execution. There is a proxy server deployed between the mining pool and miners to divide the mining jobs that are allotted by the mining pools and to restrict connections up to a certain number. Every miner will already have an account setup with the mining pool to receive payments. The mining pool will assign an account determiner to the account holder to correctly identify mining output from a particular account, communicating with it. The mining pool monitors the blockchain and accordingly assigns miners mining jobs connected to the blockchain. The jobs are a lower version of actual proof-of-work, making it easier to record and reward gradual mining. Completion of a single job results in a block that is added to the blockchain granting revenue for that mining activity. Part of the resultant award will then be distributed among the mining accounts based on the number of jobs done by the miner, while part will be kept by the mining pool as fee [4].

2.4 Ethics and Legality

Ethically vague laws around cryptojacking keep it in the grey. Without gaining permission or informing a user can be determined as stealing of the user's resources, making cryptomining ethically wrong. Researchers found that most users are not aware of the details of their agreed upon consent. There are few legal cases brought up since the discovery of cryptojacking. Tidbit, a company based on in-browser mining faced a lawsuit in 2015, eventually having to shut down. The Attorney General of New Jersey ruling expressed "No website should tap into a person's computer processing power without clearly notifying the person and giving them the chance to opt out"¹. This decision sets a precedent for future lawsuits.

3 Related Work

This section reviews significant research works that have been carried out previously to detect, analyse and prevent cryptojacking.

3.1 Cryptojacking

Bertino and Nayeem studied the worms in IoT devices which were responsible for carrying out covert mining on a system [5]. Krishnan et al. investigated the computer malware such as HKTL BITCOINMINE and TrojanRansom.Win32.Linkup which carried out covert mining activities on a computer system [6]. In the form of browser extensions, there are a couple of solutions present to avoid cryptojacking such as *NoMiner* which is a

¹ <https://nj.gov/oag/newsreleases15/pr20150526b.html>.

popular option in this context. The *No Coin* web extension was studied by Ruth et al. [7] to check the rise of cryptojacking by analysing sites that were blacklisted by the extension. These add-ons found almost 1,491 suspected cryptomining websites after scanning the huge list of websites made available by Alexa's Top 1M list. Both these add-ons offer protection to a certain extent, but an attacker could easily avoid their algorithms using simple obfuscation methods using URL matching techniques.

The popularity of cryptojacking was also studied by Eskandari et al. [8] by looking into the vast use of *Coinhive*. However, to study the illegal mining activities in detail this study did not conduct any static or dynamic analysis of cryptojacking scripts.

Vierthaler et al. developed WebEye that collects harmful HTTP traffic automatically [9]. They used sources such as Alexa Top 1M websites, MalwareDomainList and Openphish to get their URLs and input them in their web-crawler based on Selenium. Along with the metadata acquired from GeolIP and Whois, WebEye also extracts 58 other features from web applications.

Jayasinghe et al. in their study [10] analysed 11 forms of cryptojacking attacks on cloud infrastructures. They have studied the different features of every attack and compared it with detection systems to find limitations in them.

3.2 Static Analysis

The impact by a malignant JavaScript code on browsers and machines have also been studied. Browser-based cryptojacking was studied in detail by Hong *et al.* [11]. *CMTracker* a machine-learning tool was developed to prevent cryptojacking by Hong et al. which carried out static analysis on 2,770 cryptomining websites.

A code-based analysis was carried out on 13 cryptojacking platforms by Konoth et al. [12] to provide solutions against cryptojacking by studying the different details in a JavaScript code. Based on this they were able to create MineSweeper to detect scripts seeking the use of L1 and L3 cache of the CPU and features of cryptomining in WebAssembly.

SEISMIC (Secure In-lined Script Monitors for Interrupting CryptoJacks) was created by Wang et al. using semantic features instead of syntactic to detect web browser cryptomining [13]. The authors monitored cryptomining scripts that contain WebAssembly (Wasm) to run in the browser that performs similar to non-malicious code to distinguish between mining and non-mining behaviour. The authors accepted that their method is good against syntactic obfuscation but to avoid detection semantic obfuscation could be used with compromise on performance.

Subsequently, *Outguard* was developed as a cryptojacking detection tool by Kharraz et al. [4] making use of the SVM classifier algorithm to detect secret cryptomining activities with an accuracy rate of $\approx 97\%$. They utilized 6,302 websites to retrieve seven features to train their SVM model. Although, dynamic analysis to examine the effects of cryptomining scripts on a victim's device was not carried out. Moreover, as accepted by Kharraz *et al.*, to create *Outguard* a huge drawback was the use of a supervised learning model.

Cova et al. also used machine learning methods to find any malignant JavaScript code in web pages that download the malware on the target system in the background to spread [14]. They extracted features related to four different aspects namely, redirection, de-obfuscation, environmental context and exploitation. Their method also identified obfuscated code and created signatures that could be detected for signature-based systems.

Zozzle, a JavaScript malware detection tool developed by Curtsinger et al. identified benign and malicious code by extracting features related to the abstract syntax tree using the Bayesian classification [15]. It extracts these features by hooking calls to the eval function using the library Detours instrumentation to identify JavaScript de-obfuscation in Internet Explorer.

3.3 Dynamic Analysis

Tahir et al. [16] a tool *MineGuard* was created to identify secret mining activities on the cloud in real-time. For detection, *MineGuard* developed discernible signatures for mining algorithms by using hardware-assisted profiling. A browser extension was developed by utilizing minute micro-architectural impressions for identifying cryptojacking to expand their analysis to browser-based cryptojacking.

BMDetector developed by Liu et al. was a framework that hooks JavaScript in the kernel source of Chrome Webkit and analyses the features of the data structures gained from the browser heap snapshot and stack data to detect web browser cryptomining activities [17]. Before carrying out performance extraction we need to avoid obfuscation and encryption hence this framework captures the data at the parser level of the browser. Using these features the BMDetector carries out detection built on Recurrent Neural Networks (RNN).

RAPID, was developed to use resource and API-based detection technique to identify in-browser cryptomining by Parra Rodriguez et al [18]. It was able to classify mining with an accuracy of 96%. Similarly, Carlin et al. proposed that analysis of dynamic opcodes could detect a cryptomining activity on web pages [19]. This helped in distinguishing between benign websites and cryptomining websites using the random forest method to classify tasks with higher accuracy.

CoinSpy, an in-browser tool was built using deep learning techniques to identify in-browser cryptomining by Kelton et. al. [20]. This tool makes use of compute signatures, memory signatures and network signature to achieve an accuracy rate of 97% to detect cryptomining activities.

The study done by Barbhuiya et al. [21] analyse network traffic and CPU utilization to detect DDoS and cryptojacking attacks respectively in cloud infrastructures. Through their study, they were able to detect the two attacks but failed to distinguish between a DDoS attack and a cryptojacking attack. Also, the datasets used to create their classification models have not been separated.

This study closely resembles the work by Saad et al. [22] who have carried out static analysis based on the complexity features of a JavaScript code and applied a clustering algorithm to identify cryptojacking scripts and dynamic analysis by observing changes in the CPU usage. Web Socket packet sizes and the rate of power consumption. While they have made use of only four features to design the clustering model, this study has taken more features from the dataset to improve its accuracy and have made use of unsupervised anomaly detection algorithms. Also, for the dynamic analysis, this study made use of not just the CPU usage but also considered the usage of memory, processes and filesystems for preparing a supervised machine learning model to detect covert cryptomining.

4 Methodology

4.1 Process Flow

Figure 3 shows the step-by-step process taken to implement the two methods to meet the final objective of the study.

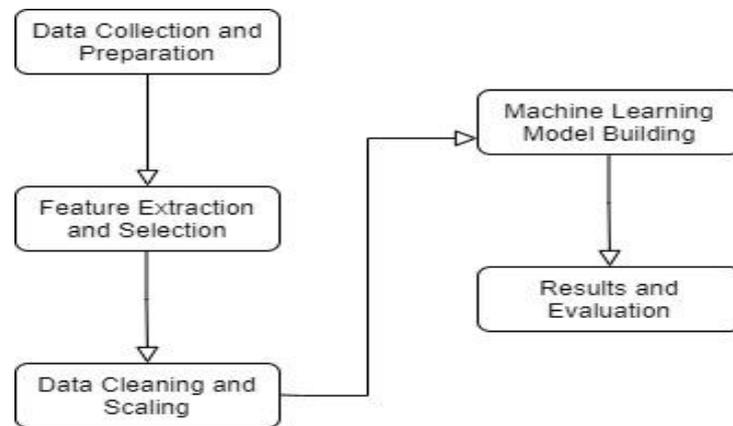


Figure 3: Step-by-Step Process

4.2 Data Collection and Preparation

For the static analysis, this study collected the cryptomining scripts from the dataset made available by Queen’s University, Belfast [23]. There were scripts of 8 unique cryptomining families i.e. Coinhive, CoinImp, CryptoLoot, deepMiner, JSECoin, Papoto, ProjectPoi and WebMinePool. To avoid having an imbalanced dataset, an equal number of benign scripts had to be considered to design the unsupervised model. For the benign JavaScript, this study extracted the code from non-cryptojacking websites ¹ using an online extracting tool. In total, the study uses 8 cryptojacking scripts along with 11 benign scripts to form the dataset.

For the dynamic analysis, the dataset used for this study was made available by Jayasinghe [24] where a server instance’s performance was analysed during a cryptomining attack.

4.3 Feature Extraction and Selection

For the static analysis, this study made use of a JavaScript code complexity tool, Plato. By running Plato, the different features related to the complexity of a JavaScript code were recorded [25]. The features are as follows:

a. Source Lines of Code

Source Line of Code (SLOC) is the calculation of the total lines of code excluding the white spaces. This helps identify how productive and maintainable the program is.

b. Cyclomatic Complexity (M)

Cyclomatic Complexity is a metric that tells how many paths the code takes on execution. The higher the number of paths, the higher the complexity of the code. This is calculated using the Control Flow Graph. This is calculated as $M=E+2Q-N$ where E

¹ <https://www.creativebloq.com/web-design/examples-of-javascript-1233964>

is the number of edges, Q is the number of connected components and N is the number of nodes in the Control Flow Graph.

c. Cyclomatic Complexity Density (M_d)

This is a measure of the Cyclomatic Complexity of the code. This helps identify any obfuscated code even when an attacker increases the size of the code the complexity stays the same. It is calculated as $M_d = E + 2Q - N / c_t$ where c_t is the number of total lines of code.

d. Halstead Complexity Metrics

These metrics are based on understanding the code as a series of tokens and classify every operator as an operand or an operator. This is counted as the total number of operators ($N1$), total number of operands ($N2$), number of unique operators ($n1$) and number of unique operands ($n2$). With the help of these metrics, other measures such as Volume (V), Effort (E), Bugs (B), Time (T), Difficulty (D) were derived and used in the study.

e. Maintainability Score (M_s)

This score is calculated using total lines of code in the file (c_t), Cyclomatic Complexity (M) and Halstead's Volume(V). $M_s = 171 - 5.2\log(V) - 0.23M - 16.2\log(c_t)$

For the dynamic analysis, the study has combined two datasets `final-normal-dataset.csv` and `final-anormal-data-set.csv` and added a new column `target` that comprised of binary values `0` for normal usage and `1` for abnormal usage of the system resources. For the feature selection, this study has applied the Two-Tailed Z-test along with the principle of multicollinearity to check the significance level of every feature from the dataset. Due to this, the final dataset was narrowed down to 29 features. These features are mentioned in Table 1.

Table 1: System Resource Features used in the dataset.

Sr. No.	Feature	Description
1	cpu_idle	Percent of total CPU spent idle.
2	cpu_iowait	Percentage of total CPU spent waiting for I/O to complete.
3	cpu_nice	Percentage of total CPU time spent in lowest-priority user processes.
4	cpu_steal	Percentage of total CPU spent waiting for other OS when running in a virtualized environment.
5	cpu_system	Percentage of total CPU time spent in kernel mode.
6	cpu_total	Total number of processors installed
7	cpu_user	Percent of total CPU time spent in normal user processes
8	diskio_sda1_read_bytes	Kbytes read per second by the disk sda1
9	diskio_sda1_write_bytes	Kbytes written per second by the disk sda1
10	diskio_sda_read_bytes	Kbytes read per second by the disk sda
11	fs/_free	Total kilobytes available to be used in the filesystem
12	fs/_percent	Percent of a filesystem

13	fs/_used	Total kilobytes of space used in the filesystem
14	load_cpucore	Number of CPU cores on the system
15	load_min1	The average load over the last minute.
16	mem_active	Recently accessed active memory.
17	mem_available	The amount of memory that is available.
18	mem_buffers	Temporary storage used for raw disk blocks.
19	mem_inactive	Inactive memory that has not been accessed yet.
20	mem_shared	Total memory shared.
21	mem_total	Total usable RAM
22	mem_used	Total RAM used at present
23	memswap_used	Total used swap space for RAM
24	percpu_0_iowait	Total idle time of CPU when there was an outstanding task
25	percpu_0_softirq	Total interrupt requests on a cpu
26	percpu_0_system	CPU Performance of the system
27	processcount_running	Total number of processes at present running on the system
28	processcount_thread	Total number of threads used by the processes
29	processcount_total	Total number of processes

4.4 Data Cleaning and Scaling

In the dynamic analysis, the dataset had varying values in all the features which made it necessary to normalize the data across all the features. Hence, the values of all features have been transformed to within the range of 0 and 1. This transformation has been done by performing the min-max scaling to keep distributions between variables the same. The final dataset was highly imbalanced but there was no sampling carried out on this dataset as the intention of this study is to correctly detect abnormal usage of a system from when normally used which is generally the case.

4.5 Machine Learning Model Building

For the static analysis, the study makes use of unsupervised machine learning models namely Isolation Forest, Local Outlier Factor and One-Class SVM. The data has been split into Inliers which are normal data and outliers which are abnormal data.

For the dynamic analysis, the study makes use of the supervised machine learning classification algorithms K-Nearest Neighbours, Support Vector Machines and Naive Bayes. The data has been split into 75% as training data while 25% as test data. The training features help the model to detect abnormal usage. The necessary packages and libraries required for both the models are installed and activated.

4.6 Results and Evaluation

The results for both processes indicate that some of the models used in this study were able to identify covert cryptomining behaviour on the system easily with a high accuracy rate. The confusion matrix showed the number of false negative and false positive cases the models made. The final results are supported by the Matthew's

Correlation Coefficient statistical test score and visualised with the help of Microsoft Excel to show these results.

5 Design Specification

This section gives a summary of the steps taken to carry out the two processes to identify cryptomining activity through a browser. These steps are illustrated in Figure 4. For both the processes, similar steps were involved except for the models that have been applied to get a result. These steps involve importing the respective datasets in the Python environment and carrying out pre-processing on it. It is then followed up with the feature extraction process to determine the steps for training the models to check for any abnormality in the data that has been fed. This is then followed by building the models and training them. For the static analysis, the unsupervised learning models Isolation Forest, Local Outlier Factor and One-Class SVM are used while for the dynamic analysis the supervised learning models K-Nearest Neighbours (KNN), Support Vector Machines (SVM) and Naive Bayes Classification. The final results then generated by the trained models are visually represented.

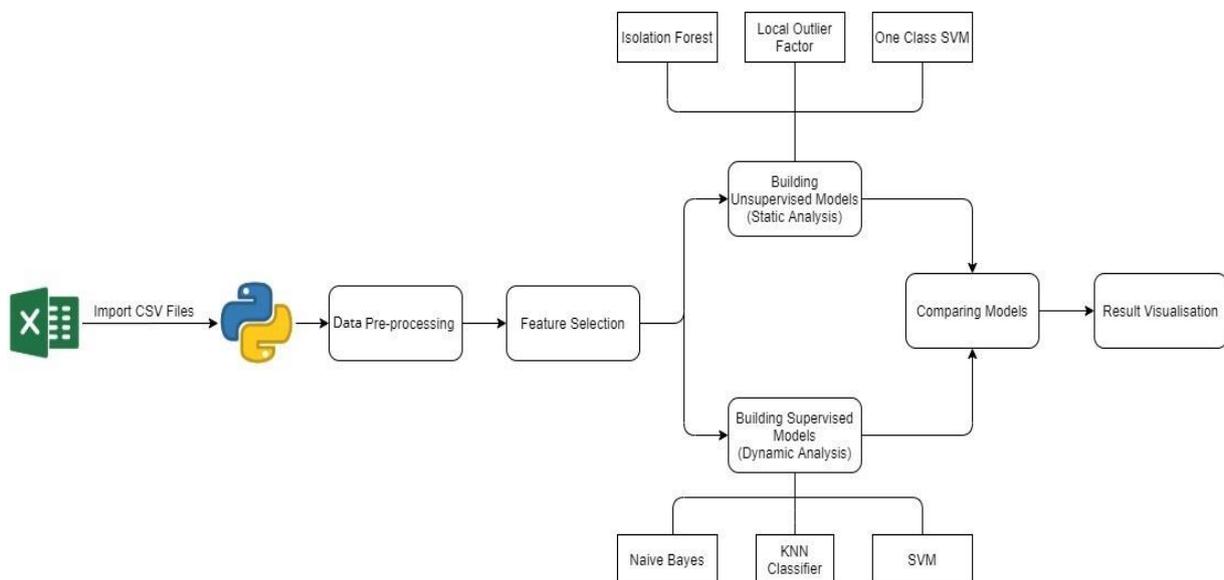


Figure 4: Framework for Different Approaches for In-Browser Cryptomining Detection

6 Implementation

This section describes the tools and functions used to setup the proposed models.

Hardware Specifications: For the successful implementation of this project, the study has been carried out on an 8th Gen Intel Core i5 Processor Laptop with 12 GB Ram.

Software Specification: The operating system on the laptop was Windows 10 Home 64-bit. Given below is a list of software used for the implementation of this project.

- **Python 3.8.1** – Python is a programming language that is quite popular and is supported on various IDEs. It is easy to learn and use, making it the main reasons why Python was chosen to build the proposed study models.

- **Jupyter Notebook** – This is an open-source web application that helps us in carrying out various functions. Additionally, it also supports Python which was the basis of the execution of the proposed models.
- **Python Packages** – Multiple packages of Python were used to implement the models for this study. These packages helped in performing various activities such as pre-processing on the dataset and applying the machine learning models. A list of these packages along with their versions is given in the Configuration Manual.
- **Node.js 12.18.2** – This is an open-source tool that helps execute JavaScript outside the browser. This was used to run the *Plato* tool to create the dataset for the static analysis.
- **Plato** – This tool is used to get the different features of the complexity of a JavaScript code. This tool was used to create the dataset used in the static analysis. This required Node.js as a prerequisite.

Two-tailed Z-Test: It is a statistical test used to carry out testing on a large dataset. This test has been used in the dynamic analysis of this research to determine significant features to make the machine learning models better. The test has been carried out on data of normal system usage and has tried to find a significant difference between the data of abnormal usage and normal usage

Hypothesis: H_0 : No difference between features

H_1 : There is a difference between the features

Population Data: Normal use of the resources

Sample Data: Abnormal use of the resources

Level of Significance: 0.01

Corresponding Critical Value: 2.58

Z-Score Formula:

$$Z_{score} = (\bar{x} - \mu) / S.E$$

Matthew's Correlation Coefficient (MCC): This test is used as a performance evaluation metric for machine learning models when there is a case of unbalanced data. This test is useful in both binary classification as well as multiclass classification [26]. Hence, this study takes into consideration the Matthew's Correlation Coefficient (MCC) value to support the results obtained by this study. The MCC value lies between -1 and +1 where the value +1 represents the model as perfect, while -1 represents the model as not good. It is calculated using the confusion matrix of every model by using the following formula where TP = True Positive Cases, TN = True Negative Cases, FP = False Positive Cases, FN = False Negative Cases:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}}$$

Machine Learning Models: This study involves three machine learning models for both static and dynamic analysis respectively which are mentioned below.

1. **Static Analysis:** Unsupervised machine learning models are used for this analysis to check its accuracy in identifying abnormality in the JavaScript code.
 - **Isolation Forest** – This is a newly developed technique that works on the principle of anomaly data points far from the normal data points. This results in the isolation of the anomaly data points.

- **Local Outlier Factor** – This is a technique to identify outliers in the data done by computing the density of deviation from the neighbouring data points of a single data point. If the density is lower of a data point compared to its neighbours it is considered as an anomaly.
 - **One Class SVM** – This model is fed only normal data for training and learns only normal transactions. When there is any data that does not resemble the normal data, it is considered as an outlier, while any data that resembles normal data will be considered as an inlier.
2. **Dynamic Analysis:** Supervised machine learning models were applied to the dataset for this study to check for the accuracy in the classification of abnormal usage and normal usage of system resources.
- **K-Nearest Neighbours (KNN)** – The KNN algorithm makes assumptions that similar data are close to each other. This algorithm works on the idea of calculating the distance between different points on a graph.
 - **Naive Bayes Classification** – The Naive Bayes classification algorithm is based on Bayes' theorem. This model assumes that the features are independent of each other to give a result.
 - **Support Vector Machines (SVM)** – This model is used for the classification of two groups problems. It works based on a line being plotted on the graph to divide the two classes for identification. The best line is determined by the distance of the line from the nearest data point which is the largest.

Output: The output consists of accuracy, precision, F-1 scores and confusion matrix which would be useful for evaluating the different models.

7 Evaluation

In this section, the results are shown for both the analysis carried out by applying different machine learning models for this project. The study has evaluated the models based on the test set for this report.

7.1 Static Analysis

In table 2, the evaluation metrics for the different unsupervised machine learning models applied for the static analysis are given and visually represented in Figure 5. It must be noted that the One-Class SVM has performed better than both Isolation Forest and Local Outlier Factor. The accuracy in distinguishing between the benign scripts and cryptomining scripts of One-Class SVM is 78.9% which is far greater when compared to the accuracy rate of the other models. The confusion matrix for the three models is shown in Figure 6 where the True Positive, True Negative, False Positive and False Negative values can be seen. The results obtained by applying the machine learning models for this analysis have been supported by taking into consideration the Matthew's Correlation Coefficient (MCC) values for them which are shown in Table 2. This value has been achieved by using each model's respective confusion matrix

Table 2: Evaluation Metrics of the Static Analysis

Model	Accuracy	Precision	Recall	F1-Score	MCC
Isolation Forest	0.526	0	0	0	-0.2
Local Outlier Factor	0.632	1	0.125	0.222	0.28
One-Class SVM	0.789	0.667	1	0.8	0.65

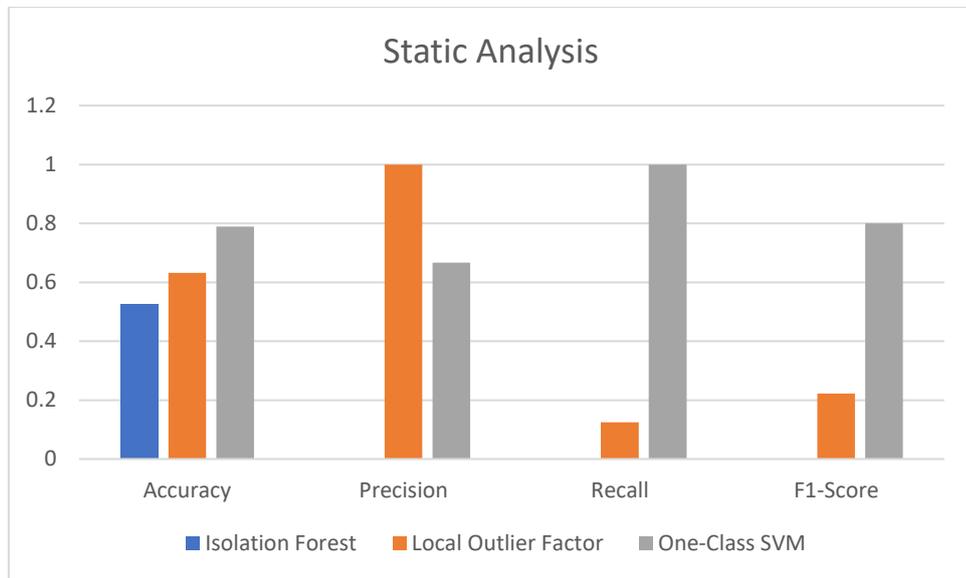


Figure 5: Visual Representation of the Evaluation of Static Analysis

Class	Benign	Cryptomining	Class	Benign	Cryptomining	Class	Benign	Cryptomining
Benign	10	1	Benign	11	0	Benign	7	4
Cryptomining	8	0	Cryptomining	7	1	Cryptomining	0	8

Isolation Forest
Local Outlier Factor
One-Class SVM

Figure 6: Confusion Matrix of the Different Unsupervised Models

7.2 Dynamic Analysis

In table 3, the evaluation metrics for the different supervised machine learning models applied for the dynamic analysis are given and visually represented in Figure 7. It can be noted that the KNN model has performed better than both SVM and Naive Bayes models. The accuracy in classification between the normal usage and abnormal usage of the system resources due to cryptomining scripts being run on the browser of the KNN model is 98.8% which is slightly better than the accuracy rate of the other two models. The confusion matrix for the three supervised models are shown in Figure 8 where the True Positive, True Negative, False Positive and False Negative values can be seen. These results have been supported by taking into consideration the Matthew's Correlation Coefficient (MCC) values for the three different models using their respective confusion matrix which is shown in Table 3.

Table 3: Evaluation Metrics of the Dynamic Analysis

Model	Accuracy	Precision	Recall	F1-Score	MCC
SVM	0.979	0.99	0.94	0.96	0.93
KNN	0.988	0.98	0.98	0.98	0.96
Naive Bayes	0.977	0.97	0.94	0.95	0.92

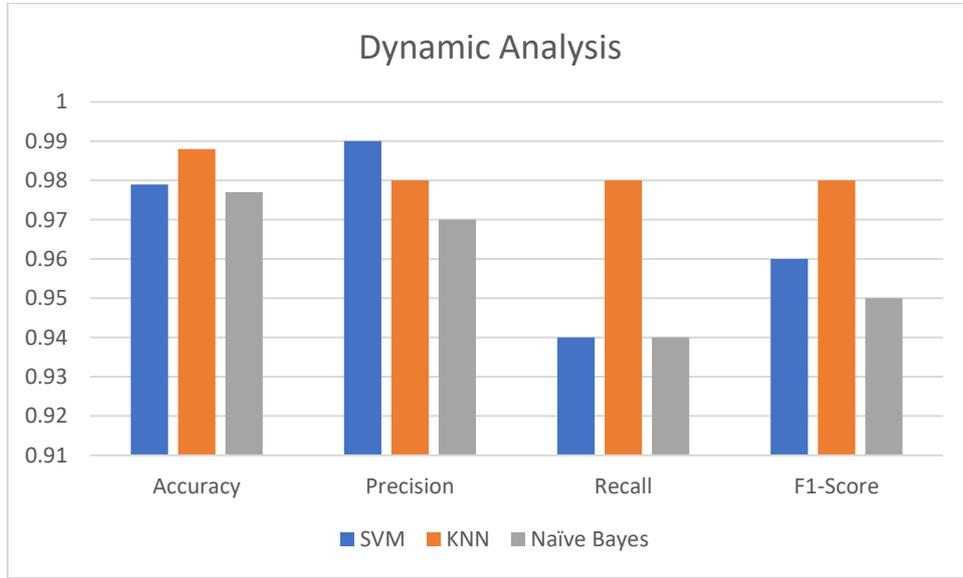


Figure 7: Visual Representation of the Evaluation of Dynamic Analysis

Class	Benign	Cryptomining
Benign	19780	0
Cryptomining	492	3556
SVM		

Class	Benign	Cryptomining
Benign	19638	142
Cryptomining	134	3914
KNN		

Class	Benign	Cryptomining
Benign	19721	59
Cryptomining	487	3561
Naive Bayes		

Figure 8: Confusion Matrix of the Different Supervised Models

7.3 Discussion

For the static analysis, the results are acceptable to detect cryptomining scripts. The dataset used for the static analysis is limited and therefore prevents a complete understanding of how the research could work with a larger dataset. However, the reliability and validity of the research process used makes replicability for a larger dataset possible. For the initial research, it can be said that the One-Class SVM model is a good fit for the static analysis. It can also be seen that the accuracy rate of the anomaly detection models is not as good as the accuracy rate obtained by [16].

For the dynamic analysis, all three models were able to identify any sudden rise in the usage of the system resources. Reducing the number of features with the help of Z-test and by applying the principle of Multicollinearity aided in getting better results for this analysis. As the dataset was created from a Linux OS these results could be valid only for a Linux distro and may vary for other operating systems. High use of the system resources could be possible due to video editing programs and/or gaming which could need higher processing power. Hence, just based on high resource consumption one cannot say if there is covert cryptomining taking place on one's system. These good results could also be a result of overfitting of data due to the highly imbalanced dataset.

But as this study focusses on detecting cryptomining during normal usage of a system the models must be able to detect abnormal activity during such times. Matthew's Correlation Coefficient values for every model used in this study have supported the results obtained by them.

8 Conclusion and Future Work

This study presented two approaches to detect a cryptojacking attack on a system. It has used a statistical approach to identify the features for the dynamic analysis. The final unsupervised models implemented for the static analysis did not perform well with an average accuracy as compared to that of the previous work. On the contrary, the supervised models implemented for the dynamic analysis were highly successful in detecting a cryptojacking attack. For static analysis, One-Class SVM has outperformed the other two unsupervised models while for dynamic analysis KNN has outperformed the other two supervised models in detecting cryptojacking. The results of this research help answer the main question that set this study in motion, 'Whether machine learning can help to detect an in-browser cryptojacking attack?'. Also, existing literature was studied for their techniques and limitations.

The future work on this study would be to work with a larger dataset for the static analysis which was a limitation in this study. Also, for the dynamic analysis evaluation of multiple operating systems for their performance parameters during a cryptojacking attack would widen the scope of this kind of research. There could even be a browser extension developed using the two approaches mentioned in the study to detect abnormal activity on the system in real-time on a web page.

References

- [1] P. H. Meland, Y. F. F. Bayoumy, and G. Sindre, "The Ransomware-as-a-Service economy within the darknet," *Computers & Security*, vol. 92, p. 101762, May 2020, doi: 10.1016/j.cose.2020.101762.
- [2] D. Y. Huang *et al.*, "Botcoin: Monetizing Stolen Cycles," in *Proceedings 2014 Network and Distributed System Security Symposium*, San Diego, CA, 2014, doi: 10.14722/ndss.2014.23044.
- [3] N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, "CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data," in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, Nara, Japan, 2016, pp. 303–312, doi: 10.1109/ICDCS.2016.46.
- [4] A. Kharraz *et al.*, "Outguard: Detecting In-Browser Covert Cryptocurrency Mining in the Wild," in *The World Wide Web Conference on - WWW '19*, San Francisco, CA, USA, 2019, pp. 840–852, doi: 10.1145/3308558.3313665.
- [5] E. Bertino and N. Islam, "Botnets and Internet of Things Security," *Computer*, vol. 50, no. 2, pp. 76–79, Feb. 2017, doi: 10.1109/MC.2017.62.
- [6] H. Krishnan, S. Saketh, and V. Tej, "Cryptocurrency Mining – Transition to Cloud," *ijacsa*, vol. 6, no. 9, pp. 115-124 2015, doi: 10.14569/IJACSA.2015.060915.

- [7] J. R uth, T. Zimmermann, K. Wolsing, and O. Hohlfeld, "Digging into Browser-based Crypto Mining," *Proceedings of the Internet Measurement Conference 2018*, pp. 70–76, Oct. 2018, doi: 10.1145/3278532.3278539.
- [8] S. Eskandari, A. Leoutsarakos, T. Mursch, and J. Clark, "A first look at browser-based Cryptojacking," *arXiv:1803.02887 [cs, econ]*, Mar. 2018 [Online]. Available: <http://arxiv.org/abs/1803.02887>. [Accessed: 09-Jul-2020]
- [9] J. Vierthaler, R. Kruszelnicki, and J. Sch utte, "WebEye - Automated Collection of Malicious HTTP Traffic," *arXiv:1802.06012 [cs]*, Feb. 2018 [Online]. Available: <http://arxiv.org/abs/1802.06012>. [Accessed: 09-Jul-2020]
- [10] K. Jayasinghe and G. Poravi, "A Survey of Attack Instances of Cryptojacking Targeting Cloud Infrastructure," in *Proceedings of the 2020 2nd Asia Pacific Information Technology Conference*, Bali Island Indonesia, 2020, pp. 100–107, doi: 10.1145/3379310.3379323.
- [11] G. Hong *et al.*, "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*, Toronto Canada, 2018, pp. 1701–1713, doi: 10.1145/3243734.3243840.
- [12] R. K. Konoth *et al.*, "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*, Toronto Canada, 2018, pp. 1714–1730, doi: 10.1145/3243734.3243858.
- [13] W. Wang, B. Ferrell, X. Xu, K. W. Hamlen, and S. Hao, "SEISMIC: SEcure In-lined Script Monitors for Interrupting Cryptojacks," in *Computer Security*, vol. 11099, J. Lopez, J. Zhou, and M. Soriano, Eds. Cham: Springer International Publishing, 2018, pp. 122–142 [Online]. Available: http://link.springer.com/10.1007/978-3-319-98989-1_7. [Accessed: 09-Jul-2020]
- [14] M. Cova, C. Kruegel, and G. Vigna, "Detection and analysis of drive-by-download attacks and malicious JavaScript code," in *Proceedings of the 19th international conference on World wide web - WWW '10*, Raleigh, North Carolina, USA, 2010, p. 281, doi: 10.1145/1772690.1772720.
- [15] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert, "ZOZZLE: Fast and precise in-browser JavaScript malware detection," in *Proc. 20th USENIX Conference On Security 2011*, pp. 33–48 [Online]. Available: https://www.usenix.org/legacy/events/sec11/tech/full_papers/Curtsinger.pdf [Accessed: 09-Jul-2020].
- [16] R. Tahir *et al.*, "Mining on Someone Else's Dime: Mitigating Covert Mining Operations in Clouds and Enterprises," in *Research in Attacks, Intrusions, and Defenses*, vol. 10453, Cham: Springer International Publishing, 2017, pp. 287–310. [Online]. Available: http://link.springer.com/10.1007/978-3-319-66332-6_13. [Accessed: 09-Jul-2020].
- [17] J. Liu, Z. Zhao, X. Cui, Z. Wang, and Q. Liu, "A Novel Approach for Detecting Browser-Based Silent Miner," in *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, Guangzhou, 2018, pp. 490–497, doi: 10.1109/DSC.2018.00079.

- [18] J. D. P. Rodriguez and J. Posegga, "RAPID: Resource and API-Based Detection Against In-Browser Miners," in *Proceedings of the 34th Annual Computer Security Applications Conference*, San Juan PR USA, 2018, pp. 313–326, doi: 10.1145/3274694.3274735.
- [19] D. Carlin, P. OrKane, S. Sezer, and J. Burgess, "Detecting Cryptomining Using Dynamic Analysis," in *2018 16th Annual Conference on Privacy, Security and Trust (PST)*, Belfast, 2018, pp. 1–6, doi: 10.1109/PST.2018.8514167.
- [20] C. Kelton, A. Balasubramanian, R. Raghavendra, and M. Srivatsa, "Browser-Based Deep Behavioral Detection of Web Cryptomining with CoinSpy," in *Proceedings 2020 Workshop on Measurements, Attacks, and Defenses for the Web*, San Diego, CA, 2020, doi: 10.14722/madweb.2020.23002.
- [21] S. Barbhuiya, Z. Papazachos, P. Kilpatrick, and D. S. Nikolopoulos, "RADS: Real-time Anomaly Detection System for Cloud Data Centres," *arXiv:1811.04481 [cs]*, Nov. 2018 [Online]. Available: <http://arxiv.org/abs/1811.04481>. [Accessed: 09-Jul-2020]
- [22] M. Saad, A. Khormali, and A. Mohaisen, "Dine and Dash: Static, Dynamic, and Economic Analysis of In-Browser Cryptojacking," in *2019 APWG Symposium on Electronic Crime Research (eCrime)*, Pittsburgh, PA, USA, 2019, pp. 1–12, doi: 10.1109/eCrime47957.2019.9037576.
- [23] J. Burgess, "CryptoJacking Data (including raw HTML/JS files)." Queen's University Belfast, 2020 [Online]. Available: [https://pure.qub.ac.uk/en/datasets/cryptojacking-data-including-raw-htmljs-files\(ea782cda-b3ac-4fc3-b78b-c81324453280\).html](https://pure.qub.ac.uk/en/datasets/cryptojacking-data-including-raw-htmljs-files(ea782cda-b3ac-4fc3-b78b-c81324453280).html). [Accessed: 31-Jul-2020]
- [24] "Cryptojacking Attack Timeseries Dataset." [Online]. Available: <https://kaggle.com/keshanijayasinghe/cryptojacking-attack-timeseries-dataset>. [Accessed: 31-Jul-2020]
- [25] B. Curtis, S. B. Sheppard, P. Milliman, M. A. Borst, and T. Love, "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics," *IEEE Trans. Software Eng.*, vol. SE-5, no. 2, pp. 96–104, Mar. 1979, doi: 10.1109/TSE.1979.234165.
- [26] D. Chicco and G. Jurman, "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation," *BMC Genomics*, vol. 21, no. 1, p. 6, Dec. 2020, doi: 10.1186/s12864-019-6413-7.