Dine and Dash: Static, Dynamic, and Economic Analysis of In-Browser Cryptojacking

Muhammad Saad
University of Central Florida
saad.ucf@knights.ucf.edu

Aminollah Khormali
University of Central Florida
aminkhormali@knights.ucf.edu

Aziz Mohaisen
University of Central Florida
mohaisen@ucf.edu

***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 resources of the website visitors. To understand and counter such attacks, we systematically analyze the static, dynamic, and economic aspects of in-browser cryptojacking. For static analysis, we perform content-, currency-, 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 unsupervised learning to distinguish cryptojacking scripts from benign and other malicious JavaScript samples with 96.4% accuracy. For dynamic analysis, we analyze the effect of cryptojacking on critical system resources, such as CPU and battery usage. Additionally, we 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 large negative profit and loss gap, indicating that the model is economically infeasible. Finally, by leveraging insights from our analyses, we build countermeasures for in-browser cryptojacking that improve upon the existing remedies.**

I. Introduction

Proof-of-Work (PoW) is used in several cryptocurrencies, where new tokens are mined through extensive hash operations. However, the use of PoW in cryptocurrencies has led to several avenues of abuse: an adversary may employ various techniques to abuse others' resources for mining purposes and to perform extensive hash calculations at no or low cost. One such technique is cryptojacking, where system resources of a target device are used to compute hashes on behalf of an adversary. Conventional cryptojacking involved installation of a software binary on a target machine that secretly solves PoW and communicates the results to a remote server [1]. Such cryptojacking requires user permission to download the software and a persistent Internet connection to communicate PoW results to the adversary. However, the conventional cryptojacking proved infeasible for several reasons, including the detection by antivirus scanners, and requiring a persistent connection and infection vector [2].

Another form of cryptojacking, known as in-browser cryptojacking, has recently emerged. In-browser cryptojacking does not require installing binaries, or any authorization from users, where *JavaScript* code is used to compute PoW in a web browser and transmit the PoW to a remote server controlled by the adversary [3]. As such, and since they are shielded in the browser's process, the in-browser cryptojacking scripts are undetected by the antivirus scanners, and mining during web browsing ensures uninterrupted transmission of PoW over a persistent Internet connection. Recent works on in-browser cryptojacking perform static and dynamic analysis on cryptojacking websites and *JavaScript* code to understand the *modus operandi* of in-browser cryptojacking, and construct machine learning models to prevent it [4], [5], [6], [7], [8]. However, these studies do not evaluate a parallel and noteworthy premise of cryptojacking, whereby it is considered to be a suitable replacement for online advertisement [9]. Moreover, while the effect of cryptojacking on devices intuitively known to be harmful, a quantitative assessment of such a harm yet to be explored. On the defence side, the existing countermeasures are either ineffective due to simplistic assumptions or suffer from high overheads due to the machine learning models. Novel to this paper, and in addition to performing static and dynamic analysis, we carry out a quantitative evaluation of the impact of cryptojacking on various user devices and we present an economic analysis to study the feasibility of cryptojacking as a replacement for online advertisement. Finally, addressing the limitations of existing detection schemes, we provide simple yet robust methods to counter in-browser cryptojacking.

Contributions and Roadmap. In summary, this paper makes the following key contributions: ❶ Using more than 5,700 websites with cryptojacking scripts, we conduct an in-depth analysis of cryptojacking, highlighting affinities using sectors, top-level domains, etc. (§III). ❷ We conduct static analysis of cryptojacking scripts, to highlight distributions of cryptocurrency used in cryptojacking and code (script) complexity (§IV). As an application of our static analysis, using code complexity features we built an unsupervised clustering system that automatically identifies cryptojacking, malicious, and benign scripts (§IV-C). A reference-based (using ground truth) evaluation of our clustering algorithm yielded an accuracy of $\approx 96\%$. ❸ We perform dynamic analysis to unveil unique characteristics of process and battery usage. Using that, we analyze the effect of cryptojacking on usre devices. We also study the network artifacts of cryptojacking through WebSocket inspection, and use it to later propose effective countermeasures (§V). ❹ We investigate the economics of cryptojacking as an alternative to online advertisement and build an analytical model to estimate users' cost and cryptojacking websites' gain due to cryptojacking operation (§VI). We compare the two

models, vis-à-vis, and show cryptojacking is not a feasible replacement for online advertisement. ❺ Learning from our static and dynamic analyses, we propose simple and effective defense techniques to address cryptojacking (§VII).

Additionally, the rest of this paper includes related work and comparison in §II, background and preliminaries in §III, discussion and concluding remarks in §VIII.

II. Related Work And Comparison

In this section, we review notable research works done to detect, analyze, and prevent cryptojacking. We will also perform a comparative analysis to position our work.

Rüth *et al.* [10] studied the prevalence of cryptojacking by analyzing blacklisted sites from the No Coin (§VII-A) web extension. They mapped those sites on a large corpus of websites obtained from the Alexa Top 1M list, and found 1,491 suspect websites involved in cryptojacking. However, as we later show in §VII-A1, blacklisting approach may have major shortcomings, and is therefore not a feasible defence approach. Eskandari *et al.* [5] also looked into the prevalence of cryptojacking, showing the use of *Coinhive* as the most popular platform. However, they did not perform static or dynamic analysis of cryptojacking scripts to study the intricacies such covert mining practices. A more systematic treatment of in-browser cryptojacking was performed by by Hong *et al.* [4] and Konoth *et al.* [11]. Hong *et al.* performed static analysis on 2,770 cryptojacking websites and developed a machine learning tool called *CMTracker* that detects and prevents cryptojacking. Concurrently, Konoth *et al.* [11] performed a code-based analysis on 13 cryptojacking platforms to analyze various features in cryptojacking *JavaScript* code and develop countermeasures for it. Later, Kharraz *et al.* [8] presented *Outguard*; a cryptojacking detection tool that uses supervised learning to accurately detect covert mining operations with ≈97% accuracy. In their work, they used 6,302 websites and extracted seven features to train an SVM classifier. These notable efforts indeed contributed significantly towards our general understanding of in-browser cryptojacking. However, they did not perform dynamic analysis of cryptojacking scripts to analyze their effect on the user devices. Furthermore, as acknowledged by Kharraz *et al.* [8], a major limitation of their work is the supervised learning model used to develop *Outguard*. Therefore, *Outguard* is vulnerable to an adaptive adversary who can use evasion techniques. In contrast, we use unsupervised learning and achieve a detection accuracy of $\approx 96\%$. Therefore, in static analysis domain, our detection model may be more useful against an adaptive adversary.

In the domain of dynamic analysis, Tahir *et al.* [12] presented a tool called *MineGuard*, that performed a real-time detection of covert mining operations in the cloud. *MineGuard* used hardware-assisted profiling to create discernible signatures for mining algorithms and later use it for detection. Extending their analysis to the in-browser cryptojacking [12], they developed a browser extension that used fine-grained micro-architectural footprint to detect cryptojacking. In our dynamic analysis, we take a different approach to perform a resource profiling and analyze the effect of cryptojacking on user devices. We further look into the semantics of traffic exchange during mining operations and use them to develop effective countermeasures for the real-time detection.

Finally, a key aspect of analysis that requires a comprehensive treatment is the use of cryptojacking as an alternative to the online advertisement. In each of the aforementioned work, we could not find an economic model that justifies or nullifies the replacement of online advertisement ecosystem with cryptojacking. In this paper, we fill the gap and show that cryptojacking may not be a suitable replacement.

III. Background and Preliminaries

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. Generally, *JavaScript* is automatically executed when a website is loaded. Upon visiting a website with cryptojacking code, the visiting host starts a 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 running on this host has *JavaScript* enabled in it.

An ongoing debate sparked in the community for whether cryptojacking can serve as a replacement to 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 'The Pirate Bay", started using cryptojacking as a revenue substitute for online advertisements [13], [14] and become "ads-free operation". However, a counter argument to this model is the claimed to be 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 a user consent, but would also drain batteries in battery-powered platforms, would indirectly affect the user experience, and by locking the CPU power and not allowing users to use other applications.

A. Data Collection

We assembled a data set of cryptojacking websites published by Pixalate [15] and Netlab 360 [16]. Pixalate is a network analytics company that provides data solutions for digital advertising and research. In Nov. 2017, they published a list of 5000 cryptojacking websites that were actively stealing their visitor's processing power to mine cryptocurrency. Netlab 360 (Network Security Research Lab at 360) is a data research platform that provides a wide range of datasets spanning Domain Name Servers (DNS) and Distributed Denial-of-Service (DDoS) attacks. From Netlab 360, we obtained 700 cryptojacking websites, released on Feb 24, 2018.

The top-level domain (TLD) distribution of the combined dataset, including the TLD type (generic, new, or country-level) and the corresponding percentage, is shown in Table I. While, unsurprisingly, .com and .net occupy the first and second spot of the top-10 TLDs represented in the dataset,

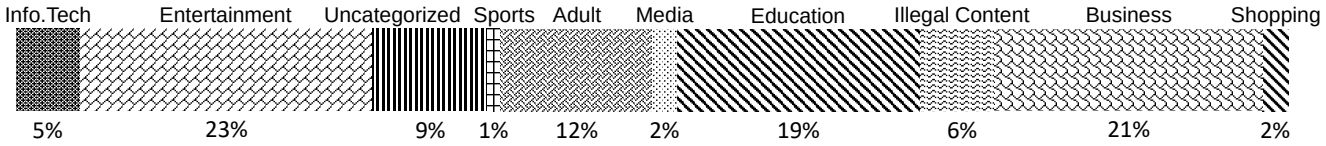


Figure 1: Categorization of websites based on the main topic of their content. Notice that most websites belong to Entertainment, Business, and Education. A sizable chunk (12%) belonged to the Adult category.

Table I: Distribution of cryptojacking websites with respect to top-level domains in our dataset.

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

with a combined total of 40.3% of the websites belong 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 the Pixalate's dataset, 6 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, while 56.8% websites had no "terms and conditions" statement, and 49.3% did not have both the privacy policy and the terms and conditions. This indicates that the majority of those websites could not formally, through those statements, inform their visitors regarding the usage of their processing resources for mining cryptocurrencies, where cryptojacking is used in instead of online advertisement.

B. Methodology

In the static analysis, we categorize the websites based on content and the currency they mine. We extract the cryptojacking code and develop code-based features to examine their properties. We compare them, using those static properties, 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 the 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

Table II: Detailed results of 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

Windows and Linux operated laptops and an Android phone, 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 websites ranking, volume of traffic, and the average time spent by visitors on those websites [17].

In the economic analysis, we first evaluate the profit generated by mining operations. Next, we use the average time spent by various devices on a website to compute the maximum profit generated by those websites. Finally, we compare the profit earned through cryptojacking with the actual revenue earned from advertisement for feasibility analysis.

IV. Static Analysis

We pursue three directions: content- currency-, and code-based analysis. Content-based categorization provides insights into the nature of websites used for cryptojacking, while the currency-based categorization shows services and platforms affected by the threat. The code-based analysis provides insight into the complexity of the cryptojacking scripts, using code complexity measures.

A. Content and Currency-based Categorization

We categorized the websites based on their contents into various categories using the *WebShrinker* website URL categorization API. *WebShrinker* assigns categories to websites based on the main usage of those websites using their contents. The results are shown in Figure 1. As it can be seen in Figure 1, miners have utilized a wide range of categories for in-browser cryptocurrency mining, including education, business, entertainment, etc. Notice in Figure 1, some websites are categorized as "Illegal Content", which were mostly torrent

websites serving pirated contents. Moreover, 19% websites were categorized as "Education" which can be attributed to the exploitation of trust by adversaries behind cryptojacking [18].

By analyzing our dataset, we found eight platforms providing templates to mine two types of cryptocurrencies: Monero and JSEcoin. In Table II, we provide details of the eight platforms and their mining cryptocurrency. We found that a large percentage of the websites (≈81.57%) use *Coinhive* [19] to mine Monero cryptocurrency [20], 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 [21], which is responsible for mining the JSEcoin cryptocurrency.

B. Code-based Analysis

For static analysis, we gathered cryptojacking scripts from all the major cryptojacking service providers found in our dataset, such as *Coinhive*, JSEcoin, Crypto-Loot, Hashing, deepMiner, Freecontent, Miner, and Authedmine. We observed that all the service providers had unique codes, specific to their own platform. In other words, the websites using *Coinhive*'s services had the same *JavaScript* code template across all of them. Therefore, ≈81.57% of the websites in our dataset were using the same *JavaScript* template for cryptojacking. Similarly, all the websites using JSEcoin used the same standard template for their mining. However, the code template of each service provider was different from one another, 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 other standard *JavaScript* for a baseline comparison.

1) Data Attributes: 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 among all the websites, each of which belongs to one of the service providers. As a control experiment, we collected an equal number of malicious and benign *JavaScript* codes to design a clustering algorithm. Our aim was to obtain a set of features that were unique only to the cryptojacking scripts, and aid in their detection. With such knowledge of those features, more accurate countermeasures can be further developed that will accurately predict if a given host machine is under cryptojacking attack. To avoid bias towards a certain class, we were limited to include equal size of malicious and benign *JavaScript* samples for the static analysis. Although there are many samples of malicious and benign *JavaScript* in the wild [22], only eight cryptojacking scripts are available in comparison. Since our work is focused on distinguishing cryptojacking scripts from both malicious and benign *JavaScript*, we had to balance the size of each class. While the number of scripts might seem as a limitation of our work, we believe the promise of this work is substantial: as more currencies and platforms start to use cryptojacking, more samples will be available for a broader study.

In lieu, we used the existing data of the cryptojacking websites (§III-A) and online resources from GitHub for malicious *JavaScript* sample [23] . For benign *JavaScript*, we used the set of non-cryptojacking websites and parsed their HTML code to extract benign *JavaScript* code [24]. In summary, we had 8 samples of cryptojacking *JavaScript* samples, spanning all the websites. Accordingly, we selected 10 malicious and 10 benign scripts for our clustering analysis (which serves as a multi-class classification).

2) Feature Extraction: We use various features that provide insights into the structure of the code and its maintainability. In the following, we describe the features we extracted for our static analysis of cryptojacking, malicious, and benign scripts.

Cyclomatic Complexity. Cyclomatic complexity [25], [26] measures the complexity of code using Control Flow Gaph (CFG). It relies on a directed flow graph where each node represents a function to be executed and a directed edge between the two nodes indicates that the node representing the function will be executed after the previous node. Let E be the number of edges, N be the number of nodes, and Q be the number of connected components in the CFG of a program, then 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 [27] is a measure of Cyclomatic complexity, defined above, spread over the total code length. Usually, malware authors obfuscate their code to avoid detection. As such, among many other possibilities of obfuscation, they may alter the flow of a program and add extra functions. While adding more functions and lines of code will certainly increase the size of the code, its complexity will remain the same, which could be used as a feature of their detection. Let c_l be the total number of lines of code, then the cyclomatic complexity density, denoted by M_d, can be computed as $M_d = \frac{E+2P-N}{c_l}$

Halstead Complexity Measures. In software testing, the Halstead complexity measures are used as metrics to characterize the algorithmic implementation of a programming language [28]. 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:

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

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

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

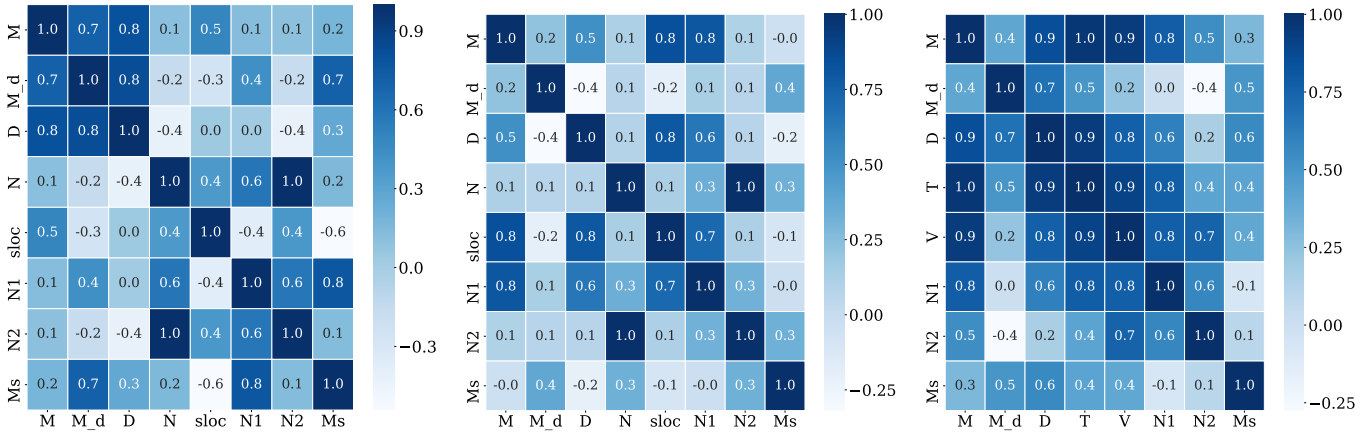
$$T = (D \times V)/18, \qquad B = E^{\frac{2}{3}}/3000 \tag{4}$$

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:

$$M_s = 171 - 5.2 \log(V) - 0.23M - 16.2 \log(c_l)$$

Table III: Static features of cryptojacking, malicious, and benign samples. Mean ((μ)) and Standard Deviation (σ) are 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



(a) Correlation in cryptojacking JavaScript (b) Correlation in malicious JavaScript (c) Correlation in benign JavaScript

Figure 2: Heatmap of correlation coefficients among the features of three categorizes of JavaScript. It can be noted that features among benign scripts appear to be highly correlated while the features among malicious scripts remain highly uncorrelated. Correlation among the features of cryptojacking scripts remains in the middle, relative to the other two.

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

Results: To extract the aforementioned features in our code-based analysis, we used *Plato*, a *JavaScript* static analysis and source code complexity tool [29]. For each *JavaScript* code, we run *Plato* and record the 17 extracted features, highlighted above, as reported in Table III. From Table III, we observed that certain features, such as M, M_d, V, and T, are clearly discriminative among all the categories. For further analysis, in the next section we will look into the correlation of these features among each category to see whether there is a unique pattern among each category, which allow us to build a classification system that can automatically classify different *JavaScript* categories based on the extracted features.

3) Correlation Analysis: Presenting individual features among those analyzed above, while meaningful, might not shed light on their distinguishing power given their large

numbers. To this end, we pursue a correlation analysis to understand their patterns. In particular, we conducted a correlation analysis to observe the similarity of features among the three categorizes of scripts, the cryptojacking, malicious, and benign. The correlation analysis showed the consistency of the relationship distinctive to each category of the *JavaScript* codes. As such, this provided us with insights into coding patterns and features unique to the style of coding cryptojacking scripts, malware scripts, and benign scripts. We computed the correlation of the features in all the scripts belonging to each category of *JavaScript*. We used the Pearson correlation coefficient for this analysis, which is defined as $\rho(X,Y) = Cov(X,Y)/(\sqrt{Var(X)Var(Y)})$, where X and Y are random variables, Var and Cov are the *variance* and *covariance*, respectively.

The results of all the features belonging to each category of *JavaScript* is omitted for the lack of space. In lieu, we selected eight features from the complete set of seventeen features and plotted their result in Figure 2. It can be observed that cryptojacking scripts are more correlated with respect to Cyclomatic complexity M score, Time T and Volume V while malicious scripts are not as correlated over those parameters. To this end, these features can be used to identify the coding patterns of scripts belonging to each class of *JavaScript* and can further be utilized as building blocks to extract program flow information during code execution analysis.

C. Clustering

In this section, we build a classification system that automatically recognizes cryptojacking scripts from malicious and benign scripts based on the code complexity features alone, which could be easily extracted from the cryptojacking scripts and are common among a large number of cryptojacking websites. It is desirable for our classification system to classify scripts even with minimal information regarding the labels of the scripts. Therefore, we utilized the Fuzzy C-Means (FCM) clustering algorithm [30], which has the advantage of being an unsupervised learning algorithm. In the other words, in comparison with supervised classification algorithms, such as the Support Vector Machine (SVM) and Random Forest (RF), which require labels of the dataset in the training phase, FCM has the advantage of performing well on the unlabeled dataset.

The main goal of the FCM is to group a dataset X into C clusters in which every data point belongs to every cluster to a certain degree. In other words, a data point that lies close to the center of a certain cluster will have a higher membership degree to that cluster, whereas the membership degree of the data point that lies far away from the center of this cluster will be lower [30]. We utilized the FCM clustering algorithm to group the scripts to cryptojacking, malicious, and benign clusters. In order to evaluate the performance of the clustering experiment, we used standard evaluation metrics; the confusion matrix, Accuracy Rate (AR), False Positive Rate (FPR), and False Negative Rate (FNR), reported in Table IV.

As shown in Table IV, the clustering algorithm is able to classify the scripts with high performance: AR of ≈96.4%, FPR of 3.3%, and FNR of 3.7%. In addition, we have visualized these clusters based on two major principal components of their features, which in Figure 3, clearly show a natural separation between the clusters using the underlying features.

Table IV: Confusion matrix and evaluation metrics of the cryptojacking (CJ), malicious, and benign scripts' clustering results based on FCM clustering algorithm. $^{(1)}$ Evaluation metrics' names are abbreviated. FPR= False Positive Rate, FNR= False Negative Rate, and AR=Accuracy Rate.

Class	Benign	Malicious	CJ	FPR%$^{(1)}$	FNR%$^{(1)}$	AR%$^{(1)}$
Benign	9	0	1	10	0	90
Malicious	0	10	0	0	0	100
CJ	0	0	8	0	11.1	100
Total				3.3	3.7	96.42

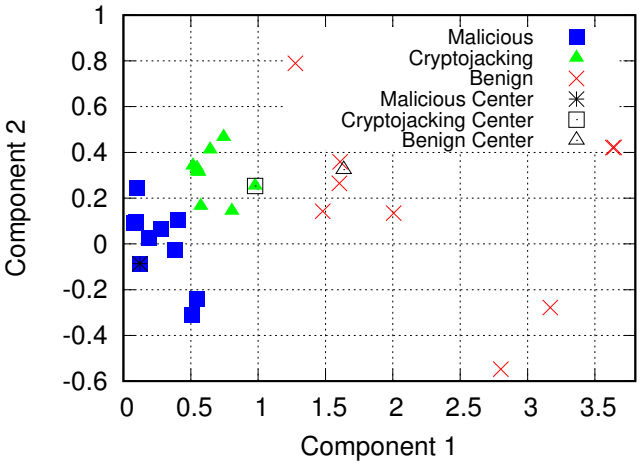


Figure 3: Clustering of the cryptojacking, malicious, and benign scripts using FCM clustering algorithm.

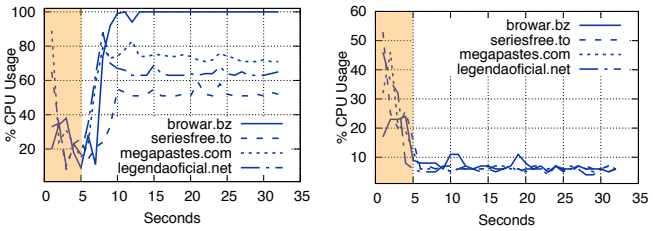
In March 2019, *Coinhive*, the leading platform for in-browser cryptojacking was shut down due to a fall in the price of Monero. Although it was widely perceived that this event might also deter cryptojacking, several leading researchers postulate that cryptojacking has been on the surge during 2019 [31], since alternatives to Coinhive, such as Cryptoloot and JSEcoin, already exist and are becoming increasingly popular. New platforms may also emerge and may simply use the publicly available *JavaScript* code of Coinhive. Therefore, cryptojacking as a threat cannot be ignored, and our work carried out prior to the closure of Coinhive, is still useful in providing such fundamental and new insights.

V. Dynamic Analysis

Despite the clear benefits of the static analysis outlined above, it is limited, and subject to circumvention through *JavaScript* code obfuscation. To this end, we conduct dynamic analysis that looks into profiling the usage of cryptojacking *JavaScript* code of various host resources: CPU, and

Listing 1: Script code found in cryptojacking websites.

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



(a) JavaScript enabled (b) JavaScript disabled

Figure 4: Processor usage by four different cryptojacking websites with JavaScript enabled and disabled.

battery. We then look into the characteristics of cryptojacking in their use of network resources.

A. Resource Consumption Profiling

We conduct an extensive analysis of CPU and battery usage of the various cryptojacking scripts.

1) 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 ultimately sent. Listing 1 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 tag. In this script, we also noticed a *throttling parameter*, which is used as a mean of controlling how much resources a cryptojacking script uses on the host. We use such a 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 resources usage in different platforms, we use battery-powered machines running Microsoft Windows, Linux, and Android operating systems (OSes). For our experiments, we selected three laptops, each with one of those OSes. The Windows laptop used in the experiment was Asus V502U, with Intel Core i7-6500U processor operating at 3.16 GHz. The Linux laptop was Lenovo G50, with Intel Core i5-5200U processor (4 cores) running at 2.20 GHz, and the Android phone was *Samsung Galaxy J5*, with android version of 6.0.1.

For our cryptojacking script construction, using the various parameters learned above, we set up an account on *Coinhive* to obtain a key that links our "experiment website" to the server. Next, we set up a test website and embedded the code in Listing 1 within the HTML tags of the website. Finally, to measure the usage of resources while running cryptojacking websites, we set up a Selenium-based web browser automation and run cryptojacking websites, for various evaluations. Selenium is a portable web-testing software that mimics actual web browsers [32], [33].

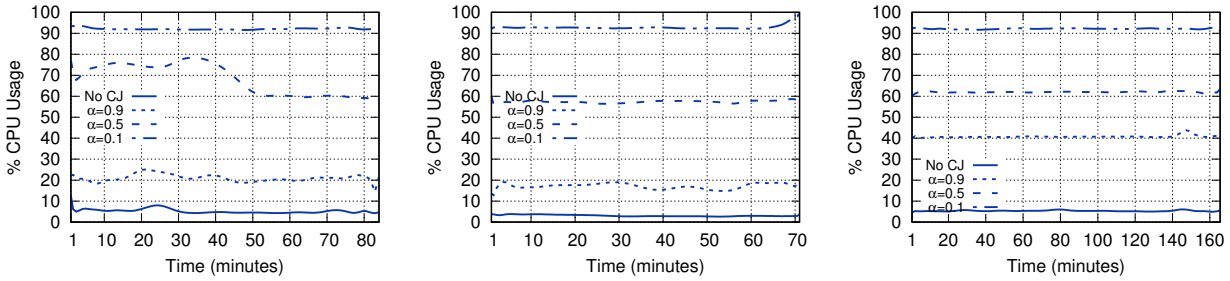
2) CPU Usage: First, we baseline our study to highlight CPU usage as a fingerprint across multiple websites that employ cryptojacking using the aforementioned configurations and measurement environment. We study the usage of CPU with and without cryptojacking in place. For this experiment, we select four cryptojacking websites. To measure the impact of cryptojacking on CPU usage, we ran those websites in our *Selenium* environment, for 30 seconds, with *JavaScript* enabled (thus running the cryptojacking scripts) and disabled (baseline; not running the cryptojacking scripts). We use this experiment as our control.

Results. We obtained two sets of results for each website, with and without cryptojacking. In Figure 4, we plot four test samples obtained from our experiment to demonstrate the behavior of websites with and without cryptojacking. From those results, we observe that when a website is loaded initially it consumes a significant CPU power (shaded region), in both cases. Once the website is loaded, the CPU consumption decays if the *JavaScript* is disabled, indicating no cryptojacking. When *JavaScript* is enabled, the CPU consumption is high, indicating cryptojacking. It can also be observed in Figure 4, that the CPU usage varied across the websites, indicating the usage of the throttling parameter highlighted above. The same behavior as with *JavaScript* disabled is exhibited when loading a page with *JavaScript* that is either benign or of other types of maliciousness than cryptojacking. We found that cryptojacking consumes anywhere between 10 and 20 times compared to when not using cryptojacking on the same host.

To understand the impact of throttling on CPU usage in different platforms, we conduct another measurement where we used $\alpha = \{0.1, 0.5, 0.9\}$ with the different testing machines. We found a consistent pattern, whereby the relationship between α and the CPU usage is linear(Figure 5).

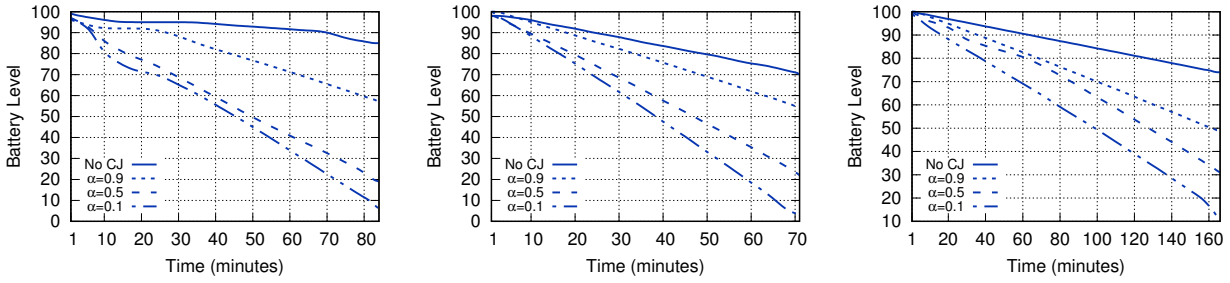
3) Battery Usage: Clearly, high CPU usage translates to higher power consumption, and quicker battery drainage. To further investigate how cryptojacking affects battery drainage, we carried out several experiments using various α values for the various platforms. Here we are interested in the order of battery drainage from a baseline, rather than comparing various platforms. The batteries of the different machines are as follows: 65 watt-hour for Windows, 41 watt-hour for Linux and ≈9.88% watt-hour for Android.

Results. For each $\alpha \in \{0.1, 0.5, 0.9\}$, and using the different devices, 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 measure the baseline by running our script without the cryptojacking code. The results are shown in Figure 6. As expected, with $\alpha = 0.1$, corresponding to the lowest throttling and highest CPU usage, the battery drained very quickly, to ≈10% of its capacity within 80



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

Figure 5: CPU usage recorded on three devices. Windows machine consumed more processing than the other devices during cryptojacking. This also explains the high battery drainage in Figure 6(a).



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

Figure 6: Battery usage recorded on three devices used in the dynamic analysis.

minutes, compared to ≈85% within the same time when not using cryptojacking. The same result is demonstrated for both the Linux laptop and Android phone. We also notice that relationship between α and the battery drainage is also linear.

In examining the CPU and battery usage by cryptojacking websites, as shown above, we highlight a clear and unique patterns that can be used to identify those websites. We also notice that the different operating systems do not have any architectural support to prevent activities like cryptojacking from happening on the device.

B. Network Usage and Profiling

Dynamic network-based artifacts are essential in analyzing cryptojacking scripts, especially when those scripts are obfuscated. To this end, we also explore the network-level artifacts to reconstruct the operation of cryptojacking services.

We noticed that during cryptojacking website execution, the *JavaScript* code establishes a WebSocket connection with a remote server and preforms 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 can still be gathered, such as source, destination, payload size, and request timings, the actual data transferred remain encrypted, preventing further analysis. To perform a deeper analysis on 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.

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 actual user who owns the key of the *JavaScript* and adds balance of hashes to the user's account. The server then authenticates the request parameters and responds back with *authed* message. The *authed* message length is 50 Bytes and it includes a token and the total number of hashes received so far from the client's machine. In the *authed* message, the total number of hashes is 0, since the client has not sent any hashes yet. Then, the server sends *job* message to the client. The *job* message has a length 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 acknowledgement and the total number of hashes received during the session. The *hash_accept* message is 48 Bytes long. This is to be noted that once a webpage is refreshed, the WebSocket connection terminated and restarted. On the other hand, if multiple tabs are opened in the same browser,

Listing 2: WebSocket frames exchanged during cryptojacking.

```
// auth request from client to server
{"type": "auth",
    "params": {
    "site_key": "32 characters key",
    "type": "anonymous", "user": null, "goal": 0 }}
// authed response from server to client
{ "type": "authed",
        "params": {
        "token": "", "hashes": 0 }}
// job request sent by the server to client
{ "type": "job",
        "params": {
        "job_id": "164698158344253",
        "blob": "152 characters blob string",
        "target": "ffffff00" }}
// submit message by client to server
{ "type": "submit",
        "params": {
        "job_id": "164698158344253", "nonce": "
            cfe539d3",
        "result": "256-bit hash" }}
// hash_accept sent by server to client
{ "type": "hash_accept",
        "params": {
        "hashes": 256 }}
```

Table V: Types of messages exchanged between the client and the server during cryptojacking WebSocket connection.

Message	Source	Sink	Length (Bytes)	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

the WebSocket connection remains unaffected. In Table V, we provide details about the WebSocket connection during a cryptojacking session. In Listing 2, we provide the the actual data frames exchanged between the browser and the server during WebSocket session. The data frames are structured in "JavaScript Object Notation" (JSON).

VI. Economics of Cryptojacking

In this section, we evaluate a key aspect of our work that evaluates the economic feasibility of cryptojacking by extrapolating on 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 while 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.

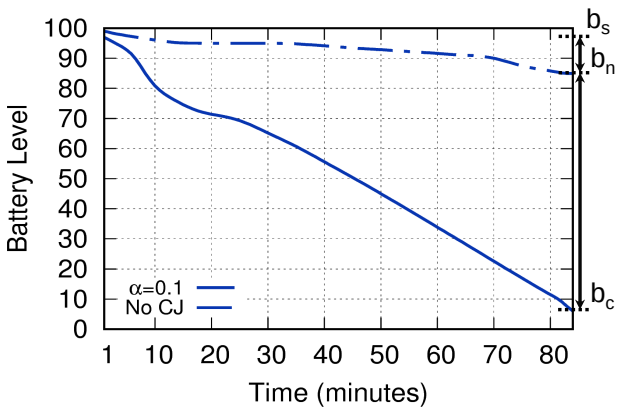


Figure 7: Battery drain sample of Windows i7.

A. Analytical Model

To set a stage for our analysis, in Figure 7 we present the results from one sample experiment conducted on Windows i7 machine with 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 normal operation of the system. 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, and for the time $\Delta t = 85$ minutes from Figure 7, 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 battery during cryptojacking 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 7), b_c = 10% and t_r = 0.015 hour. Thus, the esti-

Table VI: Results of cryptojacking with different devices. Here α is the throttling parameter, h, Δt, b_n, b_c, W, P, and L are the parameters obtained from Equation 5 and Equation 6. T is the estimated time required for each device to mine 1 XMR.

Device	Δt (mins)	$b_n(\%)$	α	h (hps)	b_c (%)	W(W/h)	P(USD)	L (USD)	$L-P$ (USD)	T(years)
Windows	85	82	0.1	21	10	65	6.4×10^{-4}	4.5×10^{-3}	3.8×10^{-3}	50
			0.5	14	19	65	3.1×10^{-4}	3.7×10^{-3}	3.4×10^{-3}	104
			0.9	5	57	65	4.4×10^{-5}	1.6×10^{-3}	1.5×10^{-3}	367
Linux	71	70	0.1	26	3	41	6.6×10^{-4}	5.5×10^{-3}	4.8×10^{-3}	40
			0.5	16	22	41	4.1×10^{-4}	4.2×10^{-3}	3.8×10^{-3}	66
			0.9	5	54	41	1.3×10^{-4}	2.6×10^{-3}	2.5×10^{-3}	214
Android	163	76	0.1	5	11	9.9	2.8×10^{-4}	9.5×10^{-4}	6.7×10^{-4}	220
			0.5	3	32	9.9	1.7×10^{-4}	7.2×10^{-4}	5.5×10^{-4}	369
			0.9	2	49	9.9	1.1×10^{-4}	5.4×10^{-4}	4.3×10^{-4}	574

Table VII: Monthly profit to be earned by top websites by applying cryptojacking. GR: global rank, CR: country rank, visits are in Billions, average time duration of visits is in mm-ss, P-CJ: profit earned by cryptojacking, and P-Ads: revenue earned through ads. "—" is used for undisclosed profit.

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	—

Table VIII: Estimated monthly earnings of top websites in our dataset. Visits are in millions, average time duration of each visit is in mm-ss and 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

mated loss during cryptojacking session L was $\approx \$4.5\times10^{-3}$ USD, which is 7 times the value of P, highlighting a big gap cryptojacking's operation model.

Using the same analysis, we examine if cryptojacking can be used as a source of income by users. 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). In Table VI, we report all the results obtained from the experiment for each device used in for our experiments in the dynamic analysis, along with the amount of time required for each device to mine 1 XMR.

B. Cryptojacking and Online Advertisement

In-browser cryptojacking is argued as an alternative to online advertisement. To understand this, we performed an experiment to compare the monetary value of in-browser cryptojacking as a replacement to 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. We use this information and our model from section VI-A to measure the potential profit those websites could have made using cryptojacking. We assume that visitors on these websites have the average hash rate of 20 hashes/second. We report the results in Table VII, highlighting that those websites would make between $3.65 million USD (for *youtube.com*) and $0.10 million USD (*qq.com*) per month (on average).

Statista [35] publishes annual online advertisement revenue reports. We collect the revenues generated by each of those top-10 websites for the year 2017 (most recent report). 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 VII, 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 per month, at most. 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 the estimated profit earned by them, shown in Table VIII. We notice that the maximum profit, earned by *firefoxchina* is ≈$2,747 USD. Although, the ad revenue for these websites is not available online,

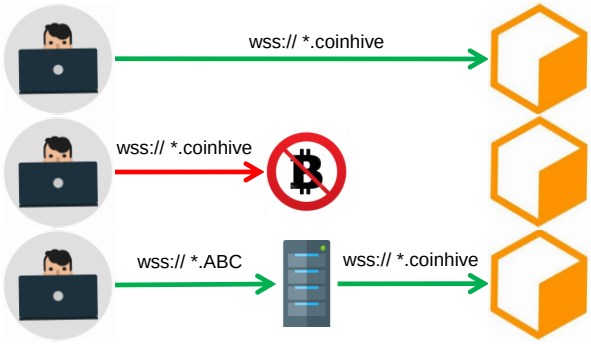


Figure 8: Circumventing cryptojacking detection by relaying WebSocket requests through a third party proxy server.

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 are in-line with recent reports indicating that an adversary who compromised 5,000 websites and injected his own cryptojacking scripts was only able to make $24 USD [36].

VII. Countermeasures

A. Existing Countermeasures

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

1) Evading Detection: An attacker can evade detection by setting his own third party server to relay data to and from cryptojacking server. In Figure 8, we show how the current countermeasures for cryptojacking can be circumvented by an adaptive attacker. To practically demonstrate that, we set up a test website using *Coinhive* script and installed a local relay server. We installed four chrome extensions that block the in-browser cryptojacking: No Coin, Anti Miner, No Mining, and Mining Blocker. In the first phase of the experiment, we installed the *Coinhive* script and ran the website. Each extension detected the WebSocket request and blocked it. We then configured our relay server to act as a proxy. In the *Coinhive* script, we modified the code and replaced the *Coinhive* socket address with our server address. Next, when we visited the website, it started cryptojacking on the client machine undetected.

2) Countering Adaptive Attacker: Instead of blocking specific URLs, the extensions can monitor the messages exchanged between the user and the server during cryptojacking session. If the messages follow the sequence of web frames that we have illustrated in Listing 2, the extension can flag them as cryptojacking. This will prevent cryptojacking even if WebSocket requests are relayed through a third party. We developed a chrome web extension that detects the strings of web frames shown in Listing 2, and notifies the user when the website starts cryptojacking. To test our extension against the existing countermeasures, we deployed a proxy server that relayed the data between our test website to the *dropzone* server as shown in Figure 8. Our web extension installed in the browser immediately flagged cryptojacking upon reading the actual data exchanged between browser and relay server.

VIII. Discussion and Conclusion

By showing a huge negative profit/loss gap, we settle the argument that cryptojacking is not a viable alternative for online advertisement at the moment, and due to the adverse effects on user devices, it may not be an ethical way of generating revenues for online web service. Furthermore, the scope of the ethical use of cryptojacking may be limited, and it is likely that the unethical use may increase as the cryptocurrency market grows and the websites remain vulnerable to *JavaScript* injection attacks. Cryptojacking attacks can be launched solely to abuse devices of websites visitors, thereby influencing the reputation of such websites. Therefore, cryptojacking provides multiple attack avenues and we cannot ignore their potential threat, especially with the availability of a large array of services that provides cryptojacking scripts and capabilities.

As a direct result of our dynamic analysis, we show that simple and effective real-time detection of cryptojacking can be possible by the direct inspection of WebSocket payload (§V). Using that as a primary detection method can decrease the overhead in comparison with sophisticated machine learning tools, and address the limitations of blacklisting approaches.

In summary, our work systematically analyzed in-browser cryptojacking through the lenses of characterization, static and dynamic analyses, and economic analysis. Our static analysis unveils unique code complexity characteristics and can be used to detect cryptojacking code from malicious and benign code samples with ≈96% accuracy. We explore, through dynamic analysis, how in-browser cryptojacking uses various resources, such as CPU, battery, and network, and use that knowledge to reconstruct the operation of cryptojacking scripts. We also study the economical feasibility of cryptojacking as an alternative to advertising, highlighting its infeasibility. By surveying prior countermeasures and examining their limitations, we show simple and effective methods to counter cryptojacking, capitalizing on the insights from our dynamic analysis.

References

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

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

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

[4] G. Hong, Z. Yang, S. Yang, L. Zhang, Y. Nan, Z. Zhang, M. Yang, Y. Zhang, Z. Qian, and H. Duan, "How you get shot in the back: A systematical study about cryptojacking in the real world," in *ACM SIGSAC Conference on Computer and Communications Security, CCS, Toronto, ON, Canada*, Oct 2018, pp. 1701–1713. [Online]. Available: https://doi.org/10.1145/3243734.3243840

[5] S. Eskandari, A. Leoutsarakos, T. Mursch, and J. Clark, "A first look at browser-based cryptojacking," in *IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops, London, United Kingdom*, Apr 2018, pp. 58–66. [Online]. Available: https://doi.org/10.1109/EuroSPW.2018.00014

[6] R. Tahir, S. Durrani, F. Ahmed, H. Saeed, F. Zaffar, and S. Ilyas, "The browsers strike back: Countering cryptojacking and parasitic miners on the web," in *IEEE Conference on Computer Communications, INFOCOM, Paris, France*, April 2019, pp. 703–711. [Online]. Available: https://doi.org/10.1109/INFOCOM.2019.8737360

[7] D. Carlin, P. O'Kane, S. Sezer, and J. Burgess, "Detecting cryptomining using dynamic analysis," in *Annual Conference on Privacy, Security and Trust, PST 2018, Belfast, Northern Ireland, Uk*, Aug 2018, pp. 1–6. [Online]. Available: https://doi.org/10.1109/PST.2018.8514167

[8] A. Kharraz, Z. Ma, P. Murley, C. Lever, J. Mason, A. Miller, N. Borisov, M. Antonakakis, and M. Bailey, "Outguard: Detecting in-browser covert cryptocurrency mining in the wild," in *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA*, May 2019, pp. 840–852. [Online]. Available: https://doi.org/10.1145/3308558.3313665

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

[10] J. Rüth, T. Zimmermann, K. Wolsing, and O. Hohlfeld, "Digging into browser-based crypto mining," in *Proceedings of the Internet Measurement Conference*, ser. IMC. New York, USA: ACM, 2018. [Online]. Available: http://doi.acm.org/10.1145/3278532.3278539

[11] R. K. Konoth, E. Vineti, V. Moonsamy, M. Lindorfer, C. Kruegel, H. Bos, and G. Vigna, "Minesweeper: An in-depth look into drive-by cryptocurrency mining and its defense," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, 2018. [Online]. Available: http://doi.acm.org/10.1145/3243734.3243858

[12] 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 *International Symposium on Research in Attacks, Intrusions and Defenses (RAID), Atlanta, GA, USA*, Sept 2017, pp. 287–310.

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

[14] Ernesto, "The pirate bay website runs a cryptocurrency miner (updated)," Sep 2017. [Online]. Available: https://torrentfreak.com/the-pirate-bay-website-runs-a-cryptocurrency-miner-170916/

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

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

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

[18] A. Zarras, A. Kapravelos, G. Stringhini, T. Holz, C. Kruegel, and G. Vigna, "The dark alleys of madison avenue: Understanding malicious advertisements," in *Proceedings of Internet Measurement Conference, IMC, Vancouver, Canada*, Nove 2014, pp. 373–380. [Online]. Available: https://tinyurl.com/ybqmcjmb

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

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

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

[22] C. Curtsinger, B. Livshits, B. G. Zorn, and C. Seifert, "ZOZZLE: Fast and precise in-browser javascript malware detection," in *Proceedings of the 20th USENIX Security Symposium (Security)*, CA, Aug. 2011.

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

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

[25] T. J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, no. 4, pp. 308–320, 1976.

[26] T. J. M. Arthur H. Watson and D. R. Wallace, *Structured testing: A testing methodology using the cyclomatic complexity metric*. US Department of Commerce, National Institute of Standards and Technology, 1996, vol. 500, no. 235.

[27] 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.

[28] A. Serebrenik, "Software metrics," http://www.win.tue.nl/~aserebre/2IS55/2010-2011/10.pdf, 2011.

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

[30] J. C. Bezdek, R. Ehrlich, and W. Full, "Fcm: The fuzzy c-means clustering algorithm," *Computers and Geosciences*, vol. 10, pp. 191–203, 1984.

[31] K. Williamss, "Cryptojacking is making a comeback," 2019. [Online]. Available: https://smartermsp.com/cryptojacking-is-making-a-comeback/

[32] A. Bruns, A. Kornstadt, and D. Wichmann, "Web application tests with selenium," *IEEE software*, vol. 26, no. 5, 2009.

[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