Analyzing and Detecting In-browser Cryptojacking

Muhammad Saad and David Mohaisen *Senior Member, IEEE.*

Abstract—Cryptojacking is the permissionless use of a target device to covertly mine cryptocurrencies. With cryptojacking, attackers use malicious JavaScript codes to force web browsers into solving proof-of-work puzzles, thus making money by exploiting the resources of the website visitors. We systematically analyze the static, dynamic, and economic aspects of in-browser cryptojacking to understand and counter such attacks. For static analysis, we perform currency-based and code-based categorization of cryptojacking samples to 1) measure their distribution across websites, 2) highlight their platform affinities, and 3) study their code complexities. We apply machine learning techniques to distinguish cryptojacking scripts from benign and malicious JavaScript samples with 100% accuracy. For dynamic analysis, we analyze the effect of cryptojacking on critical system resources, such as CPU and battery usage. We also perform web browser fingerprinting to analyze the information exchange between the victim node and the dropzone cryptojacking server. We also build an analytical model to empirically evaluate the feasibility of cryptojacking as an alternative to online advertisement. Our results show a sizeable negative profit and loss gap, indicating that the model is economically infeasible. Finally, leveraging insights from our analyses, we build countermeasures for in-browser cryptojacking that improve the existing remedies.

Index Terms—Cryptojacking; Coinhive; Illegal Mining

1 Introduction

Blockchain-based cryptocurrencies have emerged as an innovation in distributed systems, enabling a transparent and distributed storage of transactions. Various proof mechanisms, such as the Proof-of-Work (PoW), prevent abuse and improve cryptocurrency trustworthiness. In Bitcoin, for example, individual miners mine new coins through extensive hash operations, which are then verified by distributed nodes in a peer-to-peer (P2P) network [2], [3]. However, PoW led to abuse: an adversary may employ various techniques to abuse public resources for mining purposes and to perform extensive hash calculations at no or low cost.

Cryptojacking is the use of resources of a target host to compute hashes and make a profit out of mining without the consent of the target's owner. Conventional cryptojacking involved the installation of a software binary on a target host that secretly solved PoW and communicated the results to a remote server [4]. Such conventional cryptojacking required user permission to download the software and a persistent Internet connection to communicate the PoW result to the adversary or a *dropzone* server controlled by him. However, conventional cryptojacking has several limitations. First, not all devices have a persistent Internet connection to send PoW results. If not sent immediately after being solved, PoWs become easily outdated. Secondly, antivirus companies can easily identify binaries used for cryptojacking and detect them [5]. Finally, this attack requires an infection vector, whereby users would enable the attack by mistakenly installing the cryptojacking binaries.

A recent form of in-browser cryptojacking that does not suffer from those issues has emerged. In-browser cryptojacking does not require installing binaries or authorization from users to operate. In-browser cryptojacking instances use *JavaScript* code to compute PoW in the web browser and transmit the PoW to a remote server [6]. Since the cryptojacking activity is shielded in the browser's process, it is often not detected by antivirus scanners. Moreover, mining during browsing ensures uninterrupted transmission of PoW over a persistent Internet connection.

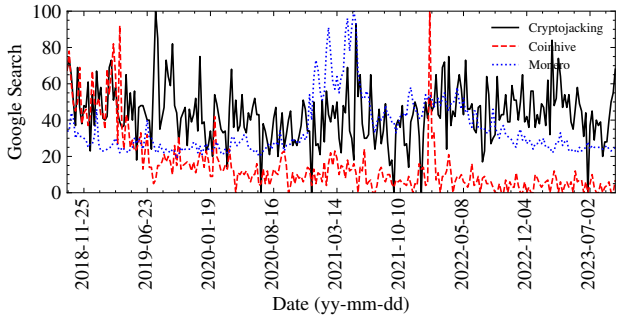


Figure 1. Google search trends for Cryptojacking, Monero, and *Coinhive* from 2018 to 2023. The results have been normalized in this plot's range of [0-100]. The trends show that Cryptojacking and Coinhive are still actively searched terms on Google.

Initially. cryptojacking was intended for good use as an alternative revenue source to online advertisement [7]. Cryptojacking was made easy by online services such as *Coinhive* [8], which provided *JavaScript* templates for cryptojacking. *Coinhive* provided scripts to mine Monero cryptocurrency and the mining rewards were distributed proportionally to the hashes contributed by the miners. cryptojacking has been a major concern to users, as evidenced by the search engine trends showcasing growing interest in the subject. Figure 1, shows the Google search trend for

M. Saad, and D. Mohaisen are with PayPal and the University of Central Florida. M.Saad is the corresponding author. E-mail: muhsaad@paypal.com.
An earlier version of this work has appeared in APWG Symposium on Electronic Crime Research [1]

"Cryptojacking", "Monero", and "Coinhive" from 2018 to 2023. It can be observed that all three terms were actively searched until early 2019. Later, there was a decrease in the search for Coinhive, while Monero and Cryptojacking remained popular search terms.

In-browser cryptojacking serves as an attack avenue for hackers who inject malicious *JavaScript* scripts into popular websites without the knowledge of website owners and mine cryptocurrency for themselves. According to Symantec's Security Threat Report (ISTR), cryptojacking attacks on websites rose by 8500% during 2017 [9], [10]. In February 2018, a major cryptojacking attack hit more than 4000 websites worldwide, including the websites of the US Federal Judiciary and the UK National Health Service (NHS) [11]. Also, in February 2018, Tesla became the victim of a cryptojacking attack in which attackers hijacked Tesla cloud and deployed their cryptojacking code [12]. After such unusual incidents, the UK's National Cyber Security Centre (NCSC) indicated cryptojacking as a "significant threat" in its latest cyber security report [13], [14].

The use of cryptojacking as a replacement for advertisement also has witnessed a great debate. For example, some popular websites, such as "The Pirate Bay", started using cryptojacking as a revenue substitute to online advertisement [15]. "The Pirate Bay" website later disclosed to its users that it would use the CPU cycles of the visitors in exchange for ad-free web browsing, garnering users' approval. As some other websites started using cryptojacking as a revenue generation mechanism, a debate was sparked on the ethics of using cryptojacking [16] and the absence of user consent. Furthermore, it was observed that the continuous CPU-intensive mining, especially on battery-powered devices, resulted in the quick drainage of those devices, adding a new variable to the debate of whether cryptojacking is a good alternative to online advertising.

Motivated by these events, we conduct an in-depth study on in-browser cryptojacking and its effects on website visitors and their devices. We start by analyzing and characterizing more than 5,700 websites with cryptojacking scripts. We then explore static and dynamic analysis tools to understand the behavioral traits of in-browser cryptojacking scripts toward their detection. Using various features extracted through this analysis, we build a classifier for detecting cryptojacking scripts among benign scripts, as well as other malicious types of *JavaScript* codes. We also measure the impact of in-browser cryptojacking on user devices in terms of CPU usage and battery drainage. Finally, in examining the feasibility of cryptojacking as an alternative to online advertisement, we conduct an in-depth end-to-end analysis that considers the implications of such an alternative on both users and websites.

Contributions and Roadmap. In summary, our work explores in-browser cryptojacking by performing static, dynamic, and economic analysis. Our findings are summarized below as key contributions.

1) We collect a dataset of over 5,700 cryptojacking websites and analyze their distribution across the top-level domains (§3).
2) In static analysis, we analyze the code features of cryptojacking scripts to study the distribution of cryptocurrencies used by websites (§4) and develop models for cryptojacking detection. We apply supervised learning techniques for cryptojacking detection and our models achieve a high detection accuracy (§4.3).
3) In dynamic analysis, we analyze how cryptojacking affects the user devices especially their CPU usage and battery drainage (§5). Our dynamic analysis shows that cryptojacking is highly resource intensive and cryptojacking scripts use WebSockets as their communication channel. We later use the WebSocket payload to propose cryptojacking countermeasures (§7).
4) We examine the economic arguments for cryptojacking as an alternative to online advertisement and build a model to estimate the cost of cryptojacking to the users and the gain to websites conducting cryptojacking (§6). We show the economic model is impractical for benign use, and unprofitable for malicious use.

Atop of the contributions in [1], this paper extends our analysis by: 1) a more systematic and in-depth background about various aspects of cryptojacking, including its prevalence, popularity, and association with blockchain-based cryptocurrencies, 2) adapting a supervised learning approach in which we used logistic regression, linear discriminant analysis, k-nearest neighbors, support vector machine, and random forest to improve the detection accuracy of cryptojacking codes (at the website level), and 3) analyzing the memory footprints of in-browser cryptojacking.

To show that cryptojacking is still relevant, we identify 620 out of the 5703 websites that are still online and revise our results in §3 and §4. Our new dataset and recent reports [17] show that cryptojacking is still a problem.

The rest of the paper includes a background in §2, the related work in §8, and conclusion in §9, respectively.

2 Background

2.1 Blockchain-based Cryptocurrencies

In 2009, the first blockchain-based digital currency "Bitcoin" was introduced by Satoshi Nakamoto [18] that involved the exchange of transactions without using a central authority. In Bitcoin, the role of the trusted central authority was replaced by a transparent and tamper-proof public blockchain that acted as a public ledger to maintain the records of transactions. The consensus in the decentralized peer-to-peer Bitcoin network was augmented by a cryptographically secure algorithm known as the proof-of-work (PoW). Bitcoin remained the only cryptocurrency for two years after which several more digital currencies joined the market. Some other notable cryptocurrencies that use the public blockchain include Ethereum, Litecoin, Ripple, Monero, and Dash.

2.2 Mining in Cryptocurrencies

The key operations in every cryptocurrency involve the exchange of transactions among peers, the mining of transactions in blocks, and the publishing of blocks containing those transactions. Computing a valid block results in the generation of new coins in the system.

However, computing a valid block is a non-trivial process in which miners must solve mathematical challenges

and provide a PoW for their solutions. In Bitcoin, PoW involves finding a *nonce* that, when hashed with the data in the block, produces a hash value less than the target threshold the system sets. The target is a function of network difficulty and is denoted by a 256-bit unsigned integer encoded in a 32-bit "compact" form and stored in the block header. In solving the challenge, miners spend effort and, in return, get rewarded with new coins for each valid PoW. As more miners join the network, the hash power of the network and the probability of computing a block increase. The network's difficulty is adjusted every two weeks to keep the average block computation time within the fixed range.

We show how the block computation time, $T(B)$, is affected by the hashing rate, H_r, the *target*, $Target$, the probability of finding a block, $P_r(B)$, and the average number of hashes required to solve the target, H. To keep $T(B)$ in a fixed range (10 minutes), as the H_r increases, the target value is adjusted to keep $P_r(B)$ constant. As such, we calculate $P_r(B) = Target/2^{256}$, $H = 1/P_r(B)$, and $T(B) = H/H_r = 1/(P_r(B) \times H_r)$.

2.3 Cryptojacking

Generally, attackers utilize two main strategies for unauthorized use of a victim's machine to mine digital currencies through cryptojacking: installing a binary on the machine or using an in-browser script. The first one loads the mining code on the victim's machine as a stand-alone binary (or an infection of a binary). As such, it requires information about the target machine, including its operating system and hardware constructs. For example, a malicious cryptojacking binary developed for Windows cannot be executed on Linux. However, the second strategy is platform agnostic. The cryptojacking *JavaScript* is executed upon loading the website in the victim's browser. In both cases, the mining code works in the background. Below, we briefly discuss the two cryptojacking strategies. However, the main focus of this paper is in-browser cryptojacking, which we will discuss in the rest of the paper.

Software-based Cryptojacking. Software-based cryptojacking involves installing a compromised binary on the target host that sends PoW solutions to a *dropzone* server. The most popular cryptojacking software is XMRig [19], a cross-platform mining software supporting four different PoW protocols. Typically, mining pools legitimately use XMRig. However, its malware versions are also available and target non-miners. An XMRig-based malware called "WaterMiner" targets the online gaming community [19]).

In-Browser Cryptojacking. In-browser cryptojacking is done by injecting a *JavaScript* code in a website, allowing it to hijack the processing power of a visitor's device to mine a specific cryptocurrency. The precise nature of the cryptocurrency (*i.e.,* mining protocol, difficulty, message exchange, etc.) is specified by the mining script embedded within a website. Upon visiting a website with cryptojacking code, the browser loads the web page and executes the *JavaScript* snippet that contains instructions for mining and data transfer. As a result, the visiting host starts the mining activity by becoming part of a cryptojacking mining pool. A key feature of in-browser cryptojacking is being platform-independent: it can be run on any host, PC, mobile phone, tablet, etc., as long as the web browser supports *JavaScript*.

2.3.1 Cryptojacking as a Replacement to Advertisement

An ongoing debate sparked in the community for whether cryptojacking can serve as a replacement for online advertisement. Those advocating the approach have pointed out that users providing their CPU power to a website for mining can use the website without viewing online advertisements. Towards that, some websites, including the aforementioned 'The Pirate Bay", started using cryptojacking as a revenue substitute for online advertisements [15] and become "ads-free operation". However, a counterargument to this model is the excessive abuse of the cryptojacking website to the visitor's CPU resources. In-browser cryptojacking scripts will not only run in the background without the user's consent. Still, they will also drain batteries in battery-powered platforms, indirectly affecting the user experience by locking the CPU power and not allowing him to use other applications.

3 Dataset and Preliminary Analysis

3.1 Data Collection

We assembled a data set of cryptojacking websites published by Pixalate [20] and Netlab 360 [21]. Pixalate is a network analytics company that provides data solutions for digital advertising. In Nov. 2017, they collected a list of 5,000 cryptojacking websites actively stealing visitors processing power to mine cryptocurrency. We obtained a list of cryptojacking websites from Pixalate. Netlab 360 (Network Security Research Lab at 360) is a data research platform that provides many datasets. From Netlab 360, we obtained 700 cryptojacking websites, released on Feb 24, 2018.

The combined dataset's top-level domain (TLD) distribution, including the TLD type and the corresponding percentage, is shown in Table 1. Unsurprisingly, .com and .net occupy the first and second spots of the top 10 TLDs represented in the dataset, with a combined total of 40.3% of the websites belonging to them. Country-level domains have a significant presence, with countries such as Slovenia, Russia, and Brazil well represented in the dataset. New-gTLDs were also present in the top-10 gTLDs, with .site having ≈2.0% of the sites. In Pixalate's dataset, six websites were found in the Alexa top 5000 websites, and 13 were among the Alexa top 10000 websites. Among the cryptojacking site, 68.3% did not have a privacy policy.

In contrast, 56.8% of websites had no "terms and conditions" statement, and 49.3% did not have both a privacy policy and terms and conditions. This indicates that the majority of those websites could not formally, through those statements, inform their visitors of the usage of their resources for mining cryptocurrencies, where cryptojacking is used instead of online advertisement [22].

As mentioned in the §1, among 5703 websites analyzed in January 2018, we found 620 websites were online as of September 2023. We use those 620 websites to faithfully reproduce the behavior of cryptojacking websites and extract their features in one of our machine learning experiments in §4.3. We also revise our TLD distribution analysis in Table 1 and report our new findings in Table 2. Our results

Table 1
Distribution of cryptojacking websites with respect to top-level domains in our dataset (type: generic, country, and new).

Rank	TLD	Type	Sites	%
1	.com	g	1945	34.1
2	.net	g	359	6.2
3	.si	c	358	6.2
4	.online	g	349	6.1
5	.ru	c	242	4.2
6	.org	g	191	3.3
7	.sk	c	169	2.9
8	.info	g	169	2.9
9	.br	c	157	2.7
10	.site	n	116	2.0
11	others	—	1648	28.8
Total	—	—	**5703**	**100**

Table 2
Distribution of currently active cryptojacking websites with respect to top-level domains (type: generic, country, and new).

Rank	TLD	Type	Sites	Sites%
1	.com	g	331	53.4
2	.net	g	55	8.9
3	.org	g	27	4.4
4	.ru	c	27	4.4
5	.tv	g	11	1.8
6	.info	g	10	1.6
7	.co	c	10	1.6
8	.me	c	9	1.5
9	.sk	c	9	1.5
10	.de	c	9	1.5
11	others	—	122	20
Total	—	—	**620**	**100%**

Table 3
Detailed results of the currency-based analysis. [1] The variable name is abbreviated. No CJ: No cryptojacking.

Platform	Websites #	Websites %	Cryptocurrency	Websites #	Websites %
Coinhive	4652	81.57			
Hashing	67	1.17			
deepMiner	56	0.98			
Freecontent	39	0.68	Monero	4926	86.37
Cryptoloot	38	0.67			
Miner	38	0.67			
Authedmine	35	0.61			
JSEcoin	149	2.61	JSEcoin	149	2.61
No CJ	628	11.01	—	628	11.01
Total	5703	100.00	—	5703	100.00

show that among the active sites, .com is still the most dominant TLD, and country-level domains are now more prevalent in the dataset.

3.2 Methodology

We perform static and dynamic analysis of the cryptojacking *JavaScript* code. In the static analysis, we categorize the websites based on the currency they mine during cryptojacking. Additionally, we extract the cryptojacking code and develop code-based features to examine their properties. Using those static properties, we compare them with malicious and benign *JavaScript* code. We use standard code analyzers to extract program-specific features.

In our dynamic analysis, we explore the CPU power consumed by cryptojacking websites and its effects on user devices. We run test websites to mimic cryptojacking websites and carry out a series of experiments to validate our hypothesis. For our experiments, we use Selenium-based scripts to automate browsers and various end host devices, including Windows, Linux, and Mac-operated laptops, to monitor the effect of cryptojacking under various operating systems and hardware architectures. For website information, we use services provided by Alexa and SimilarWeb to extract information regarding website ranking, the volume of traffic, and the average time visitors spend on that website [23].

4 Static Analysis

For static analysis, we perform currency-based and code-based analysis. In the currency-based categorization, we show the distribution of service providers and platforms providing cryptojacking templates for those websites. The code-based analysis provides insight into the complexity of the cryptojacking scripts using various code complexity measures. Using those features, we perform two experiments for cryptojacking detection. Our first experiment is a website-agnostic approach to uniquely distinguish cryptojacking *JavaScript* from other forms of malicious and benign *JavaScript* codes. Our second experiment is a website-specific approach using which we analyze 620 cryptojacking websites that are still online and compare them with non-cryptojacking websites.

4.1 Currency-based Categorization

To understand the cryptojacking ecosystem, it is critical to find out what cryptocurrencies are typically being mined through in-browser cryptojacking. Therefore, we inspected the websites' scripts to extract information about the platforms and cryptocurrencies. From our dataset, we found that there were eight platforms providing templates to mine two types of cryptocurrencies, namely, Monero and JSEcoin. In Table 3, we provide details about the eight platforms and their respective cryptocurrency. We found that a large proportion of the websites (≈81.57%) use *Coinhive* [8] platform to mine Monero cryptocurrency [24], which is one of the few cryptocurrencies that supports in-browser mining. We found that ≈86.37% of the websites in our dataset are mining Monero cryptocurrency through seven platforms. In addition, ≈2.61% of the websites are using the JSEcoin platform [25], which is responsible for mining the JSEcoin cryptocurrency.

Although PoW-based cryptocurrencies have many traits in common, they may vary in market cap, user base, application protocols, and mining rewards. In our dataset, we found two cryptocurrencies, namely Monero and JSEcoin, which are used for in-browser cryptojacking. We report the differences between the two cryptocurrencies in Table 4. While both of them are used for cryptojacking, at the time of writing this paper, JSEcoin was not launched in the market and did not have any "Initial Coin Offering" (ICO), which explains its low prevalence in our dataset. Furthermore, unlike Monero, which is resource-intensive, JSEcoin uses minimal CPU power and does not add a significant processing overhead to the target device. One of the key objectives of this paper is to characterize resource abuse in cryptocurrency mining, where Monero is shown to be a better example than the "browser-friendly" JSEcoin. Therefore, due to its high prevalence in the dataset, and the significant contribution towards the broader goal of this study, we mainly focus our work on Monero cryptocurrency.

4.2 Code-based Analysis

We perform static analysis on the cryptojacking scripts to analyze the performance and complexity of their code. Static analysis reveals code-specific features for insights into the flow of information upon code execution. For static analysis,

Table 4
Comparison of Moneroe and JSEcoin. JSEcoin has not been released in the market as yet.

Currency	Market Cap	Consensus Algorithm	Resource Intensive	Dataset Prevelance
Monero	2.3B	CryptoNight	✓	86.37%
JSEcoin	—	SHA-256	✗	2.61%

we gathered cryptojacking scripts from all major cryptojacking service providers in our dataset: *Coinhive*, JSEcoin, Crypto-Loot, Hashing, deepMiner, Freecontent, Miner, and Authedmine. We observed that all the service providers had unique codes, specific to their own platforms. In other words, the websites using *Coinhive* or JSEcoin employed their respective *JavaScript* templates. However, each provider's code template differed, which led us to believe that each script had unique static features. With all of that in mind, we performed static analysis on the cryptojacking websites and compared the results with another standard *JavaScript* for a baseline comparison.

We prepared our dataset for static analysis by collecting all of the popular cryptojacking scripts from our list of websites. We found eight unique scripts in our dataset, each belonging to one service provider. As a control experiment, we collected an equal number of malicious and benign *JavaScript* codes to design machine learning models for detection. We aimed to obtain features unique only to the cryptojacking scripts and aid in their detection. We were limited to including equal sizes of malicious and benign *JavaScript* samples for the static analysis to avoid bias towards a certain class. Although there are many samples of malicious and benign *JavaScript* in the wild, only eight cryptojacking scripts are available. Since our work is focused on distinguishing cryptojacking scripts from malicious and benign *JavaScript*, we had to balance the size of each class. While the number of scripts might seem a limitation of our work, we believe the promise of this work is substantial. As more platforms use cryptojacking, more samples will be available for a broader study.

In lieu, we used the existing data of the cryptojacking websites (§3.1) and online resources from GitHub for malicious *JavaScript* sample [26]. For benign *JavaScript*, we used the set of non-cryptojacking websites and parsed their HTML code to extract benign *JavaScript* code [27]. In summary, we had 8 samples of cryptojacking *JavaScript*, spanning all the websites. Accordingly, we selected 10 malicious and 10 benign scripts for our machine-learning model and extracted the following features for static analysis.

Cyclomatic Complexity. Cyclomatic complexity measures the complexity of code using a control flow graph (CFG), where each node represents a function and a directed edge between two nodes indicates a caller-callee relationship. Let E be the number of edges, N be the number of nodes, and Q be the number of connected components in the CFG, M can be used to denote the cyclomatic complexity of the program and is calculated as $M = E + 2Q - N$.

Cyclomatic Complexity Density. Cyclomatic complexity density [28] measures Cyclomatic complexity, defined above, spread over the total code length. Let c_l be the total number of lines of code, then the cyclomatic complexity density, denoted by M_d, can be computed as $M_d = \frac{E+2Q-N}{c_l}$

Halstead Complexity Measures. The Halstead complexity measures are used as metrics to characterize the algorithmic implementation of a programming language. Those measures include the vocabulary η, the program length n, the calculated program length n_c, the volume V, the effort E, the delivered bugs B, the time T, and the difficulty D. Let the number of distinct operators be η_1, the number of distinct operands be η_2, the total number of operators be n_1, the total number of operands be n_2, the η, n, n_l, V, E, and B are defined as follows:

$$\eta = \eta_1 + \eta_2, \qquad n = n_1 + n_2 \tag{1}$$

$$n_c = (\eta_1 \log_2 \eta_1) + (\eta_2 \log_2 \eta_2), \qquad V = n \times \log_2 \eta \tag{2}$$

$$D = (\eta_1/2) \times (n_2/\eta_2), \qquad E = D \times V \tag{3}$$

$$T = (D \times V)/18, \qquad B = E^{\frac{2}{3}}/3000 \tag{4}$$

Maintainability Score. The maintainability score M_s is calculated using Halstead volume V, cyclomatic complexity M, and the total lines of code in the *JavaScript* file c_l. The maintainability score index M_i is calculated between [0-100] and is defined as $M_s = 171 - 5.2\log(V) - 0.23M - 16.2\log(c_l)$; $M_i = \max(0, \frac{M_s}{171})$.

Source Lines of Code. Source lines of code (SLOC) measure the lines of code in the program after excluding the white spaces. SLOC is a predictive parameter to evaluate the effort required to execute the program. It also provides insights into program maintainability and productivity.

Results. To extract features in our code-based analysis, we used *Plato*, a *JavaScript* static analysis and source code complexity tool [29]. For each *JavaScript* code, we ran *Plato* and recorded the 17 extracted features as reported in Table 5. From Table 5, we observed that certain features, such as M, M_d, V, and T, are clearly discriminative among all the categories.

4.3 Classification Models

We applied machine learning techniques in two experiments for cryptojacking detection. In the first experiment, we selected the unique cryptojacking *JavaScript* codes of eight platforms shown in Table 3.[1] Our primary goal was to study the code-based features of cryptojacking scripts that are distinctly different from other types of *JavaScript* codes. Therefore, discriminative features of cryptojacking scripts can be characterized through machine learning models for classification and detection. With only eight cryptojacking scripts available for the experiment, we collected a comparable number of malicious and benign *JavaScript* code samples from Github [26], [27] to avoid bias in model training [31].

After collecting malicious, benign, and cryptojacking*JavaScript* codes, we extracted their code-based features using Plato Table 5 and applied machine learning models including Logistic Regression (LR), Linear Discriminant Analysis (LDA), k-nearest neighbors (k-NN), Support Vector Machines (SVM), and Random Forest (RF) [32]. Logistic Regression applies a logistic function to compute the probability of binary outcomes. Linear Discriminant Analysis is

1. All eight unique cryptojacking platforms can be found in [30]

Table 5
The static features of the cryptojacking, malicious, and benign samples. The mean (μ) and standard deviation (σ) of the features are also reported.

Cat.	Platforms	M	M_d	B	D	E	c_l	T	η	V	η_1	n_1	η_2	n_2	params	sloc	physical	M_s
Cryptojacking	deepMiner	184	44.2	14.1	113.0	4,810,434	4,667	267,246	554	42,533	47	2,440	507	2,227	75	416	499	67.8
	Authedmine	168	26.5	19.7	82.8	4,912,255	6,096	272,903	844	59,259	41	3,247	803	2,849	73	633	784	62.8
	Hashing	138	29.1	7.2	94.6	2,185,379	2,794	124,138	342	24,393	38	1,469	315	1,415	37	412	505	68.2
	Miner	133	27.7	9.3	90.5	2,537,930	3,239	140,996	403	28,032	39	1,690	364	1,549	49	479	617	64.1
	Coinhive	131	27.5	9.1	94.8	2,608,021	3,226	144,890	368	274,970	37	1,697	331	1,529	48	476	594	63.7
	Crypto-loot	128	39.7	11.4	88.1	3,034,935	3,788	168,607	546	34,443	45	1,962	501	1,826	62	322	389	70.3
	Freecontent	117	28.3	8.1	89.4	2,180,394	2,884	121,133	350	24,373	38	1,469	312	1,415	37	412	505	62.7
	JSEcoin	64	17.2	10.2	62.9	1,945,165	3,257	108,064	716	30,888	45	1,878	671	1,379	49	372	412	64.7
	Mean (μ)	130.3	29.9	11.3	88.9	3,026,191	3,755.1	168,121	516.4	33,925	41.3	1,981.5	475.1	1,773.6	53.8	440.3	538.1	64.9
	SD. (σ)	35.9	8.4	3.9	13.8	1,180,403	1,109.9	65,577	185.1	11,856	3.9	599.3	182.8	519.3	14.8	93.2	126.3	2.8
Malicious	20160209	92	21.5	5.6	25.1	423,925	1,833	23,551	580	16,826	27	1,032	553	801	22	427	503	44.4
	20161126	62	15.3	4.2	24.6	315,735	1,563	17,540	292	12,800	17	798	275	765	0	403	481	90.5
	20170110	14	4.4	15.0	26.7	1,211,305	4,704	67,294	782	45,210	15	2,740	767	1,964	232	313	564	93.6
	20170507	6	24.0	5.9	11.1	199,917	1,864	11,106	777	17,897	18	942	759	922	1	25	890	71.7
	20160927	3	1.4	4.0	32.5	393,555	1,575	21,864	204	12,084	13	957	191	618	0	213	98	23.2
	20170322	2	18.1	11.8	7.1	253,442	3,514	14,080	1,123	35,607	9	1,762	1,114	1,752	3	11	1,738	90.9
	20170303	2	8.6	0.2	9.4	8,338	147	463	63	878	13	73	50	74	4	23	55	78.7
	20160407	1	33.3	0.1	2.7	207	19	11	16	76	5	12	11	7	0	3	3	78.9
	20170501	1	0.9	2.1	3.3	21,464	758	1,192	322	6,314	5	431	317	327	0	105	105	35.9
	20160810	1	12.5	0.5	11.9	20,148	275	1,119	70	1,685	6	255	64	20	0	8	13	60.4
	Mean (μ)	18.4	14	4.9	15.5	284,803.7	1,625.2	15,822	422.9	14,938	12.8	900.2	410.1	725	26.2	153.1	445	66.9
	SD. (σ)	31.9	10.5	5	10.8	364,470.8	1,508.9	20,248	374.8	15,045	6.9	834.7	372.5	686.6	72.6	171.9	543.5	24.9
Benign	The Boat	2,135	69.3	110.8	392.0	130,285,522	31,916	7,238,084	1,364	332,361	59	17,341	1,305	14,575	852	3,084	3,349	66.7
	IBM Design	2,119	68.3	110.9	397.1	132,237,213	32,018	7,346,511	1,351	332,981	59	17,393	1,292	1,4625	853	3,103	3,372	66.7
	Histography	1,743	40.7	95.2	249.5	71,325,242	26,627	3,962,513	1,704	285,833	55	14,963	1,649	11,663	803	4,278	5,043	59.4
	Know Lupus	1,006	28.1	92.9	170.4	47,474,425	25,120	2,637,468	2,181	278,600	54	13,424	2,127	11,696	615	3,583	4,288	65.2
	tota11y	815	38.8	59.4	227.7	40,563,065	17,486	2,253,503	1,167	178,157	52	9,764	1,115	7,722	412	2,099	2,336	62.9
	Masi Tupungato	784	58.2	47.1	185.0	26,199,193	14,296	1,455,510	958	141,585	43	7,875	915	6,421	238	1,347	1,470	67.2
	Fillipo	703	42.9	43.1	194.3	25,139,766	12,900	1,396,653	1,045	129,377	54	7,132	991	5,768	269	1,637	1,770	61.5
	Leg Work	412	75.7	34.0	241.3	24,651,056	11,100	1,369,503	589	102,143	45	5,835	544	5,265	66	544	633	65.9
	Code Conf	409	27.8	41.1	197.1	24,336,420	12,500	1,352,023	939	123,437	49	7,162	890	5,338	315	1,469	1,753	64.9
	Louis Browns	368	35.6	21.2	106.7	6,792,400	6,529	377,355	862	63,667	51	3,393	811	3,136	68	1,034	1,357	53.3
	Mean (μ)	1,049.4	48.5	65.6	236.1	52,900,430	19,049.2	2,938,912	1,216	196,814	52.1	10,428.2	1,163.9	8,621	449.1	2,217.8	2,537.1	63.4
	SD. (σ)	694	17.8	33.6	92.8	44,755,377	9,151.2	2,486,409	459.8	100,856	5.3	4,999	456.7	4,165	310.3	1,225.4	1,418.2	4.3

suitable for multivariate data in which it separates classes using linear combinations. k-nearest neighbors is suitable for non-linear problems, and it assigns a data point to the majority class among k-nearest neighbors of the data point. Support Vector Machines is useful for high-dimensional data that separates data points of different classes by constructing a hyperplane. Random Forest combines different decision trees and makes a class prediction based on a majority vote aggregated over individual trees.

We apply these models on our dataset and report each model's precision, recall, and F1 score in Table 6. Our results show that Linear Discriminant Analysis (LDA) and Random Forest performed well, achieving an accuracy of 100% as indicated by the value 1.00 for precision, recall, and F1-score. In contrast, k-NN under-performed with 0.86, 0.75, and 1.00 values for precision, recall, and F1-score. SVM and Logistic Regression performed better than k-NN with 0.92, 0.86, and 1.00 values for precision, recall, and F1-score. As a result, we derive two key conclusions from our experiments. First, the models that expect data linearity between features and the target variable are more helpful in detecting cryptojacking scripts. Second, the *JavaScript* features of the three classes are highly discriminative, indicating unique coding patterns for each category, which are easily distinguishable.

Although our first experiment provided meaningful insights regarding cryptojacking code detection, we suspected that it may not be generalizable due to a smaller sample size used in training. To address this limitation, we conducted a follow-up experiment in which we trained our model on a larger dataset to evaluate the generalizability of our approach in detecting cryptojacking websites from non-cryptojacking websites using *JavaScript* code features.

For sample size enrichment, we used our original dataset of 5,703 and crawled 620 domains that were still online. We observed that those 620 domains were previously using *Coinhive* script for cryptojacking and had discontinued cryptojacking after *Coinhive* shutdown. For each domain, we collected the website code and extracted the same features reported in Table 5 (*i.e.*, cyclomatic complexity, Halstead difficulty, and distinct operands etc.). Since each website was previously using *Coinhive* script, we added the *Coinhive* script code to the downloaded website code (offline) and then extracted the code features using *Plato*. We then randomly selected 620 benign websites from Alexa's top 1 Million domains which were not present in our cryptojacking dataset. As a result, we obtained two classes of 620 samples each, with one class containing features of 620 cryptojacking websites while the other containing features of benign websites. By following this procedure, we achieved two main objectives. First, we increased our sample size to improve model generalizability. Second, despite the closure of *Coinhive*, we successfully reproduced the behavior of cryptojacking websites that could be captured and compared to the behavior of benign websites.

We divided our dataset into 75% training and 25% testing subsets. The results from the second experiment show that almost all classification models achieved an accuracy of 100% as indicated by the value 1.00 for precision, recall, and F1-score in Table 7. Only Random Forest had a low precision of 0.99 compared to other models. Our second experiment validates that the features of cryptojacking scripts are highly discriminative from other code types, which can be easily detected across cryptojacking and non-cryptojacking websites. Moreover, achieving a consistently high detection accuracy at a larger sample size demonstrates the generalizability of our proposed detection methodology.

Key Takeaways. From the static analysis, we derive the following key conclusions: (1) cryptojacking websites use various scripts, platforms, and cryptocurrencies for mining operations, (2) cryptojacking scripts have distinct coding

Table 6
Classification performance (first experiment) against the F1-score, precision and recall.

	F1	Pre	Rec
LR	0.92	0.86	1.00
LDA	1.00	1.00	1.00
k-NN	0.86	0.75	1.00
SVM	0.92	0.86	1.00
RF	1.00	1.00	1.00

Table 7
Classification performance (second experiment) against the F1-score, precision and recall.

	F1	Pre	Rec
LR	1.00	1.00	1.00
LDA	1.00	1.00	1.00
k-NN	1.00	1.00	1.00
SVM	1.00	1.00	1.00
RF	1.00	0.99	1.00

```
<script src="./Welcome_files/coinhive.min.js"></script>
<script>
  var miner = new coinhive.Anonymous("owner key",
      {throttle: 0.1});
    miner.start();
</script>
```

Figure 2. Malicious JavaScript code that links a website to Coinhive.

patterns that can be accurately modeled using machine learning, and (3) our machine learning models efficiently detect cryptojacking with a precision and recall of 1.

5 Dynamic Analysis

5.1 Resource Consumption Profiling

Settings and Measurements Environment. We noticed that in each cryptojacking website, a *JavaScript* snippet encodes a key belonging to the code owner and a link to a server to which the PoW is sent. Figure 2 provides a script found in websites that use *Coinhive* for mining. The source (*src*) refers to the actual *JavaScript* file that is executed after a browser loads the script. In this script, we also noticed a *throttling parameter*, which controls how much resources a cryptojacking script uses on the host. We use the throttling parameter, α, as an additional variable in our experiment. We experiment with $\alpha = \{0.1, 0.5, 0.9\}$.

To understand the impact of cryptojacking on resource usage in different platforms, we use battery-powered machines running Microsoft Windows, Linux, and Mac operating systems (OSes). We selected three laptops, each with one of those OSes. Using the above parameters, we set up an account on *Coinhive* for a key that links our "experiment website" to the server. We embedded the code in Figure 2 in the website's HTML tags. To measure the usage of resources while running cryptojacking websites, we set up a Selenium-based web browser automation and ran cryptojacking websites for various evaluations. Selenium is a portable web-testing software mimicking the behavior of the actual web browsers [33].

CPU Usage. To understand the CPU usage during cryptojacking, we conducted measurements on our devices with cryptojacking code installed on the experiment website. We modulated the throttling parameter between $\alpha = \{0.1, 0.5, 0.9\}$. We found that cryptojacking excessively uses the CPU cycles for mining operations. Moreover, there is an inverse relationship between the throttling parameter (α) value and the consumption of the CPU cycles. The lower value of α resulted in the higher consumption of CPU cycles across all testing machines, as shown in Figure 3.

Battery Usage. High CPU usage translates to higher power consumption and quicker battery drainage. To investigate how cryptojacking affects battery drainage, we conducted several experiments using various α values for each device. For each $\alpha \in \{0.1, 0.5, 0.9\}$, we ran the *JavaScript* script on a fully charged battery. We logged the battery level every 30 seconds, as the script ran on each device with the given α value, starting from a fully charged battery. Finally, we measured the baseline by running our script without the cryptojacking code. The results are shown in Figure 4. As expected, with $\alpha = 0.1$, corresponding to the lowest throttling and highest CPU usage, the battery drained quickly to ≈10% of its capacity within 80 minutes, compared to ≈85% within the same time when not using cryptojacking (Figure 4(a)). The battery usage pattern was relatively consistent across all three devices.

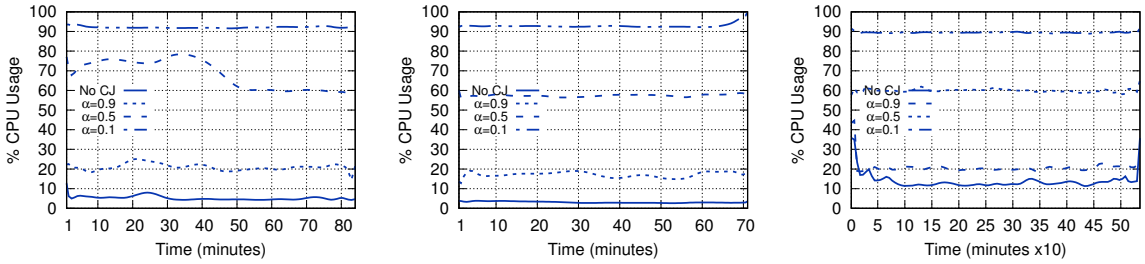
Memory Usage. In addition to analyzing CPU and battery usage, we also investigated the effect of cryptojacking on the memory usage of Windows, Linux, and Mac. We report results in Figure 5. Our results show that cryptojacking has no significant relationship with the use of memory since memory usage was random for all experiments. For Windows, with no cryptojacking, the memory usage was ≈3.5GB. For $\alpha = 0.1, 0.5,$ and 0.9, the memory usage was ≈3.1, 3.9, and 3.9GB, respectively. In contrast, for Mac, the memory usage was ≈8.5GB, irrespective of the throttling parameter α. The randomness in results suggests that memory footprint is not a good indicator for cryptojacking detection.

5.2 Network Usage and Profiling

Dynamic artifacts are essential to analyze cryptojacking scripts, especially when scripts are obfuscated. To this end, we also explore the network-level artifacts to uncover the operations of cryptojacking services.

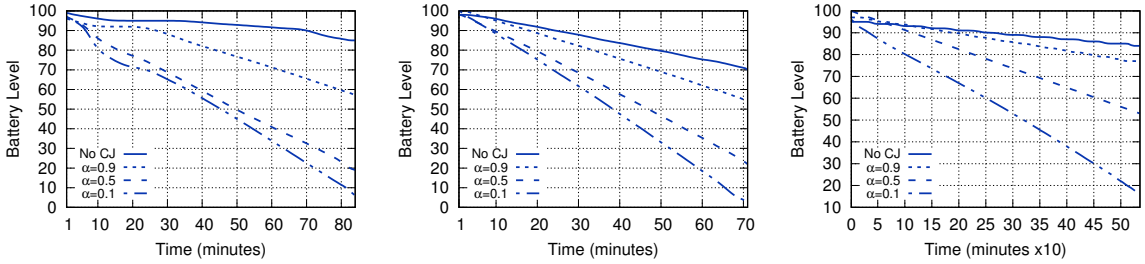
We noticed that during cryptojacking website execution, the *JavaScript* code establishes a WebSocket connection with a remote server and performs a bidirectional data transfer. The WebSocket communication can be monitored using traffic analyzers such as *Wireshark*. However, a major issue when using traffic analyzers is that browsers encrypt the web traffic during WebSocket communication. Although significant information, such as source, destination, payload size, and request timings, can still be gathered, the transferred data remains encrypted, preventing further analysis. To perform a deeper analysis of WebSocket traffic, we examined the actual data frames *in the browser* to understand the communication protocol and payload content of WebSocket connection for possible analysis of cryptojacking websites, outlined below.

When a WebSocket request is initiated, the client sends an *auth* message to the server along with the user information, including *sitekey*, *type*, and *user*. The length of *auth* message is 112 bytes. The *sitekey* parameter is used by the server to identify the user who owns the key of the *JavaScript* and adds a balance of hashes to the user's account. The server then authenticates the request parameters and responds with *authed* message. The *authed* message length is 50 Bytes and it includes a token and the total number of hashes received from the client's machine. The server then sends *job* message to the client. The *job* message has a length



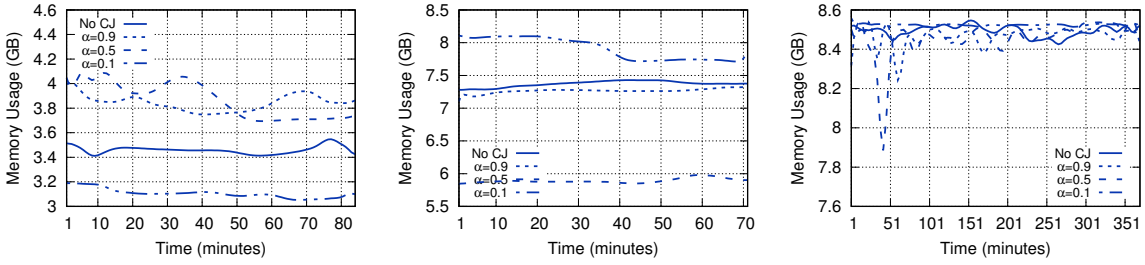
(a) CPU usage on Windows (b) CPU usage on Linux (c) CPU usage on Mac

Figure 3. CPU usage recorded on three devices. Note that decreasing the value of α increased the CPU consumption across all three devices.



(a) Battery consumption on Windows (b) Battery consumption on Linux (c) Battery consumption on Mac

Figure 4. Battery usage recorded on three devices used in the dynamic analysis. We observe that decreasing the value of α increases the CPU usage and battery drainage across all devices.



(a) Memory usage on Windows (b) Memory usage on Linux (c) Memory usage on Mac

Figure 5. Memory usage recorded on three devices. Our results indicate that cryptojacking has no relationship with memory usage. A change in α does not reflect any predictable change in memory usage on any device. Please note that the plot legends are consistent across all figures, with Figure 5(c) providing a clear legend for reference.

of 234 Bytes with a *job_id*, *blob*, and *target*. The *target* is a function of the current difficulty in the cryptocurrency to be mined. The client then computes hashes on the *nonce* and sends a *submit* message back to the server, with *job_id*, *nonce*, and the resulting hash. The *submit* message has a payload length of 156 Bytes. In response to the *submit* message, the server sends *hash_accept* message with an acknowledgment and the total number of hashes received during the session. The *hash_accept* message is 48 Bytes long. In Table 8, we provide details about the WebSocket connection during a cryptojacking session.

Key Takeaways. From dynamic analysis, we derive the following key takeaways: (1) cryptojacking is highly resource intensive and it can cause up to 90% CPU usage and battery drainage across all devices, (2) the throttling parameter in the script defines the intensity of resource usage, and (3) cryptojacking websites use WebSockets as their communication channel with the dropzone server.

6 Economics of Cryptojacking

In this section, we evaluate the economic feasibility of cryptojacking by extrapolating the results in our dynamic analysis. We look at the economic feasibility from the perspective of a cryptojacking website's owner, intentional cryptojacking, malicious cryptojacking, and website visitors. For cryptojacking, the reward of the website owner or adversary depends on the number of hashes produced when a website visitor visits the website. We formulate the analysis as a feasibility: how much of the energy consumed by cryptojacking scripts (cost) is transferred to the cryptojacking website owner, whether malicious or benign, and how that translates as an alternative to online advertisement.

Table 8
Messages exchanged between the client and the server during WebSocket connection. Length is measured in bytes.

Message	Source	Sink	Length	Parameters
auth	client	server	112	sitekey, type, user
authed	server	client	50	token, hashes
job	server	client	234	job_id, blob, target
submit	client	server	156	job_id, result
hash_accept	server	client	48	hashes

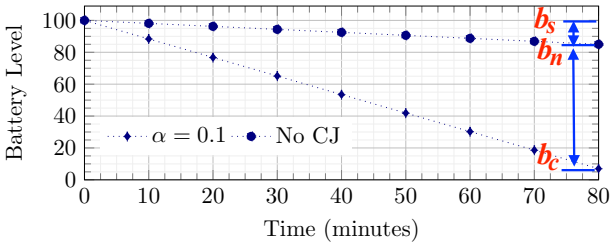


Figure 6. Battery drain in Windows i7. b_s denotes the starting point of the battery, b_n the normal 80 minutes battery drainage without cryptojacking and b_c denotes the battery drainage with maximum cryptojacking.

6.1 Analytical Model

To set a stage for our analysis, in Figure 6, we present the results from one sample experiment conducted on Windows i7 machine with a cryptojacking website set to minimum throttling (α=0.1), indicating a maximum cryptojacking. In this figure, the region between b_s and b_n is a baseline unrelated to cryptojacking–due to the system's normal operation. On the other hand, the region between b_n and b_c is the battery drainage due to cryptojacking. We refer to the energy loss due to such cryptojacking as L for a given user. To formulate the cost (to users) and benefit (to cryptojacking website), let P be the benefit (profit) during a cryptojacking session of Δt minutes, and h be the hash rate of the device in hashes/second. At the time of writing this paper, *Coinhive* pays $2,894 \times 10^{-8}$ (XMR; currency unit) for 1 million hashes, where 1 XMR equals 200 USD. Therefore, the profit P in XMR in $\Delta t = t_f - t_s$ (t_f and t_s refer to the finish and start time of a session, respectively) can be computed as:

$$P(\text{XMR}) = (2,894 \times 10^{-8} \times h \times \Delta t)/10^6 \quad (5)$$

The average hash rate of our test device was 21 hashes/second. For $\Delta t = 85$ minutes from Figure 6, the profit P earned during the session was 3.19×10^{-6} XMR or \$ 6.38×10^{-4} USD (\$ 1.06×10^{-5} USD/second). This is the upper bound of profit that the device can make in one battery charge.

To calculate L, corresponding to battery drainage due to cryptojacking ($b_n - b_c$), we first measure the time it takes to recharge 1% of the battery and denote it by t_r. Therefore, the time required to recover $b_n - b_c$ can be calculated as $t_r \times (b_n - b_c)$. Let W be the power consumed by the laptop to run for one hour and C be the cost of electricity in USD/KWH. Therefore, the loss L in USD for the use of the battery can be computed using:

$$L(USD) = C \times W \times t_r \times (b_n - b_c) \quad (6)$$

For our test device, we had the following parameters: $W = 65$ watt-hour, $C = 6.418 \times 10^{-5}$ USD/(watt-hour), b_n = 82% (in Figure 6), b_c = 10% and t_r = 0.015 hour. Thus, the estimated loss during cryptojacking session L was $\approx \$4.5 \times 10^{-3}$ USD, which is seven times the value of P, highlighting a big gap cryptojacking's operation model.

Using the same analysis, we examine if users can use cryptojacking as a source of income. With the same device as above, the number of hashes required to make 1 XMR (\$200 USD) is 3.45×10^{10} hashes. Given that the same device generates 21 hashes/second, the time required to make 1 XMR is approximately 52 years, while the energy consumed is many orders of magnitude more costly (note that the calculations here are quite theoretical; to mine 1 XMR, it would take ≈321,543 battery charging cycles, each of which would cost 0.41 cent (total of ≈ 1318).

Table 9
Monthly Profit earned by top websites by applying cryptojacking. GR denotes global rank, CR denotes the country rank, visits are in Billions, average time duration of visits is in mm-ss, P-CJ is the profit earned by cryptojacking, and P-Ads is revenue earned through ads. "—" denotes the revenue of the companies that we could not find online.

Website	GR	CR	Visits	Time	P-CJ	P-Ads
google.com	1	1	47.09	07:23	2.41 M	7.94 B
youtube.com	2	2	26.22	20:05	3.65 M	291 M
baidu.com	3	1	19.08	08:56	1.18 M	234 M
wikipedia.org	4	6	6.55	03:51	0.17 M	160 M
reddit.com	5	4	1.69	10:38	0.12 M	—
facebook.com	6	3	29.87	13:28	2.80 M	3.3 B
yahoo.com	7	7	5.21	06:19	0.22 M	250 M
google.co.in	8	1	5.33	07:46	0.29 M	1.1 B
qq.com	9	2	3.66	04:02	0.10 M	—
taobao.com	10	3	1.73	06:25	0.08 M	—

6.2 Cryptojacking and Online Advertisement

In-browser cryptojacking is being argued as an alternative to online advertisement. To understand the soundness of this argument, we performed an experiment to analyze and compare the monetary value of in-browser cryptojacking as a replacement for online advertisements.

We select Alexa's top 10 websites [34]. For each website, we obtained the average number of visitors and the time they spent on those websites during March 2018. Using that and our model from section 6.1 to measure the potential profit those websites could have made using cryptojacking. We assume that visitors on these websites have an average hash rate of 20 hashes/second. We report the results in Table 9, highlighting that those websites would make between \$3.65 million USD (for *youtube.com*) and \$0.10 million USD (*qq.com*) per month.

Statista [35] publishes annual online advertisement revenue reports. We collect the revenues generated by each of those top-10 websites for 2017. We use those figures to examine the potential of cryptojacking as an advertisement alternative at scale. For that, we first obtain a monthly revenue figure for each website by dividing the annual revenue by 12. We compare those numbers to the cryptojacking alternative highlighted above. The results are shown in Table 9, where it can be seen that the revenue earned by operating cryptojacking is negligible compared to the revenue earned through online advertisements. For example, if Google is to switch to cryptojacking, it will make \$2.41 million USD

Table 10
The estimated monthly earnings. Visits are in millions, the average time of each visit is in mm-ss and the profit (P-CJ) is in USD.

Website	GR	CR	Visits	Time	P-CJ
firefoxchina.cn	1,088	132	87.24	04:32	2,746.9
baytpbportal.fi	1,613	591	12.16	05:36	472.9
mejortorrent.com	1,800	37	22.83	04:50	766.4
moonbit.co.in	2,761	1,289	15.68	28:37	3,116.5
shareae.com	3,331	1,071	5.86	04:49	196.0
maalaimalar.com	4,090	112	3.38	03:26	80.6
icouchtuner.to	6,084	518	7.96	02:98	200.8
paperpk.com	6,794	2,050	3.01	03:23	70.7
scamadviser.com	6,847	668	4.20	02:08	62.2
seriesdanko.to	7,253	1,452	5.44	04:59	188.2

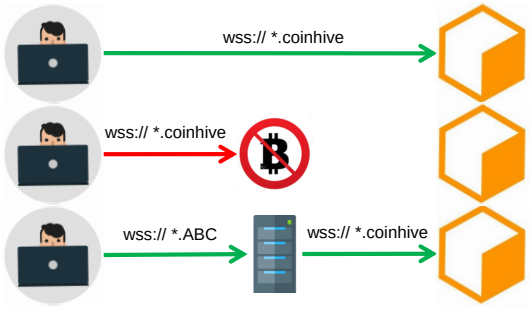


Figure 7. Circumventing cryptojacking detection by relaying WebSocket requests through a third-party proxy server.

per month. In contrast, Google earns ≈$7.94 Billion USD monthly from online advertisement.

To estimate the revenue by cryptojacking websites, we conducted the same experiment on the top-10 websites in our dataset and computed their estimated profit, shown in Table 10. We notice that the maximum profit earned by *firefoxchina* is ≈$2,747 USD. Although the ad revenue for these websites is not available online, we still suspect that $2,747 USD per month is far too low for a website that has 87.24 million monthly views, each with an average duration of 4 minutes and 32 seconds, as compared to the potential revenues for online advertisement. Those findings align with recent reports indicating that an adversary who compromised 5,000 websites and injected his own cryptojacking scripts could only make $24 USD [36].

Key Takeaways. From our economic analysis, we find that in-browser cryptojacking is not a feasible alternative for online advertisement since it generates negligible revenue. Even the most popular online website that generates over $7.9 billion through ads can only make up to $2.41 million through cryptojacking. It is, therefore, plausible to assume that the cryptojacking is unlikely to replace online advertisement as a revenue source for websites.

7 Countermeasures

7.1 Existing Countermeasures

At the browser level, existing countermeasures include web extensions such as No Coin, Anti Miner, and No Mining [37]. Each web extension maintains a list of uniform resource locators (URLs) to block while browsing websites. If a user visits a website that is blocklisted by the extension, the user is notified about cryptojacking. However, we show that blocklisting is ineffective since an adaptive attacker can circumvent detection by creating new links not found in the public list of blocklisted URLs.

We set these extensions up on Chrome and evaluated them. All the extensions detected cryptojacking by reading the WebSocket requests generated by the website to *Coinhive*. However, in the next phase, we removed the binding key of our script shown in Figure 2. Without the key, the website establishes the WebSocket connection but does not perform cryptojacking as it cannot verify itself with the server without the key. However, when we tested that on the extensions, all of them wrongly signaled the presence of active cryptojacking. Since extension-based blocklisting does not read the data frames exchanged between web sockets, even the presence of an outdated key or a broken link is falsely labeled as cryptojacking.

Evading Detection. An attacker, knowing the blocklist, can evade detection by setting his own third-party server to relay data to and from the cryptojacking server. The cryptojacking website can establish a WebSocket connection to a third-party server and send data frames and keys to the server. Since anti-cryptojacking extensions will not have the address of a third-party server blocklisted, they will not be able to prevent the connection and cryptojacking. In Figure 7, we show how an adaptive attacker can circumvent the current countermeasures for cryptojacking. To practically demonstrate that, we set up a test website using *Coinhive* script and installed a local relay server. We installed four Chrome extensions blocking the in-browser cryptojacking: No Coin, Anti Miner, No Mining, and Mining Blocker. We installed the *Coinhive* script in the experiment's first phase and ran the website. Each extension detected the WebSocket request and blocked it. To mimic an adaptive attacker, we configured our relay server to act as a proxy, receive socket requests from the browser, and relay them to *Coinhive* server. We modified the code in the *Coinhive* script and replaced the *Coinhive* socket address with our server address. Next, when we visited the website, it started cryptojacking on the client machine. No extension was detected, showing that it is possible to circumvent the blocklisting technique.

Countering Adaptive Attacker. To counter an adaptive attacker and overcome the limitation of existing countermeasures, a better approach is message-based cryptojacking detection in web extensions. Instead of blocking specific URLs, the extensions can monitor the messages exchanged between the user and the server during the cryptojacking sessions. If the messages follow the data exchange sequence illustrated in Table 8, the extension can flag them as cryptojacking. This will prevent cryptojacking even if WebSocket requests are relayed through a third party.

To experimentally demonstrate that, we developed a web extension that detects the strings of data exchange shown in Table 8 and notifies the user when the website starts cryptojacking. To test our extension against the existing countermeasures, we deployed a proxy server relaying the data between our test website to the *dropzone* server as shown in Figure 7. We installed four Chrome extensions that detect cryptojacking: No Coin, Anti Miner, No Mining, and Mining Blocker. Since all of these extensions

take a blocklisting approach for detection, they failed to detect cryptojacking in the presence of the relay server. However, when we used our web extension, it immediately flagged cryptojacking upon reading the data exchanged between the browser and the relay server. Therefore, we conclude that the blocklisting approach is insufficient to counter cryptojacking. In contrast, better countermeasures can be developed through web extensions that inspect the WebSocket payloads. Lightweight browser extension will not cause excessive resource usage as opposed to cryptojacking itself. Since the extension will only read the messages in the WebSocket connection setup, the expected resource overhead will be negligible.

7.2 Adaptive Adversary and Countermeasures

An adaptive adversary can avoid detection by modifying the cryptojacking script to be similar to benign *JavaScript*. Additionally, the adversary can circumvent WebSocket detection through encryption and dummy messages. We note that at the code level, and as shown in our datasets, benign *JavaScript* codes are clearly different from cryptojacking codes. Therefore, if an adversary wants to avoid detection, the adversary needs to significantly alter the cryptojacking scripts to mix their features with the features of the benign scripts. Given the gap in the feature space between the two classes, as discussed in the paper, an adaptive adversary that tries to mimic the features of another class (i.e., benign features) may be able to do that, but not without sacrificing functional properties of the cryptojacking code which may not be acceptable to the adversary. Cryptojacking scripts are designed to (1) take control of the CPU power, (2) solve PoW challenges, and (3) maintain persistent connections with a *dropzone* server to exchange data. These characteristics are quite unique and different from other *JavaScript* codes that may simply render an image on the website. Therefore, from a developer's standpoint, writing a cryptojacking script that can perform all such functionalities while giving the same code features indistinguishable from an image rendering *JavaScript* can be difficult to achieve and, therefore, not observed in the wild.

Similarly, WebSocket-based communication in cryptojacking differs from other WebSocket applications (*i.e.,* online chat). In this case, one method to bypass detection (also acknowledged in your comments) is by adding an encryption layer or dummy messages not detected by the browser extension. Although this is a viable circumvention approach, however, the adversary will (1) bear the encryption cost and (2) circumvent detection in the WebSocket channel only. An intrinsic property of cryptojacking is computing hashes on a *nonce* by the victim machine, which can be detected even in the presence of the said circumvention technique. Therefore, resource-based lines of defense can be leveraged to construct more effective countermeasures despite encryption and dummy messages.

7.3 Discussion

Cryptojacking for Revenue. Demonstrating a negative profit/loss disparity, we refute that cryptojacking is a feasible substitute for online advertising. Additionally, the adverse reputation linked with it may deter users from visiting websites engaged in such practices. While ethical boundaries restrain its use, unethical exploitation could escalate as the cryptocurrency market expands and websites remain susceptible to JavaScript injection attacks. Despite limitations, cryptojacking may remain enticing for adversaries seeking quick profits through compromising vulnerable websites and targeting their visitors.

Cryptojacking Countermeasures. As shown in §7.1, the existing countermeasures for cryptojacking can be easily circumvented. Therefore, strong countermeasures are required to prevent websites from becoming an attack vector for cryptojacking. Web hosting platforms and ISPs can apply static and dynamic analyses (§4 and §5) to detect cryptojacking code and analyze its operations using web traffic payloads. Furthermore, based on the *dropzone* server location, ISPs can filter their traffic and prevent payload communication from stopping cryptojacking. As a result, they prevent the spread of cryptojacking as well as inform the website owners and visitors.

8 Related Work

Concurrent to this work, Rüth *et al.* [38] carried out a measurement study to observe the prevalence of cryptojacking among websites. They obtained blacklisted URLs from the No Coin (§7.1) web extension and mapped them on a large corpus of websites obtained from the Alexa Top 1M list. In total, they found 1491 suspect websites involved in cryptojacking. However, as shown in §7.1, the blacklisting approach to detect and prevent cryptojacking has major limitations and may yield insufficient results to measure prevalence accurately. This perhaps explains the smaller size of their dataset (1491 sites). Concurrently, Eskandari *et al.* [39] also examined the prevalence of cryptojacking among websites and used *Coinhive* as the most popular platform for cryptojacking. While carried out in parallel to ours, the studies highlight the issue of cryptojacking through measurements but stop short of conducting any code analysis, detection, and economic analysis for cryptojacking as alternative online ads, two directions which we pursue in detail in this paper.

Huang *et al.* [40] were among the first to notice the illegal use of CPU cycles, through malware attacks, for Bitcoin mining. Tahir *et al.* [41] studied the abuse of virtual machines in cloud services for mining digital currencies. They used micro-architectural execution patterns and CPU signatures to determine if a virtual machine in the cloud was being illegally used for mining purposes and proposed *MineGuard*, a tool to detect mining. Bartino and Nayeem [42] highlighted worms in IoT devices that hijacked them for mining purposes, pointing to the infamous *Linux.Darlloz* worm that hijacked devices running Linux on Intelx86 chip architecture for mining. Sari and Kilik [43] used Open Source Intelligence (OSINT) to study vulnerabilities in mining pools with the Mirai botnet as a case study.

Bijamin *et al.* [44] presented a new attack vector where Internet routers were hijacked to launch man-in-the-middle cryptojacking attacks. Another work by Bijamin*et al.* [45] analyzed 204 cryptojacking campaigns launched over the Internet and observed that most cryptojacking campaigns

were software-based rather than browser-based. Similarly, Pastrana *et al.* [46] performed a longitudinal study of the evolution of illicit cryptomining operations over the Internet and uncovered the dynamics of various cryptomining campaigns over the last decade. Papadopoulos *et al.* [47] examined the impact of in-browser cryptojacking on victim devices and reported that cryptojacking websites increased the CPU temperature by≈53% and decreased the CPU performance by up to 57%. In a similar context, Meland *et al.* [48] derived an opposite conclusion to [47], stating that a well-configured cryptojacking attack does not harm a user device and may go unnoticed by the users.

Cryptojacking remains a subject of interest in the web ecosystem due to evolving attack methods such as cloud-based mining attacks [49] and in-browser cryptojacking. Moreover, in light of recent works [50], [51], it is apparent that existing countermeasures, such as web browser blockers, are inefficient in cryptojacking detection. To improve defenses against evolving cryptojacking attacks, prominent works by Pott *et al.* [52] and Feng *et al.* [53] apply hardware counters or network monitoring techniques to detect cryptojacking. It can be inferred through these studies that accurate cryptojacking detection cannot be achieved solely through one analytical approach (*i.e.,* static analysis). Instead, this problem requires rigorous treatment through multiple analytical approaches. Our work bridges this gap by uniquely consolidating three major dimensions of in-browser cryptojacking through static, dynamic, and economic analysis. Moreover, our dataset evaluation over the years promises a robust solution to combat cryptojacking.

9 Conclusion

This study analyzes in-browser cryptojacking from various angles: characterization, static and dynamic analyses, and economic aspects. Static analysis of 620 websites identifies distinctive code complexity features, allowing for perfect detection of cryptojacking code. Dynamic analysis examines how cryptojacking scripts use CPU, battery, and network resources, shedding light on their functioning. The economic viability of cryptojacking versus advertising is assessed, showing it to be unfeasible. Prior countermeasures are evaluated, and long-term solutions are proposed, drawing on insights from static and dynamic analyses and clustering.

References

[1] M. Saad, A. Khormali, and A. Mohaisen, "Dine and Dash: Static, Dynamic, and Economic Analysis of In-browser Cryptojacking," in *eCrime*, 2019.

[2] M. Saad, A. Anwar, S. Ravi, and D. Mohaisen, "Revisiting nakamoto consensus in asynchronous networks: A comprehensive analysis of bitcoin safety and chainquality," in *ACM CCS*, 2021. [Online]. Available: https://doi.org/10.1145/3460120.3484561

[3] M. Saad, S. Chen, and D. Mohaisen, "Root cause analyses for the deteriorating bitcoin network synchronization," in *IEEE ICDCS*, 2021. [Online]. Available: https://doi.org/10.1109/ICDCS51616.2021.00031

[4] M. Scott, "Cryptomining malware fuels most remote code execution attacks," Feb 2018. [Online]. Available: https://tinyurl.com/y9vhrq9w

[5] M. J. Zuckerman, "Microsoft blocked more than 400,000 malicious cryptojacking attempts in one day," Apr 2018. [Online]. Available: https://tinyurl.com/ya6oj6wm

[6] SLM, "In-browser cryptojacking: What is it and how can you avoid it?" Jan 2018. [Online]. Available: https://supremelevelmedia.com/browser-cryptojacking-can-avoid/

[7] B. Kerbs, "Who and what is coinhive?" 2018. [Online]. Available: https://krebsonsecurity.com/2018/03/who-and-what-is-coinhive/

[8] Coinhive, "Monero JavaScript Mining," 2018. [Online]. Available: https://coinhive.com/documentation

[9] N. Mathur, "Cybersecurity: Cryptojacking attacks exploded by 8,500% in 2017, says report," Apr 2018. [Online]. Available: https://tinyurl.com/y84alobt

[10] D. Singh, "Cryptojacking attacks rose by 8,500% globally in 2017: report," 2018. [Online]. Available: https://tinyurl.com/y9k4ug2q

[11] J. Condliffe, "A cryptojacking attack hit thousands of websites," 2018. [Online]. Available: https://tinyurl.com/ybjck22l

[12] A. D. Rayome, "Tesla public cloud environment hacked, attackers accessed 'non-public' company data," 2018. [Online]. Available: https://tinyurl.com/y8m79px4

[13] N. De, "UK cyber security division issues warning on pc 'cryptojacking'," Apr 2018. [Online]. Available: https://www.coindesk.com/uk-cyber-security-division-issues-warning-on-pc-cryptojacking/

[14] NCSC, "The cyber threat to uk business 2017-2018 report," Apr 2018. [Online]. Available: https://www.ncsc.gov.uk/cyberthreat

[15] R. Shaikh, "The pirate bay is cryptojacking its visitors' computers to mine monero," 2017. [Online]. Available: https://tinyurl.com/y9s5mhce

[16] M. Zuckerman, "The ethics of cryptojacking: Rampant malware or ad-free internet?" 2018. [Online]. Available: https://tinyurl.com/yd6u9h39

[17] TeamSymantec, "Threat landscape trends q2 2020." [Online]. Available: https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/threat-landscape-trends-q2-2020

[18] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[19] A. Zimba, Z. Wang, M. Mulenga, and N. H. Odongo, "Crypto mining attacks in information systems: An emerging threat to cyber security," *J. Comput. Inf. Syst.*, vol. 60, no. 4, pp. 297–308, 2020. [Online]. Available: https://doi.org/10.1080/08874417.2018.1477076

[20] T. Loechner, "Pixalate unveils the list of sites secretly mining cryptocurrency," 2017. [Online]. Available: https://tinyurl.com/y9sbgx92

[21] X. Yang, "List of top Alexa websites with web-mining code embedded on their homepage," 2017. [Online]. Available: https://tinyurl.com/ybo6u4pf

[22] S. Calzavara, A. Rabitti, and M. Bugliesi, "Semantics-based analysis of content security policy deployment," *TWEB*, vol. 12, no. 2, pp. 10:1–10:36, 2018. [Online]. Available: https://doi.org/10.1145/3149408

[23] SimilarWeb, "Top websites ranking," 2018. [Online]. Available: https://www.similarweb.com/top-websites

[24] M. Community, "Monero cryptocurrency," 2018. [Online]. Available: https://monero.org/

[25] J. Community, "JSECoin: Digital currency - designed for the web," 2018. [Online]. Available: https://jsecoin.com/

[26] Wizsche, "Malicious javascript dataset," https://github.com/geeksonsecurity/js-malicious-dataset.git, 2017.

[27] C. B. Staff, "21 top examples of javascript," 2017. [Online]. Available: https://tinyurl.com/y8wqarpb

[28] N. E. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Transactions on software engineering*, vol. 25, no. 5, pp. 675–689, 1999.

[29] B. Badge, "Es-analysis/plato," Aug 2016. [Online]. Available: https://github.com/es-analysis/plato

[30] NetLab360, "Netlab360 cryptojacking code dataset." [Online]. Available: https://web.archive.org/web/20180209220357/https://blog.netlab.360.com/file/top_web_mining_sites.txt

[31] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, 2009. [Online]. Available: https://doi.org/10.1109/TKDE.2008.239

[32] J. Liu, J. Chen, and J. Ye, "Large-scale sparse logistic regression," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France*, 2009, pp. 547–556. [Online]. Available: https://doi.org/10.1145/1557019.1557082

[33] S. Community, "Selenium browser automation," 2018. [Online]. Available: https://www.seleniumhq.org/docs/

[34] Alexa, "The top 500 sites on the websites listed by their 1 month Alexa traffic rank." 2018. [Online]. Available: https://www.alexa.com/topsites
[35] Statista, "Google: ad revenue 2001-2017," 2018. [Online]. Available: https://tinyurl.com/h4rwfyf
[36] A. Hern, "Huge cryptojacking campaign earns just $24 for hackers," Feb 2018. [Online]. Available: https://tinyurl.com/yc5xgvad
[37] R. Keramidas, Feb 2018. [Online]. Available: https://github.com/keraf/NoCoin
[38] J. Rüth, T. Zimmermann, K. Wolsing, and O. Hohlfeld, "Digging into browser-based crypto mining," in *ACM IMC*, 2018, pp. 70–76.
[39] S. Eskandari, A. Leoutsarakos, T. Mursch, and J. Clark, "A first look at browser-based cryptojacking," in *IEEE EuroS&P Workshops*, 2018. [Online]. Available: https://doi.org/10.1109/EuroSPW.2018.00014
[40] D. Y. Huang, H. Dharmdasani, S. Meiklejohn, V. Dave, C. Grier, D. McCoy, S. Savage, N. Weaver, A. C. Snoeren, and K. Levchenko, "Botcoin: Monetizing stolen cycles," in *ISOC NDSS*, 2014. [Online]. Available: https://www.ndss-symposium.org/ndss2014/botcoin-monetizing-stolen-cycles
[41] R. Tahir, M. Huzaifa, A. Das, M. Ahmad, C. A. Gunter, F. Zaffar, M. Caesar, and N. Borisov, "Mining on someone else's dime: Mitigating covert mining operations in clouds and enterprises," in *RAID*, 2017.
[42] E. Bertino and N. Islam, "Botnets and internet of things security," *Computer*, vol. 50, no. 2, pp. 76–79, 2017.
[43] A. Sari and S. Kilic, "Exploiting cryptocurrency miners with oisnt techniques," *Transactions on Networks and Communications*, vol. 5, no. 6, 2017.
[44] H. L. J. Bijmans, T. M. Booij, and C. Doerr, "Just the tip of the iceberg: Internet-scale exploitation of routers for cryptojacking," in *ACM CCS*, 2019. [Online]. Available: https://doi.org/10.1145/3319535.3354230
[45] H. L. Bijmans, T. M. Booij, and C. Doerr, "Inadvertently making cyber criminals rich: A comprehensive study of cryptojacking campaigns at internet scale," in *USENIX Security*, 2019. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/bijmans
[46] S. Pastrana and G. Suarez-Tangil, "A first look at the crypto-mining malware ecosystem: A decade of unrestricted wealth," in *ACM IMC*, 2019. [Online]. Available: https://doi.org/10.1145/3355369.3355576
[47] P. Papadopoulos, P. Ilia, and E. P. Markatos, "Truth in web mining: Measuring the profitability and the imposed overheads of cryptojacking," in *ISC*, 2019. [Online]. Available: https://doi.org/10.1007/978-3-030-30215-3_14
[48] P. H. Meland, B. H. Johansen, and G. Sindre, "An experimental analysis of cryptojacking attacks," in *Nordic Conference Secure IT Systems*, 2019. [Online]. Available: https://doi.org/10.1007/978-3-030-35055-0_10
[49] R. Xiao, T. Li, S. Ramesh, J. Han, and J. Han, "Magtracer: Detecting GPU cryptojacking attacks via magnetic leakage signals," in *Proceedings of International Conference on Mobile Computing and Networking, MobiCom Madrid, Spain*, X. Costa-Pérez, J. Widmer, D. Perino, D. Giustiniano, H. Al-Hassanieh, A. Asadi, and L. P. Cox, Eds. ACM, 2023, pp. 68:1–68:15. [Online]. Available: https://doi.org/10.1145/3570361.3613283
[50] P. Rajba and W. Mazurczyk, "Limitations of web cryptojacking detection: A practical evaluation," in *International Conference on Availability, Reliability and Security, Vienna,Austria*. ACM, 2022, pp. 52:1–52:6. [Online]. Available: https://doi.org/10.1145/3538969.3544466
[51] R. K. Sachan, R. Agarwal, and S. K. Shukla, "DNS based in-browser cryptojacking detection," in *International Conference on Blockchain Computing and Applications, BCCA, San Antonio, TX, USA.*, M. A. Alsmirat, M. Aloqaily, Y. Jararweh, and I. Alsmadi, Eds. IEEE, 2022, pp. 259–266. [Online]. Available: https://doi.org/10.1109/BCCA55292.2022.9922245
[52] C. Pott, B. Gülmezoglu, and T. Eisenbarth, "Overcoming the pitfalls of hpc-based cryptojacking detection in presence of gpus," in *ACM Conference on Data and Application Security and Privacy, CODASPY, Charlotte, NC, USA*. ACM, 2023, pp. 177–188. [Online]. Available: https://doi.org/10.1145/3577923.3583655
[53] Y. Feng, J. Li, and D. Sisodia, "Cj-sniffer: Measurement and content-agnostic detection of cryptojacking traffic," in *International Symposium on Research in Attacks, Intrusions and Defenses, Limassol, Cyprus*. ACM, 2022, pp. 482–494. [Online]. Available: https://doi.org/10.1145/3545948.3545973

Muhammad Saad obtained his Ph.D. in Computer Science from the University of Central Florida in 2021. Since then, he has been a senior research scientist at PayPal. His research interest is focused on the security of distributed systems, emphasizing blockchain, cryptocurrency, and fraud prevention. His work has appeared in various reputable venues, including ACM CCS, IEEE S&P, and IEEE ICDCS, among others, and received the best paper award at ACM DLoT 2018.

David Mohaisen obtained his Ph.D. in Computer Science from the University of Minnesota in 2012. He is currently a professor of computer science at the University of Central Florida, where he leads the Security and Analytics Lab (SEAL), which he has been leading since 2017. Previously, he was an Assistant Professor at SUNY Buffalo (2015-2017) and a Senior Scientist at Verisign Labs (2012-2015). His research interests are in applied security and privacy, covering aspects of computer and networked systems, software systems, IoT and AR/VR, and machine learning. His research has been published in top conferences and journals, with multiple best paper awards. His work was also featured in the New Scientist, MIT Technology Review, ACM Tech News, Science Daily, etc. Among other services, he has been an Associate Editor of IEEE TMC, TDSC, TCC, and TPDS. He is a senior member of ACM (2018) and IEEE (2015), a Distinguished Speaker of the ACM, and a Distinguished Visitor of the IEEE Computer Society.