Hunting Traits for Cryptojackers

Gabriel Jozsef Berecz[1,2][a] and Istvan-Gergely Czibula[1][b]
[1]*Department of Computer Science, Babeş-Bolyai University, M. Kogălniceanu Street, Cluj-Napoca, Romania*
[2]*Cyber Threat Proactive Defense Lab, Bitdefender, Romania*

Keywords: Cryptojacker, Computer Security, Learning-based Classifier.

Abstract: Cryptocurrencies are renowned world wide nowadays and they have been adopted in various industries. This great success comes from both the technology innovation they brought to the world, the blockchain, and the financial opportunities they opened up for investors. One of the unpleasant aspects are the cybercriminals who took advantage of this technology and have developed malicious software (i.e. cryptojacker) in order to gain profit by mining cryptocurrencies on their victims' personal computer without any consent. This paper proposes to analyze standalone cryptojackers, both statically and dynamically, with the aim of identifying specific traits. The approach draws out features specific to cryptojackers that are selected using statistical methods and explains why a cryptocurrency mining malware has such traits. Based on 20 selected specific features, three different supervised learning classification models have been trained, which are able to differentiate between clean applications and cryptojackers reliably. In experiments, an average accuracy of $\approx$ 92.46% has been achieved.

1 INTRODUCTION

In the past two years, cryptocurrencies have seen a rapid growth. By the start of 2018, they achieved a total market capitalization of 820 billion dollars (Coinmarketcap, 2013). This great success can be attributed to both the technology innovation it brought to the world, the blockchain, and the financial opportunities it opened up for investors. Mining is the process of validating transactions within the blockchain network. A miner is a machine connected to the blockchain network which validates transactions within the network (Swan, 2015). Cybercriminals took advantage of this technology and have developed malicious software in order to gain profit by mining cryptocurrencies on each victim's personal computer and collect the rewards in their wallets. Thus, cryptojackers have become wide spread over the Internet and many users are mining cryptocurrencies without consent. Since they have first appeared, cryptojackers have been noticed running inside a browser (i.e. browser-based cryptojackers that execute Javascript code) (Eskandari et al., 2018) and running locally on machines as standalone executables.

With the growth of Cryptocurrency market, cryptojackers have become widely spread among Internet users. They get executed through different methods, some of them are usually malicious software such as trojans, while other are delivered through complex attacks like exploits or file-less attacks. Because they are stealthy and, for the most of the time, unrecognized, they are able to persist on an affected machine for months. During the whole time, the attacker gains his profit while the user's hardware components are getting damaged. Once infected with such a threat, it could take minutes up to months to notice that something is wrong, but the attack has to be detected as early as possible in order to produce as little damage as possible. Thus, a specialized component responsible for identifying cryptojackers is required nowadays.

Moreover, Pastrana et al. (Pastrana and Suarez-Tangil, 2019) have studied cryptojacking campaigns presenting how attackers earned at least 56 million USD. Their study proved that malicious actors are nowadays targeting this kind of malware as it may be unnoticeable by its victims and delivering profit continuously. This represents only a part of the mining going on in the wild, not taking into consideration users and companies that are the victims of this phenomena without even knowing.

The term of cryptojacker refers to malicious software used to stealthily mine cryptocurrencies, without user's consent. We focused on cryptojackers that are standalone executable files as they have not been

[a] https://orcid.org/0000-0002-0923-9742
[b] https://orcid.org/0000-0003-0076-584X

Berecz, G. and Czibula, I.
Hunting Traits for Cryptojackers.
DOI: 10.5220/0007837403860393
In *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications (ICETE 2019)*, pages 386-393
ISBN: 978-989-758-378-0

studied in-depth, unlike the browser-based ones. The features were extracted based on behavioral analysis and static analysis of both cryptojackers and clean applications. The paper presents only those features that are relevant and have a major contribution in the classification process.

Based on an in-depth analysis of cryptojackers, along with a comparison between them and clean applications, this paper aims to answer the following questions:

- Are there any anomalies in the characteristics (i.e. features) of cryptojackers which could differentiate them from any clean application?
- Why would a cryptojacker have a specific feature present?
- Can we reliably separate clean applications from cryptojackers based on specific features?

The first question attempts to identify features present among many cryptojackers, the second one tries to explain the traits and behavior of such malicious applications, while the third implies an experimental evaluation. The main contributions of this paper can be summarized as follows:

- Identify features that are specific to standalone cryptojackers.
- Analyze and explain why would cryptojackers have such a trait or behavior.
- Introduce and experimentally validate cryptojackers classification methods based on the identified features.

The paper presents existing approaches and solutions to the problem of cryptojacking detection in Section 2. After that, the paper introduces our approach in Section 3 and continues with the experimental evidence of the proposed approach in Section 4. Finally, in Section 5 we compare the results obtained by our proposal with the ones already present in the literature. Conclusions and future improvements are discussed in Section 6.

2 RELATED WORK

Browser-based cryptojackers have been studied during the past years and systems aiming to identify such threats have been developed. The paper from Eskandari et al. (Eskandari et al., 2018) presented the recent trends in terms of cryptocurrency mining. According to their study, Coinhive (Coinhive, 2012) is the most used Javascript cryptocurrency miner, representing 92.0% of the total number of websites using Javascript cryptocurrency miners, followed in top 3 by JSEcoin (JSEcoin, 2017) and Crypto-Loot (CryptoLOOT, 2017). Saad et al. (Saad et al., 2018) focused on code-based analysis among browser-based cryptojackers, presenting code complexity features that are used to identify cryptojacking Javascript code with an accuracy of 96.4%. These two approaches present information related to browser-based cryptojackers, not for standalone applications.

Krishnan et al. (Krishnan et al., 2015) discuss about moving the mining process into cloud presenting the limitations of the traditional machines and why cloud computing is the next step. A study (Tahir et al., 2017) concerning cloud services was published, in which the authors presented how virtual machines from cloud are abused with the aim of mining cryptocurrencies. Another approach (Sari and Kilic, 2017) used Open Source Intelligence in order to investigate vulnerabilities present in mining pools, having the Mirai botnet (Antonakakis et al., 2017) as reference. Hong et al. (Hong et al., 2018) presented attack vectors specific to browser-based cryptojackers along with their distribution techniques. The framework proposed by Rauchberger et al. (Rauchberger et al., 2018) is specialized in scanning for cryptojackers Javascript code and it revealed that in Alexa Top 1 million websites, there were 3,178 which contain Javascript code mining cryptocurrencies.

In terms of generic malicious software detection, Zabidi et al. (Zabidi et al., 2012) extract features present among different malicious software families, such as trying to identify if there is a debugger attached to the running process or if the sample is running inside a sandboxed environment. The system proposed by Baldangombo (Baldangombo et al., 2013) et al. detects malicious software using features extracted from static analysis, using a total number of 25,592 features and 3 different classifiers (i.e. support vector machine, J48 and Naive Bayes) in order to achieve a detection rate of 99.6%. In a survey for automated dynamic malware analysis (Egele et al., 2012) authors presented different approaches, such as function call monitoring, functions parameters analysis, data flow tracking, instruction trace and persistence methods. Also, an automatic behavior-based classification of generic malicious samples (Devesa et al., 2010) has an average accuracy of 94.8% using four different classifiers. In order to detect polymorphic malware, Kim et al. (Kim and Moon, 2010) present how dependency graphs can be used to identify malicious scripts. All these approaches are trying to separate generic malware from clean applications, they are not targeting a specific malicious class (i.e cryptojackers) and they are either static or dynamic analyzing samples. Hybrid malware approaches have been published (Roundy and Miller, 2010) (Yang et al., 2015) (De Paola et al., 2018), but they are generically classifying malicious applications, not focusing on a specific prevalent family, the cryptojackers.

3 METHODOLOGY

Our approach for investigating and identifying standalone cryptojackers' specific features consists of the following steps:

- Step 1: Data gathering - collect various samples, both cryptojackers and clean applications.
- Step 2: Feature extraction - collect information from the gathered data.
- Step 3: Feature selection - choose features specific to cryptojackers using statistical methods.
- Step 4: Evaluation models - using the identified features, build different supervised learning models.
- Step 5: Performance evaluation criteria - measure performance for the evaluation models.

3.1 Data Gathering

We collected our data using the following sources:

- Hybrid-analysis (Hybrid-analysis, 2014) - automated malware analysis tool providing an in depth description for the submitted samples. We used this tool in order to gather applications labeled as cryptocurrency mining software.
- Malshare (MalShare, 2013) - an online malicious software repository that we used to gather more cryptojackers.
- Other sources - the clean applications were collected using Ninite (Ninite, 2009) software and Filehippo (FileHippo, 2004).

In order to get as close as possible to the ground truth, we have selected only clean applications that are not detected by any anti virus company and cryptojackers that are statically signed by at least 80% of anti virus companies.

3.2 Feature Extraction

Each sample from the data set was analyzed both statically and dynamically using custom tools developed specifically for this experiment (any other tool that extracts the below mentioned information is suitable). The static features extracted consist of:

- Information from the executable's header (it may differ as it depends on the used operating system)
- Entropy (Lyda and Hamrock, 2007)
- Imported modules and functions.
- Exported functions and modules.
- Information about every section (name, raw size and virtual size).

The second step in feature extraction was a dynamic analysis in which we monitored function calls (i.e. API calls specific to the operating system). The dynamic analysis step was required as many malicious software, not just cryptojackers, resolve their imports at runtime in order to avoid static detection (Moser et al., 2007). Also, many malicious applications, as well as clean applications, are delivered obfuscated and packed, thus avoiding static analysis.

3.3 Feature Selection

Having the aforementioned information, we wanted to see anomalies concerning cryptojackers. Thus, we compute the most used (imported or run time called) functions by cryptojackers and, at the same time, rarely used by clean applications. By doing this we can identify functions being frequently used by cryptojackers. We do the same for imported modules in order to hunt them, too. For every numerical feature extracted, we also separately compute the average value among cryptojackers and clean applications. Moreover, identifying features that are specific to clean applications, but are not present or rarely present at cryptojackers, will contribute to a better separation of these two classes. After having the final features, we are going to normalize every value using the formula:

$$f_i = \frac{x_i - min(x)}{max(x) - min(x)} \quad (1)$$

3.4 Evaluation Model

To experimentally evaluate our selected features we used three different classifiers already implemented in the scikit-learn (Pedregosa et al., 2011) Python library:

- Support vector machine classificator (Wang, 2005). The kernel function used for the algorithm, is the Radial Basis Function (Musavi et al., 1992) where γ is obtained using Hastie et. al's algorithm (Hastie et al., 2004):

$$e^{-\gamma||x-x'||^2} \quad (2)$$

- Multi-layer perceptron classifier (Riedmiller, 1994) experimentally tuned up using ReLU activation function (Xu et al., 2015) and Adam (Kingma and Ba, 2014) method for weight optimization.

- Random forest classifier (Pal, 2005) experimentally optimized.

3.5 Performance Evaluation Criteria

The first step in order to evaluate the performance is cross validation. We split the data set in two sets: training (consisting of 80% of the total samples) and prediction (consisting of 20%). In order to measure the model's generalization ability, we applied K-fold cross validation with k = 10. By doing so, the total number of samples was 10 times split into 10 different sets of training and prediction. In terms of performance, we measure the results using the area under the ROC curve (Metz, 1978) and the average value over the 10 cases for accuracy, precision, recall (true positive rate). They are calculated as equations (3), (4) and (5) show, where *tp* (i.e. true positive) represents the number of cryptojackers correctly classified, *tn* (i.e true negative) is the number of clean applications correctly classified, *fp* (i.e false positive) represents the number of cryptojackers classified wrong and *fn* (i.e. false negative) is the number of clean applications being classified wrong.:

$$accuracy = \frac{tp+tn}{tp+tn+fp+fn} \tag{3}$$

$$precision = \frac{tp}{tp+fp} \tag{4}$$

$$recall = \frac{tp}{tp+fn} \tag{5}$$

4 COMPUTATIONAL EXPERIMENTS

In order to validate our approach, computational experiments are needed. This section presents our results including the extracted features, the used models, accuracy, recall, precision and the ROC curve. All experiments have been conducted on Windows operating system and all samples are Windows portable executable files (Microsoft, 1991).

4.1 Data Set

The results presented in this paper are based on an analysis of 11,766 cryptojackers and 11,374 clean applications summing up to a total number of 23,140 samples. The data set was collected from diverse public sources such as honeypots, malicious URLs, malicious email attachments, as presented in Section 3. It contains various applications including packed applications and applications using evasion techniques (Afianian et al., 2018), such as importing modules and functions at runtime. All samples have been run on a machine having installed as operating system Windows 10 Redstone 3 (RS3). During the analysis, the machine was connected to Internet.

4.2 Feature Analysis

As we already mentioned in our methodology, we extracted statistics related to our features and in the next lines we analyze them. Table 1 presents functions frequently being used by cryptojackers but not by clean applications, ordered by the difference between cryptojackers usage and clean applications usage.

The results presented in Table 1 reveal that cryptojackers do have specific behavior that distinguishes them from clean applications. One noticeable difference is the frequent use of time related functions (i.e. _localtime(32/64), _ftime(32/64), _gmtime(32/64), etc.). They may be used during the mining process as seed to generate random numbers or simply for logging purposes. Another interesting fact is the usage of network communication related functions (i.e. inet_ntoa, getpeername) as cryptojackers have to communicate over the internet for mining purposes. Cryptojackers also use hashing functions during the mining process. As a result, cryptography related functions are also present in the top 10 (i.e. SHA256_Init and SHA256_Final).

Moving forward, we computed the same statistics, but applied to imported modules in order to identify what libraries do cryptojackers use. The results are presented in Table 2 and they confirm once again that cryptojackers frequently use network communication related modules (i.e. ws2_32.dll, libcurl-x.dll, wldapi32.dll, wtsapi32.dll). One noticeable remark about most used modules is that user32.dll (a library responsible for creating user interfaces) is not being used by cryptojackers as one of their goal is to remain as stealthy as possible.

Having identified functions and modules most frequently used by cryptojackers, we investigated their traits further and computed the average values for numerical values present in the portable executable header. Table 3 presents a comparison between cryptojackers and clean applications of the average value of five features extracted from static analysis. The five features were selected based on their significance and by the significance between the two average values. Cryptojackers tend to have a larger image size

Table 1: Top 10 functions frequently used by cryptojackers, but rarely by clean applications.

Function name	Cryptojackers usage (%)	Clean usage (%)	Difference
_localtime(32/64)	41.6388	2.7876	38.8512
_ftime(32/64)	37.3483	1.8357	35.5126
_gmtime(32/64)	32.9938	1.3598	31.6340
inet_ntoa	30.6664	4.2155	26.4509
getpeername	29.6817	5.7113	23.9703
curl_slist_append	20.5380	0.0679	20.4701
_time(32/64)	22.0151	2.0397	19.9754
_setusermatherr	22.0151	3.9435	18.0716
SHA256_Init	10.2690	0.3399	9.9291
SHA256_Final	10.2690	0.3399	9.9291

Table 2: Top 5 modules frequently used by cryptojackers, but rarely by clean applications.

Module name	Cryptojackers usage (%)	Clean usage (%)	Difference
ws2_32.dll	56.1983	9.7909	46.4074
libcurl-x.dll (*x* is the version)	16.4585	1.0878	15.3707
wldap32.dll	15.7552	0.8159	14.9393
opencl.dll	13.9264	0.1359	13.7905
wtsapi32.dll	10.8317	2.2437	8.5880

due to the fact that they are usually delivered without any external dependencies, the attacker aiming to infect as many victims as possible (for example, if a malware requires Python to be installed on a machine in order to successfully execute an attack, it reduces its possible victims as not all machines have Python installed). Also, it can be observed that cryptojackers import less functions, have less exports and a lower size of headers than clean applications as they have limited tasks to execute (these tasks refer to the cryptocurrency mining process), while a clean application usually is a much more complex piece of software.

To sum up, we have the top 10 functions frequently used by cyptojackers along with the top 5 imported modules and 5 features extracted during the static analysis based on their average values. Based on these 20 features we are going to train a classifier in order to separate between cryptojackers and clean applications.

4.3 Classification Model

Using the aforementioned features, we trained three different supervised learning (Duda et al., 2012) classifiers as mentioned in Section 3. We present the achieved results along with the used values for the classifiers' parameters.

The support vector machine classifier uses as a kernel function the following:

$$e^{-\gamma||x-x'||^2}, \gamma = 0.09 \quad (6)$$

The multi-layer perceptron architecture consists of 20 neurons on the input layer, two hidden layers with 16 neurons each, and one output layer with 1 neuron giving the probability of being a cryptojacker. The used learning rate is 0.0017, the penalization coefficient used by the Adam optimization (Kingma and Ba, 2014) is 0.0001 along with a maximum number of 50 iterations.

The random forest classifier uses 15 estimators, the Gini index (Gastwirth, 1972) as a measurement for a split quality and maximum depth of 8 nodes.

Using these 3 classifiers, we obtained an average accuracy of 92.4652% along with an average precision of 95.6311% and an average recall of 88.8128%. The results presented in Table 4 reveal that our model has a relatively low rate of false positives but tends to miss cryptojackers. The false negative rate is $\approx$ 11.2% (computed as $1 - recall$), taking into consideration that our model uses only 20 features that are targeted for one class' traits (i.e. cryptojackers), there is room for improvement and we talk about what can be further done in Section 6. Moreover, we present in Figure 1 the ROC curve for the trained models.

5 DISCUSSION AND COMPARISON TO RELATED WORK

Now that we have gathered and presented all our results, we are going to discuss about them and we will point out some major differences between our approach and the current available methods.

To begin with, in Section 4 we presented the av-

Table 3: A comparison of the average values for cryptojackers and clean applications on five features extracted from static analysis.

Feature name	Cryptojackers average value	Clean applications average value
SizeOfImage (bytes)	21,320,884.1981	10,539,628.71987
OverlaySize (bytes)	1,357,925.2727	4,755,346.9559
NumberOfImportedFunctions	141.7411	269.6894
SizeOfHeaders	1718.6666	1843.2000
NumberOfExports	28.5625	50.5263

Table 4: Average accuracy, precision and recall for the used classifiers.

Model	Acc.(%)	Precision(%)	Recall(%)
SVM	91.5207	97.5696	84.8525
MLP	92.1696	92.6273	91.3216
RF	93.7054	96.6966	90.2643
Average	**92.4652**	**95.6311**	**88.8128**

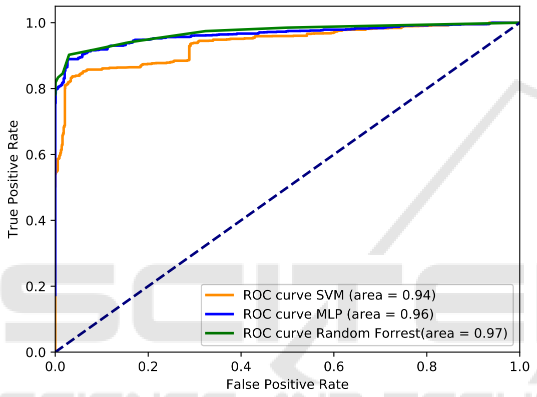

Figure 1: ROC curve for the used classifiers: SVM, MLP and Random forest.

erage accuracy, precision and recall for the classification models used. The average precision of 95.6311% along with the average recall of 88.8128% reveal that models based on cryptojackers specific traits are able to tell if an application is a cryptocurrency mining malware with very few false positives, but may miss cryptojackers. In a real world scenario, due to cryptocurrency mining malware popularity, a user is very likely to come over such a threat. Supposing that a common user comes up 10 new applications a month and few of them (hardly 5 if the user clicks on everything while he is browsing the internet) are cryptojackers (either they are downloaded from trusted sources or attached to malicious content on emails or delivered through other spreading techniques), a model based on specific cryptojackers traits is going to miss, on average, two cryptojackers in two and a half months while triggering only one false positive.

The above mentioned simulation on a real world scenario is not sufficient for a user to be fully protected, but it points out the core idea of our research paper. Cryptojackers do have specific traits that can be exploited in order to detect such threats. Moreover, cryptojackers are very different in terms of behavior from a clean application as they usually run stealthy, they intensively use APIs for network communication, hashing, APIs that may be used to get a seed for pseudo random number generators and they have less imported functions, less exported functions and their disk size is considerably higher as they aim to have no dependencies (this way they are able to run on different versions of the operating systems and on machines with missing libraries).

Approaches presented in the literature refer to generic malicious software detection. Multiple malware families are gathered into a single data set and experiments are thus conducted. Our approach focuses on a specific malicious software, cryptojackers, targeting their behavior and traits. By doing this, even if cryptojackers are delivered through different attack vectors (exploits, mail attachments, etc.) the payload (i.e. the mining component) is going to be identified. Also, our approach aims to identify specific traits to a prevalent malicious family in order to emphasize their importance in a classification problem. Baldangombo's et al. (Baldangombo et al., 2013) solution uses static analysis, if a cryptojacker is obfuscated as presented by Konoth et al. (Konoth et al., 2019), their solution is bypassed. Devesa et al. (Devesa et al., 2010) behavior based classification depends exclusively on API calls to identify malware behavior, but does not correlates this information with any static information.

The major importance of this paper is the fact that it presents how traits for standalone cryptojackers can be extracted in order to improve the security of a machine. It can be generalized to any operating system, but our focus was the Windows operating system as it is most used by regular users. On the other hand, the results presented in the literature are referring to browser-based cryptojackers and their focus is analyzing Javascript code, not the binary executable file responsible for mining cryptocurrencies.

6 CONCLUSIONS AND FUTURE WORK

Malicious software is everywhere across the Internet, from adware popping up on many websites up to ransomware encrypting user's file system. Security issues are discovered every day in different software and are quickly patched by developers. However, many users decline to update their systems, leaving their machines vulnerable to attacks, thus, making the attacks easier by targeting vulnerable machines. Malware industry has grown over the past twenty years and represents a way of living for attackers and has registered huge profit for them. New malware and new attacking vectors are developed every day and they are released in the wide world of the Internet. Cryptojackers have produced more than 56 million USD as stated by Pastrana et al. (Pastrana and Suarez-Tangil, 2019).

In order to give a final answer to the 3 main questions presented in the introductory part, we have to point out that:

- Yes, there are anomalies in the characteristics of a cryptojacker as we presented in Tables 1, 2, 3.
- In Section 5 we explained why cryptojackers have specific features present.
- We experimentally proved in Section 4 using three different classification models that cryptojackers can be reliably separated from clean applications using only 20 features.

Due to the high number of recent cryptojacking attacks, there is a need of a specific component in order to protect users from such attacks. In order to achieve that, cryptojackers had to be analyzed and features specific to them extracted. We presented three different classification models with an average accuracy of $\approx 92.46\%$ based on the features of cryptojackers along with its advantages, disadvantages and a comparison to existing solutions.

The current classification models rely on few features related to function calls and portable executable header. The main goal of the classification models is to provide an experimental validation for the extracted features and emphasize their importance. A substantial improvement would be to further analyze cryptojackers behavior and traits in terms of file system interactions (copied, written and read files), registry interactions (queried, opened, written registry keys) and network traffic (packet analysis). Thus, even in extreme cases where cryptojackers use custom made libraries and functions to achieve their goal, there would still be features revealing their malicious behavior. Another improvement would be to analyze the clean applications traits in comparison to cryptojackers (what traits are highly present at clean applications but are rarely present at cryptojackers) in order to better separate between these two classes.

REFERENCES

Afianian, A., Niksefat, S., Sadeghiyan, B., and Baptiste, D. (2018). Malware dynamic analysis evasion techniques: A survey. *arXiv preprint arXiv:1811.01190.*

Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Invernizzi, L., Kallitsis, M., et al. (2017). Understanding the mirai botnet. In *USENIX Security Symposium*, pages 1092–1110.

Baldangombo, U., Jambaljav, N., and Horng, S.-J. (2013). A static malware detection system using data mining methods. *arXiv preprint arXiv:1308.2831.*

Coinhive (2012). Coinhive monetize your business with your users cpu power. https://coinhive.com/. Accessed: 2018-12-09.

Coinmarketcap (2013). Cryptocurrency market capitalization. https://coinmarketcap.com/. Accessed: 2018-12-09.

CryptoLOOT (2017). Earn more from your traffic. https://crypto-loot.com/. Accessed: 2018-12-09.

De Paola, A., Gaglio, S., Re, G. L., and Morana, M. (2018). A hybrid system for malware detection on big data. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 45–50. IEEE.

Devesa, J., Santos, I., Cantero, X., Penya, Y. K., and Bringas, P. G. (2010). Automatic behaviour-based analysis and classification system for malware detection. *ICEIS (2)*, 2:395–399.

Duda, R. O., Hart, P. E., and Stork, D. G. (2012). *Pattern classification*. John Wiley & Sons.

Egele, M., Scholte, T., Kirda, E., and Kruegel, C. (2012). A survey on automated dynamic malware-analysis techniques and tools. *ACM computing surveys (CSUR)*, 44(2):6.

Eskandari, S., Leoutsarakos, A., Mursch, T., and Clark, J. (2018). A first look at browser-based cryptojacking. *arXiv preprint arXiv:1803.02887.*

FileHippo (2004). Download free software. https://filehippo.com/. Accessed: 2018-12-09.

Gastwirth, J. L. (1972). The estimation of the lorenz curve and gini index. *The review of economics and statistics*, pages 306–316.

Hastie, T., Rosset, S., Tibshirani, R., and Zhu, J. (2004). The entire regularization path for the support vector machine. *Journal of Machine Learning Research*, 5(Oct):1391–1415.

Hong, G., Yang, Z., Yang, S., Zhang, L., Nan, Y., Zhang, Z., Yang, M., Zhang, Y., Qian, Z., and Duan, H. (2018). 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. ACM.

Hybrid-analysis (2014). Free automated malware analysis service. https://www.hybrid-analysis.com/. Accessed: 2018-12-22.

JSEcoin (2017). Jsecoin: Digital currency - designed for the web. https://jsecoin.com/. Accessed: 2018-12-09.

Kim, K. and Moon, B.-R. (2010). Malware detection based on dependency graph using hybrid genetic algorithm. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 1211–1218. ACM.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Konoth, R. K., van Wegberg, R., Moonsamy, V., and Bos, H. (2019). Malicious cryptocurrency miners: Status and outlook. *arXiv preprint arXiv:1901.10794*.

Krishnan, H. R., Saketh, S. Y., and Vaibhav, V. T. M. (2015). Cryptocurrency mining-transition to cloud. *International Journal of Advanced Computer Science and Applications*, 6(9):115–124.

Lyda, R. and Hamrock, J. (2007). Using entropy analysis to find encrypted and packed malware. *IEEE Security & Privacy*, 5(2).

MalShare (2013). Malware repository providing researchers access to samples. https://malshare.com/. Accessed: 2018-12-22.

Metz, C. E. (1978). Basic principles of roc analysis. In *Seminars in nuclear medicine*, volume 8, pages 283–298. Elsevier.

Microsoft (1991). Peering inside the pe: A tour of the win32 portable executable file format. https://msdn.microsoft.com/en-us/library/ms809762.aspx. Accessed: 2018-12-09.

Moser, A., Kruegel, C., and Kirda, E. (2007). Limits of static analysis for malware detection. In *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual*, pages 421–430. IEEE.

Musavi, M. T., Ahmed, W., Chan, K. H., Faris, K. B., and Hummels, D. M. (1992). On the training of radial basis function classifiers. *Neural networks*, 5(4):595–603.

Ninite (2009). Install or update multiple apps at once. https://ninite.com/. Accessed: 2018-12-22.

Pal, M. (2005). Random forest classifier for remote sensing classification. *International Journal of Remote Sensing*, 26(1):217–222.

Pastrana, S. and Suarez-Tangil, G. (2019). A first look at the crypto-mining malware ecosystem: A decade of unrestricted wealth. *arXiv preprint arXiv:1901.00846*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Rauchberger, J., Schrittwieser, S., Dam, T., Luh, R., Buhov, D., Pötzelsberger, G., and Kim, H. (2018). 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*, page 18. ACM.

Riedmiller, M. (1994). Advanced supervised learning in multi-layer perceptrons—from backpropagation to adaptive learning algorithms. *Computer Standards & Interfaces*, 16(3):265–278.

Roundy, K. A. and Miller, B. P. (2010). Hybrid analysis and control of malware. In *International Workshop on Recent Advances in Intrusion Detection*, pages 317–338. Springer.

Saad, M., Khormali, A., and Mohaisen, A. (2018). End-to-end analysis of in-browser cryptojacking. *arXiv preprint arXiv:1809.02152*.

Sari, A. and Kilic, S. (2017). Exploiting cryptocurrency miners with oisnt techniques. *Transactions on Networks and Communications*, 5(6):62.

Swan, M. (2015). *Blockchain: Blueprint for a new economy.* " O'Reilly Media, Inc.".

Tahir, R., Huzaifa, M., Das, A., Ahmad, M., Gunter, C., Zaffar, F., Caesar, M., and Borisov, N. (2017). Mining on someone else's dime: Mitigating covert mining operations in clouds and enterprises. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 287–310. Springer.

Wang, L. (2005). *Support vector machines: theory and applications*, volume 177. Springer Science & Business Media.

Xu, B., Wang, N., Chen, T., and Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*.

Yang, R. R., Kang, V., Albouq, S., and Zohdy, M. A. (2015). Application of hybrid machine learning to detect and remove malware. *Transactions on Machine Learning and Artificial Intelligence*, 3(4):16.

Zabidi, M. N. A., Maarof, M. A., and Zainal, A. (2012). Malware analysis with multiple features. In *Computer Modelling and Simulation (UKSim), 2012 UKSim 14th International Conference on*, pages 231–235. IEEE.