# Thieves in the Browser: Web-based Cryptojacking in the Wild

Marius Musch
TU Braunschweig

Christian Wressnegger
TU Braunschweig

Martin Johns
TU Braunschweig

Konrad Rieck
TU Braunschweig

## ABSTRACT

With the introduction of memory-bound cryptocurrencies, such as Monero, the implementation of mining code in browser-based JavaScript has become a worthwhile alternative to dedicated mining rigs. Based on this technology, a new form of parasitic computing, widely called *cryptojacking* or drive-by mining, has gained momentum in the web. A cryptojacking site abuses the computing resources of its visitors to covertly mine for cryptocurrencies. In this paper, we systematically explore this phenomenon. For this, we propose a 3-phase analysis approach, which enables us to identify mining scripts and conduct a large-scale study on the prevalence of cryptojacking in the Alexa 1 million websites. We find that cryptojacking is common, with currently 1 out of 500 sites hosting a mining script. Moreover, we perform several secondary analyses to gain insight into the cryptojacking landscape, including a measurement of code characteristics, an estimate of expected mining revenue, and an evaluation of current blacklist-based countermeasures.

## 1 INTRODUCTION

Cryptocurrencies, such as Bitcoin and Ether, have gained popularity in the last years, as they provide an alternative to centrally controlled fiat money and a profitable playground for financial speculation. A basic building block of these currencies is the process of *mining*, in which a group of users solves computational puzzles to validate transactions and generate new coins of the currency [see 27]. Although the stability and long-term perspectives of cryptocurrencies are not fully understood, they have attracted large user communities that mine and trade coins in different markets with considerable volume. For example, Bitcoin reached an all-time high of 19,300 USD per coin in December 2017 [8], resulting in a market value comparable to major companies.

The mining of cryptocurrencies has been largely dominated by dedicated hardware systems, such as GPU and ASIC mining rigs. This situation, however, has started to change with the introduction of *memory-bound* cryptocurrencies, like Monero, Bytecoin, and Electroneum. These currencies build on computational puzzles that are memory intensive and thereby reduce the advantage of specific hardware over commodity processors [see 34, 38]. Consequently, the resulting currencies can be profitably mined on regular computer systems and thus open the door for the widespread application of cryptocurrency mining.

Unfortunately, this development has also attracted miscreants who have discovered cryptocurrencies as a new means for generating profit. By tricking users into unnoticeably running a miner on their computers, they can utilize the available resources for generating revenue—a strategy denoted as *cryptojacking* or drive-by mining [21]. A novel realization of this strategy is injecting mining code into a website, such that the browser of the victim mines during the website's visit. First variants of these attacks have emerged with the availability of the CoinHive miner in September 2017 [1, 22]. This software cleverly combines recent web technologies to implement a miner that efficiently operates on all major browsers. Although originally developed for benign purposes, CoinHive has been maliciously injected into several websites [e.g., 10, 15]. In August 2018, a vulnerability in MikroTik routers has been used to inject a cryptojacking script into traffic passing through more than 200,000 of these routers [4].

In this paper, we present a large-scale study on web-based cryptojacking. While previous work has anecdotally described this phenomenon [12], discussed detection approaches [21] and investigated a subset of the Alexa ranking [18], we systematically investigate the prevalence of mining scripts in the Alexa Top 1 million websites. To this end, we have instrumented a browser to monitor the execution of code during the visit of a website and spot indications of mining activity, such as an unusual CPU utilization, the excessive repetition of functions and the presence of suspicious scripts. In summary, our study provides the following key insights:

**(a) Web-based cryptojacking is not rare.** We observe that 1 out of 500 websites in the Alexa ranking contains a web-based miner that immediately starts once the website is visited. While the JavaScript code driving the mining is diverse and often obfuscated, we observe that almost all miners employ similar WebAssembly code from the CoinHive project. We credit this finding to the CryptoNote protocol [38] that is implemented by the CoinHive miner and can support different currencies with minor modifications.

**(b) Mining profits are moderate.** Our analysis further provides a glimpse at the ecosystem of current cryptojacking. Several attackers operate on different websites using the same wallet or API key. Based on the configuration of typical desktop computers

*ARES '19, August 26–29, 2019, Canterbury, United Kingdom*
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-7164-3/19/08…$15.00
https://doi.org/10.1145/3339252.3339261

and statistics about website visits, we estimate the revenue generated by individual miners in the Alexa ranking at a range of a few cents up to 340 USD per day under the price of the respective cryptocurrencies at time of our measurements.

**(c) Existing defenses are insufficient.** We investigate the effectivity of current defenses against cryptojacking, such as blacklists and browser extensions. While these defenses provide sufficient protection from known mining sites, such as CoinHive and CryptoLoot, the underlying static detection patterns are ineffective against customized variants of the mining code. We thus argue that better protection from web-based mining is needed and, in addition to static matching, also run-time analysis needs to be considered to reliably track down mining activity. MineSweeper [21], for instance, is a promising alternative that provides detection based on characteristics of cryptomining code.

## 2 WEB-BASED MINING

Cryptocurrencies are a specific type of electronic money that provide decentralized control using the concept of blockchains [27]. In contrast to other electronic currencies, individuals generate revenue by solving computational puzzles and thereby validating transactions—a process referred to as *mining*. While mining of classic cryptocurrencies, such as Bitcoin and Ether, requires specific hardware to be profitable, memory-bound currencies and novel web standards have paved the way for effectively mining within web browsers. In the following, we review these changes and discuss their impact on cryptojacking.

### 2.1 Memory-bound Cryptocurrencies

Classic cryptocurrencies, such as Bitcoin and Ether, build on proof-of-work functions (computational puzzles) that are *CPU-bound*, that is, the effectivity of mining mainly depends on the available computing power [5]. Hardware devices designed for demanding computations, such as GPUs and ASICs, thus provide a better mining performance than common CPUs. As a consequence, profitable mining of classic cryptocurrencies has become largely infeasible with regular desktop and mobile computer systems.

*Memory-bound functions.* This situation has not been anticipated in the original design of the first cryptocurrencies and violates the "one-CPU-one-vote" principle underlying Bitcoin mining [27]. As a remedy, alternative cryptocurrencies have been developed in the community that make use of memory-bound functions for constructing computational puzzles. One prominent example is the cryptographic mixing protocol *CryptoNote* [38] and the corresponding proof-of-work function *CryptoNight* [34].

CrypoNight is a hash function that determines the hash value for an input object by extensively reading and writing elements from a 2 Megabyte memory region. This intensive memory access bounds the run-time of the function and moves the overall mining performance from the computing resources to the available memory access performance. As memory access is comparably fast on common CPUs due to multi-level caching, CryptoNight and other memory-bound proof-of-work functions provide the basis for alternative cryptocurrencies that can be efficiently mined on regular desktop systems and hence are a prerequisite for realizing web-based miners.

*CryptoNote-based currencies.* The idea of memory-bound proof-of-work functions along with other improvements over the original Bitcoin protocol has spawned a series of novel cryptocurrencies, each forking the concept of CryptoNote. Prominent examples are Monero [XMR, 36], Bytecoin [BCN, 6], and Electroneum [ETN, 11], which reach a market capitalization between 226 million and 3.8 billion USD [8]. These currencies share the underlying CryptoNote protocol and thus can be easily implemented with the same code base. Moreover, due to the concept of anonymous transactions they provide more privacy than Bitcoin and may conceal the identity of senders and receivers [see 23, 24].

Both properties—profitable mining on desktop systems and the availability of different currencies following the same cryptographic protocol—render these currencies an ideal target for web-based mining. Furthermore, the increased privacy of transactions provides a basis for conducting cryptojacking over manipulated web sites. According to our findings, CryptoNote-based currencies are currently prevalent in web-based mining and play a major role in cryptojacking as detailed in Section 4.

### 2.2 Novel Web Standards

The decentralized nature of cryptocurrencies imposes constraints on the capabilities of mining clients. First, the clients need to efficiently communicate with each other to synchronize the solving of puzzles. Second, the clients require programming primitives that enable an optimal utilization of available hardware resources.

At a first glance, these requirements seem to contradict with classic web technology, as the underlying HTTP protocol induces a non-trivial overhead and supported scripting languages, such as JavaScript and ActionScript, do not provide efficient primitives for low-level programming. However, browser vendors and the W3C have continuously advanced web standards and developed additional functionalities. In combination, *WebSockets*, *WebWorkers* and *WebAssembly* provide a fruitful ground for web-based mining of cryptocurrencies.

*WebSockets.* The WebSocket protocol has been standardized as additional browser functionality in 2011 [14] and is supported by all major browsers as of now. The protocol enables full-duplex communication from the browser to a web server with less overhead than HTTP. From the network perspective, the protocol is a classic application-layer protocol that operates on top of the transport layer. From the web application's point of view, however, WebSockets rather provide a transport protocol that enables transferring arbitrary payloads.

In the context of web-based mining, WebSockets allow the efficient communication between miners through a web server and thus are an integral part of currently available implementations. However, WebSockets are also used in several other types of web applications, like chats and multiplayer games, and thus represent only a weak indicator of mining activity.

*WebWorkers.* A second addition are so-called WebWorkers which have been introduced in 2015 [17] and are also supported by all major browsers. This programming primitive enables JavaScript code to schedule multiple threads and conduct concurrent computations in the background. While the original programming model underlying JavaScript already supports event-driven concurrency, orchestrating the available computing resources, such as multiple cores, has been technically involved. This problem is alleviated with WebWorkers, where the number of concurrent threads can be scaled with the available processor cores easily.

Although WebWorkers are not strictly necessary for implementing web-based mining, they allow for better utilizing the available resources and thus can also be found in most implementations. For our study, we hence consider the presence multiple of WebWorker threads as an indicator for potential mining activity.

*WebAssembly.* The previous two functionalities ease the communication and scheduling of web-based miners. Yet they are not sufficient for realizing an efficient implementation, as the underlying JavaScript code requires a costly interpretation within the browser. This problem is addressed by the WebAssembly standard from 2017 [31]. The standard proposes a low-level bytecode language that is a portable target for compilation of high-level languages, such as C/C++ and Rust. WebAssembly code, or Wasm code for short, is executed on a stack-based virtual machine in the browser and improves the execution as well as loading time over JavaScript code [16]. WebAssembly is currently supported by Chrome, Safari, Firefox and Edge[1].

WebAssembly is a perfect match for implementing mining software, as it enables compiling cryptographic primitives, such as specific hash functions, from a high-level programming language to low-level code for a browser. As an example, Figure 1(a) shows a simplified snippet of C code from the cryptographic hash Skein [13]. The corresponding WebAssembly code is presented in Figure 1(b) as raw bytes and instructions. Note that the instructions do not contain any registers, due to the stack-based design of the virtual machine. The characteristic constant of the Skein hash, which here is encoded in LEB128 format—a variable-length representation of integer numbers [see 37]—is visible in line 5.

```
1  int64_t k18 = k16 ^ k17 ^ 0x1bd11bdaa9fc1a22;
```

(a) Simplified C code snippet from the Skein hash.

```
1  00: 20 10                        ; get_local 16
2  02: 20 11                        ; get_local 17
3  04: 85                           ; i64.xor
4  05: 42                           ; i64.const
5  06: a2 b4 f0 cf aa fb c6 e8 1b   ; i64 literal
6  0e: 85                           ; i64.xor
7  0f: 21 12                        ; set_local 18
```

(b) Corresponding WebAssembly code as raw bytes and instructions.

**Figure 1: Example of C and WebAssembly code.**

## 2.3 The CoinHive Miner

The first implementation of a mining software based on the aforementioned developments was released as the *CoinHive* miner in September 2017. It originally had been developed for a popular image board as an alternative payment mechanism [22]. Several variants that, similar to CoinHive, all implement the CryptoNote protocol have been developed ever since, including JSECoin and CryptoLoot. Although these implementations differ in some details, they share how WebSockets, WebWorkers and WebAssembly are used for efficient web-based mining.

The miner itself is distributed via a single JavaScript file, which the website's owner includes on the page along with a small snippet to configure and start the mining process. The snippet and its configuration may be further customized, e.g., to not execute on mobile devices, but at least requires a unique id that maps miners to identities—in the case of CoinHive, so-called *site-keys*—in order to account payouts for calculated hashes. Due to this additional indirection it usually is not possible to link miners, identified by site-keys, to specific wallet addresses. Moreover, each account may be associated with multiple site-keys, such that multiple mining sites may in fact mine for the same wallet without disclosing the fact to the public.

## 2.4 Cryptojacking

Web-based mining certainly has legit use-cases and may pose an alternative to online advertisements as scheme of monetization. Moreover, mining might even replace CAPTCHAs used for rate limitation by requiring a proof-of-work. The anonymity offered by cryptocurrencies in combination with simple deployment on the web, unfortunately, also attracts actors with less noble goals. As the effort and cost of including a miner in an existing website is negligible, all it needs is access to a frequently visited website. Recently, a variety of incidents involving mining scripts have been reported for popular websites [33, 41].

We define cryptojacking as the practice of automatically starting a web-based miner upon visiting a web page. For this, we neither consider the disclosure of the mining process to the user nor the presence of an opt-out mechanism relevant. We view a consent after the fact as an inadmissible mode of operation, similarly to how the GDPR now requires a “clear affirmative action” for tracking cookies in the EU [9]. Miners that only run after explicit consent by the user, such as Authedmine and JSEcoin, are not considered part of the problem and are thus not examined in our study. To conclude that a website employs cryptojacking, we further do not differentiate between scripts added by the website's owner and scripts injected by a third party by means of hacking the server or hijacking included scripts.

## 3 IDENTIFICATION OF WEB-BASED MINERS

Based on the discussed background, we proceed to present our systematic study of cryptojacking on the web. The goal of this study is to evaluate to which degree the recent level of hype is justified through painting a comprehensive picture of the current cryptojacking practices in the wild. To this end, we measure the prevalence of cryptojacking in today's web (Section 3.4) and examine the effectiveness of the current generation of dedicated anti-cryptojacking countermeasures (Section 3.5). After introducing our approach and documenting our experiments, we address these topics in detail.

[1]Statistics from https://caniuse.com/#feat=wasm, May 2018

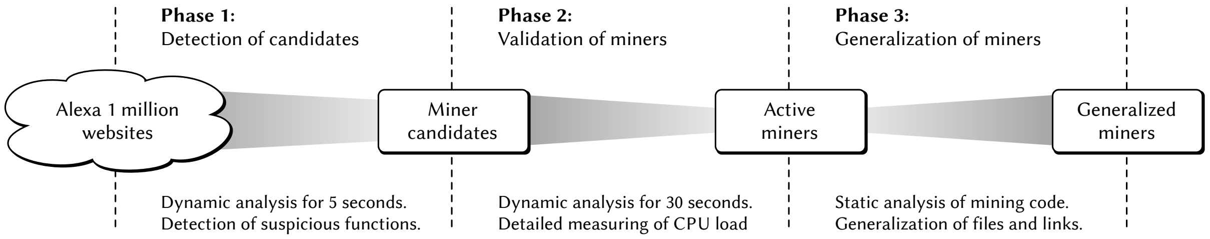

Phase 1:
Detection of candidates
Phase 2:
Validation of miners
Phase 3:
Generalization of miners
Alexa 1 million websites
Miner candidates
Active miners
Generalized miners
Dynamic analysis for 5 seconds.
Detection of suspicious functions.
Dynamic analysis for 30 seconds.
Detailed measuring of CPU load
Static analysis of mining code.
Generalization of files and links.

**Figure 2: Overview of our approach for identification of web-based miners.**

## 3.1 General Approach

For conducting our study, an empirical method is required that is accurate in its detection capabilities while being scalable to enable the analysis of large numbers of real-world websites. To accommodate these requirements, we designed a dedicated cryptojacking detection process that spans three individual phases: 1) An over-permissive first broad sweep to identify potential miner *candidates* using heuristics, 2) a thorough run-time analysis to isolate the real miners within the candidate set, and 3) a generalization step, in which we extract static indicators, that allow the identification of non-active or stealth mining scripts (see Figure 2).

*Phase 1: Detection of candidate sites.* In the first phase, our approach conducts a fast and imprecise initial analysis of websites to create a pool of candidates which likely—but not necessarily—host a mining script. To do so, we compiled a set of heuristics that hint the potential presence of a cryptojacking script and that can be measured at run-time while rendering a webpage in a browser. These heuristics were extracted from a manual analysis of verified mining scripts. For one, we initiate a short profiling of the site's CPU usage, with unusual high CPU utilization levels being interpreted as an indicator for mining. Furthermore, we mark all sites as suspicious, that use miner-typical web technologies, which are not in widespread use in the general web, namely WebAssembly or non-trivial amounts of WebWorkers. If at least one of these indicators could be found in a site, this site is marked as a potential *mining candidate*. Thus, the result of this phase is an over-approximation of the set of actual mining sites.

*Phase 2: Validation of mining scripts.* For obvious reasons, none of the used heuristics is conclusive in the identification of miners, as there is a multitude of legitimate reasons to use WebWorkers, WebAssembly or temporary high amounts of computation. However, the constant and potentially unlimited usage of CPU, caused by a *single function* within parallelized scripts is a unique phenomenon of cryptojacking. Thus, in the second phase we conduct a significantly prolonged run-time analysis of the candidate sites, in which the sites receive *no* external interaction and hence should be idle after the initial rendering and set-up in legitimate cases. However, if once the page is loaded, all JavaScript is initialized, and the DOM is rendered, the CPU usage still remains on a high level and the computation load is the result of repetitive execution of a single function within the webpage's code base, we conclude that the site hosts an *active* cryptojacking script.

*Phase 3: Generalization of miner characteristics.* The run-time measurements of the first two phases limit our approach to the detection of *active* mining scripts. We thus might miss mining sites that are inactive at the time of the test, for instance due to programming errors in the site's JavaScript code, a delayed start of the mining operation, or mining scripts waiting for external events (e.g., initial user interaction with the page). To create a comprehensive overview, it is important to identify these sites to document the *intent of mining*. To this end, we leverage the results of the second phase to generate a set of static features of mining scripts which we can be used to reevaluate the data from phase 1.

To this end, we first extract the JavaScript code from the validated mining sites that is responsible for initiating and conducting the mining operations. From this script code, we take both the URL and a hash of its contents as two separate features. Furthermore, we collect all parsed WebAssembly functions, sort them and use the hash of the whole code base as the third feature. We then apply each feature to our list of confirmed miners from the previous phase and keep only those that describe at least a certain number of miners. As the result, we obtain a set of generalized fingerprints, which can identify common mining scripts even in their *inactive* state. Applying these onto the data collected during the first phase in combination with our list of confirmed miners from the second phase yields to total number sites with a web-based miner.

## 3.2 Implementation

We implemented the previously outlined approach into our custom web crawler. In the following, we cover some details of the implementation, which enable us to obtain accurate results on a large scale in a real-world environment.

*3.2.1 Instrumented browser.* We use a normal browser to visit all websites, in this case *Google Chrome*. This ensures that we will execute all scripts and support all modern features needed to run a miner in the browser (see Section 2.2). By starting Chrome in headless mode, we can run many instances simultaneously without the overhead of a GUI. Our crawler is written in NodeJS and controls each instance via the *DevTools Protocol* [7], which allows us to instrument the browser and extract all necessary data.

*3.2.2 Fake number of cores.* The number of logical cores of a visitor's CPU is exposed in JavaScript via the `hardwareConcurrency` property of the global `navigator` object. This allows scripts to adjust the number of concurrent WebWorkers according to the available hardware and is used by miners to start the desired number of threads (usually one per core). However, we do not want a single mining site to seize all available resources on our server and interfere with simultaneous visits of other websites. Furthermore, websites might employ checks on the number of cores and not run if an unusually high number is observed, thus preventing us from detecting them. Changing the returned value can be achieved by injecting a script into each document that overwrites the property before executing any other script content.

*3.2.3 CPU Profiling.* Most importantly, instead of using standard Unix tools to measure the CPU load on a *per-process* level, we utilize the integered profiler of Chrome's underlying JavaScript engine V8 to measure the load on a *per-function* level. The profiler pauses the execution at a regular interval and samples the call stack, which enables us to estimate the time spent in each executed function. This way, we can not only determine if a *single function* consumes a considerable amount of CPU time, but also pinpoint the responsible script in the website's code.

In order to achieve this, we aggregate the collected data for each unique call stack. Wasm code itself cannot be profiled on a function level, so all samples of it are just named `<WASM UNNAMED>`. However, from the call stack we can still see how much time the Wasm code took and trace it back to the JavaScript function which caused the call into Wasm in the first place (see Table 1). By comparing the time spent in a function with the length of the profiling, we can estimate the caused CPU load for that particular function. Note that if the same code is running in several workers simultaneously, the combined time from all workers can be as high as the number of cores times the length of the profiling, e.g., our profiling for 5,000 milliseconds with 4 CPU cores in phase 1 can result in a maximum time of 20,000. Thus, taking the value of 14,375 from Table 1 as an example, would mean this function generated a load of approximately 72 %.

**Table 1: Example of a call stack with the aggregated amount of samples and time spent for each of its functions.**

| Function name | # Samples | Time in ms |
|---|---|---|
| `<WASM UNNAMED>` | 73,938 | 14,375.3 |
| `Module._akki_hash` | 1 | 0.1 |
| `CryptonightWASMWrapper.hash` | 4 | 0.6 |
| `CryptonightWASMWrapper.workThrottled` | 11 | 1.8 |
| `(root)` | 0 | 0.0 |

## 3.3 Experimental Setup

We used the aforementioned implementation to find instances of web-based cryptojacking in the wild. The following paragraphs briefly discuss the key parameters of our experimental setup for each of the three phases.

*Phase 1.* We conducted our study on the Alexa list of the top 1 million most popular sites[2]. We visit the front page of each site and wait until the browser fires the load event or a maximum of 30 seconds pass. Furthermore, to allow for sites that dynamically load further content, we wait an additional 3 seconds or until no more network requests are pending. We then start the CPU profiler and measure all code execution for 5 seconds and flag the site as suspicious, if there is a function with more than 5 % load on average. As the most CPU-heavy function on each website of the Alexa Top 1 million had an average load of only $0.2\,\% \pm 3.21$, we reckon that a value of 5% or more warrants further investigation. We also flag the site for extendend analysis, if either any Wasm code or more than 3 workers are used, which is equal or more than all the CPU cores we pretend to have. For this phase, we used a single server with 24 CPU cores and 32 GB of RAM running 24 simultaneous crawlers backed by Chrome v67.0.3396 over a time span of 4 days.

*Phase 2.* The detailed verification of suspicious sites uses the same general setup as the first phase. However, here we only run one crawler on a smaller server with 8 CPU cores. By visiting the websites one-by-one and profiling for a longer time of 30 seconds, we can more accurately determine if a website contains a mining script. If there is one function in the code base that results in an average load of 10 % or more, we label it as a confirmed and active miner. We argue that while a value lower than 10% certainly would make the miner very hard to detect, it also severly thwarts the ability to make money with cryptojacking. Furthermore, such slow mining does not even seem to be supported by popular mining scripts, as we will describe shortly.

*Phase 3.* In the final step, we create the fingerprints as outlined in Section 3.1 using the code of the confirmed miners. However, we only keep the fingerprints shared by at least 1 % of all miners. This restrictive measure ensures that only mining scripts with multiple validated instances produce fingerprints and, thus, avoids accidental inflation of potential phase 2 classification mistakes. The resulting fingerprints are then applied to the collected data from the first phase, yielding the final number of websites employing cryptojacking on their visitors.

To validate that our implementation and setup are working as intended, we created a testbed with the two popular implementations that start without the user's consent: CoinHive and CryptoLoot. This testbed consists of 24 locally hosted pages, which each contain one of the miners at a different throttling levels between 0% and 99%. Interestingly, even if the miner is configured with a throttle as high as 99%, so that it should utilize only 1% CPU, we can confirm it as a miner with our 10% threshold. Looking into the implementation of the throttling, we find that the code never sleeps for longer than two seconds between hash calculations, which makes it impossible to actually use very low throttling values. We also confirm this by monitoring Chrome's CPU usage with `htop` and find that no matter how high we set the throttling, the load on our machine never drops to below 20%. As our implementation is able to successfully detect all miners in the testbed, regardless of the used throttling value, we are confident its ability to find active cryptojacking scripts.

## 3.4 Prevalence

As first result of our crawling, we identify 4,627 suspicious sites in the Alexa ranking using the methodology and parameters outlined in the previous sections. Out of these, 3,028 are flagged for having a load-intensive function, 3,561 for using at least as many workers as CPU cores we pretend to have and 2,477 for using Wasm. Note

[2]http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

that these sets overlap, as for example the usage of Wasm often implies a CPU intensive application.

The detailed analysis of these 4,627 suspicious sites results in 1,939 sites with a continuously high CPU usage over a profiling for 30 seconds. We use the resulting set of scripts for the third phase to build fingerprints of the most popular miners, resulting in 15 hashes of JavaScript code, 12 hashes of Wasm code bases, and 8 script URLs. The latter can be found in Table 3. After applying these fingerprints, we obtain the final number of 2,506 websites, which are very likely to employ cryptojacking. Table 2 summarizes our results.

**Table 2: Prevalence of miners in the Alexa Top 1 million.**

| Phase | Result | # Websites | % of Alexa |
|---|---|---|---|
| 1 | Suspicious sites | 4,627 | 0.46 |
| 2 | Active cryptojacking sites | 1,939 | 0.19 |
| 3 | Total cryptojacking sites | 2,506 | 0.25 |

**Table 3: Common script URLs responsible for the creation of the mining workers, which resulted in fingerprints**

| URL | # Occurrences |
|---|---|
| //coinhive.com/lib/coinhive.min.js | 656 |
| //advisorstat.space/js/algorithms/advisor.wasm.js | 311 |
| //www.weather.gr/scripts/ayh9.js | 68 |
| //aster18cdn.nl/bootstrap.min.js | 59 |
| //cryptaloot.pro/lib/crypta.js | 46 |
| //gninimorenom.fi/sytytystulppa.js | 35 |
| //coinpot.co/js/mine | 27 |
| //mepirtedic.com/amo.js | 22 |

During manual investigation of a sample of the additional 567 sites only detected in phase 3, we found five reasons why our dynamic analysis missed these miners: (1) A script for web-based mining is included, but the miner is never started. (2) The miner only starts once the user interacts with the web page or after a certain delay. (3) The miner is broken—either because of invalid modifications or because the remote API has changed (as it was the case for CoinHive earlier this year). (4) The WebSocket backend is not responding, which prevents the miner from running. (5) The miner is only present during some visits, e.g., to hinder detection or due to ad banner rotation

This analysis confirms the need for a three-step identification process, as only the combination of phase 2 and 3 enable us to determine a comprehensive picture of current cryptojacking in the websites of the Alexa ranking.

## 3.5 Effectiveness of Countermeasures

To both compare our findings to existing approaches for the detection of cryptojacking and to validate our results, we select three popular solutions to block miners in the browser. For one, we use the *NoCoin adblock list*[3], which is a generic list for adblockers, such as Adblock Plus or uBlock Origin and is now also used by Opera's built in adblocker. For the remainder of this section, we refer to this list as *Adblocker*. Furthermore, we include the blacklists used by the two most popular Chrome extensions with the purpose of blocking web-based miners: *No Coin*[4] with 566,692 users and *MinerBlock*[5] with 161,630 users. We extract the detection rules these extensions contain and translate them into SQL statements while preserving the wildcards, in order to apply them to the data collected during our crawl of the Alexa Top 1 million sites. The number of identified miners for each system are presented in Table 4 in the first column. The other columns of this table compare these results for each system with the 2,506 websites we identified as miners. The second column reports on the intersection of both lists, that is the number of sites on which both approaches are in agreement. Accordingly, the last two columns each contain the number of sites that one approach reported, but not the other.

**Table 4: Detection results of our approach and three common blacklists as absolute numbers.**

| Blacklist | # Detections | # Both | # Only they | # Only we |
|---|---|---|---|---|
| Minerblock | 1,599 | 1,402 | 197 | 1,104 |
| No Coin | 1,217 | 1,039 | 178 | 1,467 |
| Adblocker | 1,136 | 1,049 | 87 | 1,457 |

Unsurpisingly, our approach mixing static and dynamic analysis clearly outperforms the three static blacklists and spots a considerable amount of additional web-based miners. Moreover, the large overlap in sites that both we and the extensions found, validates that our approach and shows that it is indeed suitable to detect cryptojacking in the wild.

There are, however, a few sites that our approach misses, but the blacklists detect. Manual analysis of a subset showed that besides overly zealous lists, the main reason is that we can only learn fingerprints of *active* miners. For example, some website owners copied CoinHive's script to host it on their own servers a few months ago. Meanwhile, all these mining scripts stopped working, as CoinHive changed its API used in the communication with the pool. Therefore, while this probably represents a cluster of inactive miners, we are unable to detect them, as no fingerprint for any of the scripts could be generated in the third phase, due to the fact that the *whole* cluster was inactive at the time of analysis.

The existing blacklists on the other hand can detect them, as their rules are curated by humans, which allows them to apply a couple of generic measures. For example, most blacklists include a rule for `*/coinhive.min.js`. In contrast, our static indicators are generated in a fully automated fashion, based on code characteristics from dynamically validated miner instances. In this process, we cannot generalize our list of fingerprinted full script URLs towards partial URLs or even only filenames without manual review, as this could lead to misclassifications. For instance, in our dataset such an attempt would end up in all scripts named `*/bootstrap.min.js` being blacklisted because a widespread mining script uses this benign-sounding name (see Table 3).

[3]https://github.com/hoshsadiq/adblock-nocoin-list/

[4]https://chrome.google.com/webstore/detail/gojamcfopckidlocpkbelmpjcgmbgjcl

[5]https://chrome.google.com/webstore/detail/emikbbbebcdfohonlaifafnoanocnebl

## 4 REVENUE ANALYSIS

After identifying web-based miners in the Alexa ranking, we proceed to analyze the efficacy of these miners in detail. For this analysis, we focus on all *active miners* discovered in phase 2 of our study, that is, websites that immediately begin mining when visited by a browser. First, we describe how we have chosen important parameters for the following estimates (Section 4.1). Based on this, we estimate the average profit of for all the 1,939 active mining websites and answer the question of whether such mining can generate significant income (Section 4.2). We then investigate how aggressively these web pages stress the visitors' CPUs to shed light on the stealthiness of current cryptojacking in the web (Section 4.3). Finally, we determine how many different implementations of miners exist in our dataset and whether these differ due to customization and obfuscation (Section 4.4).

### 4.1 Estimation Parameters

Determining the exact revenue of web-based miners is a non-trivial task, as the profit depends on several factors, such as the popularity of a website, its content, the visitor's hardware as well as the current price of the cryptocurrency. For our analysis, we thus focus on the CoinHive library and measure its run-time performance for different desktop and mobile CPUs. Results of this experiment are shown in Table 5, where the performance is presented in hashes per second for one core and the entire CPU. Due to the memory-bound proof-of-work function, the hash rate varies only slightly between the different CPU models when executed on one core. The only exception is the HiSilicon CPU whose cache is limited to 2 MB and thus is not suitable for computing the CrypoNight hash. We thus assume a rate of 80 H/s for an average CPU. Furthermore, we use the payout rate at time of our measurement crawl[6].

**Table 5: Performance of different CPUs with CoinHive.**

| **CPU model** | **Cache size** | **Hashes/s** | |
|---|---|---|---|
| Product name and clock speed | L2/L3 | Core | CPU |
| Intel® Xeon® E5-1650 v3 @ 3.50 GHz | 15 MB | 22.2 | 148.9 |
| Intel® Core™ i7-7700K @ 4.20 GHz | 8 MB | 21.4 | 115.3 |
| Intel® Core™ i7-6820HQ @ 2.70 GHz | 8 MB | 23.2 | 90.2 |
| Intel® Core™ i7-5557U @ 3.10 GHz | 4 MB | 21.1 | 35.5 |
| Apple A11 Bionic APL1W72 | 8 MB | 16.0 | 35.1 |
| HiSilicon Kirin 620 @ 1.20 GHz | 2 MB | 2.0 | 11.6 |

### 4.2 Average Revenue

We proceed to estimate the expected revenue for the active miners identified in Section 3. In particular, we make use of the SimilarWeb service to quantify the number of visits as well as the average duration for the websites hosting the miners. The results of this analysis are shown in Table 6, where we include the 10 most profitable sites identified during our analysis. These sites are able to generate between 0.53 and 1.51 XMR per day, that is, 119 to 340 USD. Given that the revenue is achieved without the consent of the visitors and visual indications, this is still a notable profit. However, we conclude that current cryptojacking is not as profitable as one might expect and the overall revenue is moderate.

[6]According to https://coinmarketcap.com/, 1 XMR =225 USD in May 2018

**Table 6: Visiting statistics for the top-10 sites containing miners.**

| | **Visitors** | **Duration** | **Core hours** | **Revenue*** |
|---|---|---|---|---|
| | per day | per visit | per day | XMR per day |
| cinecalidad.to | 1.3 M | 4'10" | 89 K | 1.5 |
| mejortorrent.com | 0.8 M | 4'30" | 60 K | 1.1 |
| kinokrad.co | 1.3 M | 2'29" | 54 K | 1.0 |
| ianimes.co | 0.2 M | 13'07" | 39 K | 0.7 |
| india.com | 1.3 M | 1'27" | 32 K | 0.6 |
| ddmix.net | 0.4 M | 5'06" | 38 K | 0.6 |
| seriesypelis24.com | 0.4 M | 5'22" | 35 K | 0.6 |
| seriesblanco.com | 0.4 M | 6'06" | 36 K | 0.6 |
| ekinomaniak.tv | 0.2 M | 10'10" | 33 K | 0.6 |
| kickass.cd | 0.3 M | 5'24" | 30 K | 0.5 |

* Estimated based on CoinHive's payout ratio[6] and 80 H/s.

We continue to asses the average profit made by mining on the web. To this end, we inspect the distribution of visits per day and the average duration of these in Figure 3. The websites with the largest outreach in our dataset (`cinecalidad.to`) has 1.3 million visits. A different site (`ianimes.co`) attracts less visitors, but engages them to stay 13 minutes on the web page. On average these websites attract 24,721 visitors per day and keep them for roughly 3 minutes on average. Overall, we thus observe a range of 0.17 to 89,000 core hours, with a mean of 1,550 core hours. With a hash rate of 80 H/s and CoinHive's payout ratio[6], a miner earns about 5.8 USD per day and website on average, which supports our observation that web-based cryptojacking currently provides only limited profit.

Next, we group websites that make use of the same site-key to calculate the overall revenue of mining entities. Tracking these relations provides valuable insights on the landscape of cryptojacking, as we can identify attackers that deploy miners on multiple websites. Note, that this analysis is not limited to CoinHive, but applies to any variant using the original implementation. We thus observe a large variety of unique site identifiers. A few instances make use of nondescript values such as `X` or `abc`, though, which we filter out for this particular measurement. Figure 4 depicts the frequency of websites per site-key in bins of 5.

Our analysis shows that site-keys are indeed reused across different websites. Few cryptojackers even pool forces across up to 40 to 55 websites, while the majority of attackers appears to act on their own or, more likely, utilize CoinHive's function of aggregating multiple site-keys with one account. Moreover, we observe that

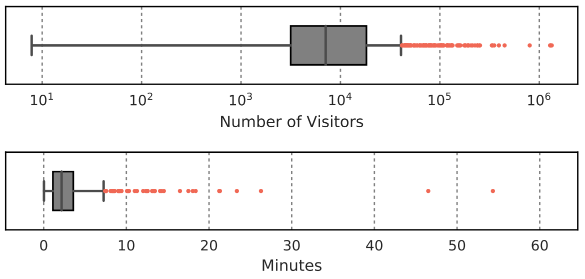


**Figure 3: Distribution of visits to web pages identified to use web-based miners (top) and the duration per visit (bottom).**

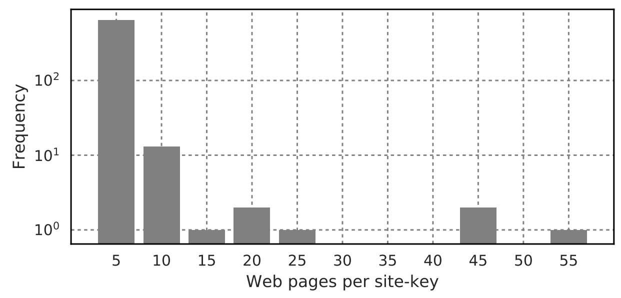


**Figure 4: Distribution of websites per site-key.**

23 web pages make use of at least two site-keys. These websites thus connect clusters of miners.

## 4.3 Greediness vs. Stealthiness

The revenue of a cryptojacking campaign may vary a lot, depending on how aggressive the miner occupies the visitor's CPU cores. Using large amounts of processing power earns the most money, but simultaneously may raise suspicion due to an unresponsive computer and audible fan noise. An attacker thus has to strike a balance between profit and stealthiness in practice.

Many popular implementations of web-based miners allow for the configuration of a throttling value. While CoinHive's default value is 0, their example recommends a value of 0.3, which means the miner only uses 70 % of the available computing power by constantly monitoring the current and maximum possible hash rate and idling or mining accordingly. The data we have gathered in *Phase 2* for the validation of miners allows us to approximate the CPU load and thus also the throttling value chosen by the website operator.

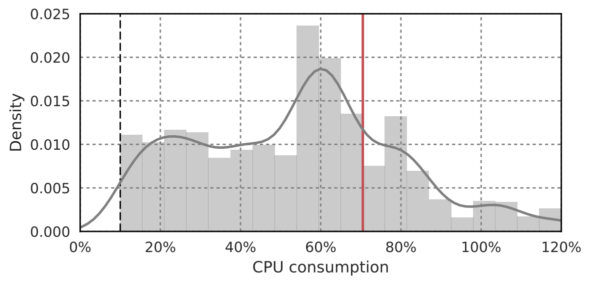


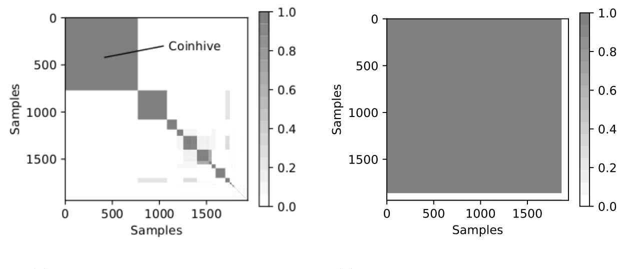
**Figure 5: CPU consumption of cryptojacking web pages.**

The results of our analysis are shown in Figure 5. The most popular setting is in the range 50–70 %. Interestingly, about 5 % of the websites attempt to even use up more CPU cores than available on the system. This suggests that some attackers are especially greedy in their attempts to max out the available processing power by starting more mining threads than the CPU can handle.

## 4.4 Code Diversity

As the last step, we analyze the diversity of the JavaScript and WebAssembly code in the identified miners. For JavaScript, we can identify the script responsible for mining by inspecting the CPU profiling. For the WebAssembly code, on the other hand, we have to first concatenate all parsed functions into a single file using their SHA1 hashes for ordering, as we separately obtain these functions from the debugger. As a result of this preprocessing, we obtain one sample of JavaScript and one merged sample of WebAssembly code for each of the 1,939 websites containing miners. While this representation simplifies our experimental setup, it requires a fuzzy analysis, as minor perturbations in the merged files obstruct the application of exact matching.

We thus employ techniques from information retrieval that can cope with noisy data. In particular, we conduct an *n*-gram analysis, where the code samples are first partitioned into tokens using whitespaces and then mapped to a vector space by counting the occurrences of *n*-grams (sequences of *n* tokens) [see 32]. This vectorial representation enables us to compute the *cosine similarity* between all samples and generate the similarity matrices shown in Figure 6. The columns and rows of the matrices are arranged using hierarchical clustering, such that larger groups of similar code samples become visible.



(a) Similarity of JavaScript code (b) Similarity of WebAssembly code

**Figure 6: Similarity and cluster analysis of JavaScript and WebAssembly code found in web-based miners.**

For JavaScript, we identify 23 clusters of similar code. These clusters correspond to diverse implementations, including the CoinHive library (largest cluster) as well as modified and obfuscated code from different cryptojacking campaigns. By contrast, the WebAssembly code present in the 1,939 websites shows almost not diversity and is highly similar to the original CoinHive implementation. We credit this finding to minor modifications of the original code that allow for supporting alternative currencies or operating over less expensive mining pools.

In summary, we conclude that the current landscape of cryptojacking is dominated by variants of CoinHive. Although we spot different JavaScript code wrapping the miners, the low-level code is in almost all cases derived from a single implementation. Apparently, the cryptographic primitives underlying the CryptoNote protocol and in particular its proof-work-functions have only been once translated to a web implementation and hence all active miners in our study rest on the same foundation.

## 5 DISCUSSION AND LIMITATIONS

Our study provides the a comprehensive view on cryptojacking in the wild. Nonetheless, several of our results are estimates and approximations, as exact measurements are hardly possible in a dynamic system such as the Internet. In the following, we discuss threats to the validity of our empirical study and how they have been addressed in our implementation and experimental setup. Moreover, we pinpoint directions for extensions that can limit certain effects in future studies.

*False positives.* The core of this paper consists of a detection approach that aims to find cryptojacking scripts at run-time. This task comes with an inherent precision problem, as we try to determine the semantics of executed code from dynamic execution artifacts based on a set of heuristics and characteristics. Thus, even though the used set of indicators and heuristics are carefully chosen, based on thorough manual analysis of validated cryptojacking code, there are no formal guarantees that the approach really identifies mining scripts or does not accidentally misclassify some examined sites, for instance, due to computation-heavy, legitimate JavaScript code.

To examine the potential problem of misclassification, we use a similarity analysis on the resulting data set of JavaScript and Wasm code (see Section 4.4). While in the collected set of JavaScript a certain degree of variance exists, the vast majority of Wasm code exhibits an astonishingly high degree of similarity, with less than 4% of outliers. To further investigate the JavaScript code, we selected one random sample from each of the 23 clusters to represent that cluster. By combining manual static and dynamic analysis and searching for mining-specific strings and functions, we were able to confirm that each sample is indeed a cryptominer and thus the whole cluster is likely to contain only scripts used for cryptojacking.

*False negatives.* The study only provides a lower bound on the overall cryptojacking landscape, as we are aware of a set of scenarios, which are currently not covered by our methodology: First, we only visit the homepages without deeper crawling of the sites. Second, sites might deliberately delay the inclusion of the mining scripts in the web document. Similarly, sites could use a non-deterministic condition or require user interaction to start the miner. However, increasing the reach of the crawling process and extended examination times would in general address the majority of the potential problems. Also, periodic repetition of the experiment will lead to the eventual detection of unreliable or currently malfunctioning miners. We leave these measures to future work.

Regarding the use of evasions to prevent detection, it should be noted that techniques like delaying the start of the miner also come with a significant drawback: Unlike traditional malware infections, where the malware likely can achieve persistance on the system, web-based mining stops as soon as the user closes the browser tab. Thus, an attacker only has limited time to run the mining code and any delays will negativly affect his profits, making it less likely to encounter such techniques.

*Data analysis.* The results of the revenue estimations (Secion 4.2) directly rely on external data from SimilarWeb. Thus, the quality of the provided analysis depends on the quality of the external data. Especially, the revenue calculations rely on *estimated* figures that are compiled using proprietary methodologies. Thus, the impact of potential problems in the underlying data should be considered when interpreting the presented results.

## 6 RELATED WORK

The study by Eskandari et al. [12] was the first to provide a peek at the cryptojacking phenomenom. However, the study is limited to vanilla CoinHive miners, and the underlying methodology is unsuited to detect alternative or obfuscated mining scripts. More recently, Konoth et al. [21] searched the web for instances of drive-by mining and proposed a novel detection based on the identification of cryptographic primitives inside the Wasm code. Similarily, Wang et al. [39] detect miners by observing bytecode instruction counts, while Rodriguez and Posegga [30] use API monitors and machine learning.

Our work, on the other hand, uses a sampling profiler to detect busy functions and is thus more closely related to the work by Hong et al. [18]. However, we crawled the whole Alexa Top 1 Million, while their study was limited to the Top 100k. Furthermore, fluctuations in the Alexa lists and the short timespan of mining campaigns add uncertainty to previously presented results. Therefore, our study provides an additional, independent data point on this new phenomenon at a different point in time. These factors are also the reason why we decided against directly comparing the number of detections between the papers.

Furthermore, unauthorized mining of cryptocurrencies is not limited to web scenarios. For example, Huang et al. [19] present a study on malware families and botnets that use Bitcoin mining on compromised computers. Similarly, Ali et al. [2] investigate botnets that mine alternative currencies, such as Dogecoin, due to the rising difficulty of profitably generating Bitcoins. To detect illegitimate mining activities, either through compromised machines or malicious users, Tahir et al. [35] propose *MineGuard*, a hypervisor-based tool that identifies mining operations through CPU and GPU monitoring. Our study extends this body of work by providing an in-depth view of mining activity in the web.

From a more general point of view, cryptocurrency mining is a form of *parasitic computing*, a type of attack first proposed by Barabási et al. [3]. As an example of this attack, the authors present a sophisticated scheme that tricks network nodes into solving computational problems by engaging them in standard communication. Moreover, Rodriguez and Posegga [29] present an alternative method for abusing web technology that enables building a rogue storage network. Unlike cryptojacking, these attack scenarios are mainly of theoretical nature, and the authors do not provide evidence of any occurrence in the wild. In a general study of the WebAssembly ecosystem, Musch et al. [26] found that cryptojacking is not the only instance of malicious WebAssembly usage, as some websites used it to hide and obfuscate their code.

On a technical level, our methodology is related to approaches using high-interaction honey browsers [e.g., 20, 25, 28, 40], which are mainly utilized to detect attacks on the browser's host system via the exploitation of memory corruption flaws, a threat also known as *drive-by-downloads.* While our approach shares the same exploration mechanism—using a browser-like system to actively visit potentially malicious sites—our detection approach diverges, as the symptoms of browser-based mining stem from the exclusive usage of legitimate functionality, in contrast to drive-by-download attacks that cause low-level control-flow changes in the attacked browser or host system.

## 7 CONCLUSION

This study provides a comprehensive view on the threat of web-based cryptojacking. We show that approximately 1 out of 500 websites in the Alexa 1 million ranking contains a miner that immediately starts mining when visiting the website. This implies that falling victim to a cryptojacker is not a rare event, and a considerable amount of energy is drained as part of this activity every day. However, our estimate of the generated revenue shows that web-based cryptojacking is not as profitable as it seems and many miners attain only moderate payouts.

Nevertheless, cryptocurrencies enjoy great popularity and provide a lucrative playground for financial speculation. As a consequence, there is a need for effective countermeasures. Unfortunately, we show in our study that current detection mechanisms are insufficient to fend off this threat, as they rely on simple blacklists that fail to cope with the complexity of JavaScript and WebAssembly code. This complexity can only be tackled if defense mechanisms are tightly integrated into the browser, such that the resources available to a website can be monitored and regulated dynamically—irrespective of the execution environment and employed web standards. Ultimately, such protection might help to generally mitigate the threat of parasitic computing inherent to current web technology.

## ACKNOWLEDGMENTS

The authors would like to thank Martina Lindorfer and Herbert Bos for providing a draft of their related paper. Furthermore, the authors gratefully acknowledge funding from the German Federal Ministry of Education and Research (BMBF) under the project VAMOS (FKZ 16KIS0534) and FIDI (FKZ 16KIS0786K), and funding from the state of Lower Saxony under the project Mobilise.

## REFERENCES

[1] AdGuard Research. Cryptocurrency mining affects over 500 million people. And they have no idea it is happening. Website https://adguard.com/en/blog/crypto-mining-fever/, Oct. 2017.
[2] S. T. Ali, D. Clarke, and P. McCorry. Bitcoin: Perils of an unregulated global p2p currency. In *Security Protocols XXIII*, pages 283–293. Springer, 2015.
[3] A.-L. Barabási, V. W. Freeh, H. Jeong, and J. B. Brockman. Parasitic computing. *Nature*, 412:894–897, 2001.
[4] Bleeping Computer. Massive Coinhive Cryptojacking Campaign Touches Over 200,000 MikroTik Routers. Website https://www.bleepingcomputer.com/news/security/massive-coinhive-cryptojacking-campaign-touches-over-200-000-mikrotik-routers/, Aug. 2018.
[5] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *Proc. of IEEE Symposium on Security and Privacy*, pages 104–121, 2015.
[6] Bytecoin. Bytecoin (BCN) – Anonymous cryptocurrency based on CryptoNote. Website https://bytecoin.org, May 2018.
[7] ChromeDevTools. Chrome DevTools Protocol Viewer. Website https://chromedevtools.github.io/devtools-protocol/, May 2018.
[8] CoinMarketCap. CoinMarketCap – Market Capitalization of Cryptocurrencies. Website https://coinmarketcap.com/currencies/, May 2018.
[9] Council of European Union. Council regulation (EU) no 679/2016. Website https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679, Apr. 2016.
[10] W. de Groot. Cryptojacking found on 2496 online stores. Website https://gwillem.gitlab.io/2017/11/07/cryptojacking-found-on-2496-stores/, Nov. 2017.
[11] Electroneum Ltd. Electroneum – The mobile based cryptocurrency. Website https://electroneum.com, 2018.
[12] S. Eskandari, A. Leoutsarakos, T. Mursch, and J. Clark. A first look at browser-based cryptojacking. In *Proc. of IEEE Security and Privacy on the Blockchain Workshop*, 2018.
[13] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker. The skein hash function family. Version 1.1, 2008.
[14] I. Fette and A. Melnikov. The websocket protocol. RFC 6455 (Proposed Standard), Dec. 2011. URL http://www.ietf.org/rfc/rfc6455.txt. Updated by RFC 7936.
[15] D. Goodin. Now even YouTube serves ads with CPU-draining cryptocurrency miners. Ars Technica, Website https://arstechnica.com/information-technology/2018/01/now-even-youtube-serves-ads-with-cpu-draining-cryptocurrency-miners/, Jan. 2018.
[16] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. F. Bastien. Bringing the web up to speed with WebAssembly. In *Proc. of ACM SIGPLAN International Conference on Programming Languages Design and Implementation (PLDI)*, pages 185–200, 2017.
[17] I. Hickson. Web workers. W3C Working Draft, Sept. 2015. URL https://www.w3.org/TR/2015/WD-workers-20150924/.
[18] 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 *Proc. of ACM Conference on Computer and Communications Security (CCS)*, Oct. 2018.
[19] 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 *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2014.
[20] C. Kolbitsch, B. Livshits, B. Zorn, and C. Seifert. Rozzle: De-cloaking internet malware. In *Proc. of IEEE Symposium on Security and Privacy*, pages 443–457, 2012.
[21] R. K. Konoth, E. Vineti, V. Moonsamy, M. Lindorfer, C. Kruegel, H. Bos, and G. Vigna. An in-depth look into drive-by mining and its defense. In *Proc. of ACM Conference on Computer and Communications Security (CCS)*, Oct. 2018.
[22] B. Krebs. Who and What Is Coinhive? Website https://krebsonsecurity.com/2018/03/who-and-what-is-coinhive, Mar. 2018.
[23] A. Kumar, C. Fischer, S. Tople, and P. Saxena. A traceability analysis of monero's blockchain. In *Proc. of European Symposium on Research in Computer Security (ESORICS)*, pages 153–173, 2017.
[24] M. Möser, K. Soska, E. Heilman, K. Lee, H. Heffan, S. Srivastava, K. Hogan, J. Hennessey, A. Miller, A. Narayanan, and N. Christin. An empirical analysis of traceability in the monero blockchain. *Proc. of Privacy Enhancing Technologies Symposium (PETS)*, 2018.
[25] A. Moshchuk, T. Bragin, D. Deville, S. D. Gribble, and H. M. Levy. Spyproxy: Execution-based detection of malicious web content. In *Proc. of USENIX Security Symposium*, 2007.
[26] M. Musch, C. Wressnegger, M. Johns, and K. Rieck. New kid on the web: A study on the prevalence of webassembly in the wild . In *Proc. of Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2019.
[27] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, May 2009. URL http://www.bitcoin.org/bitcoin.pdf.
[28] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. All your iframes point to us. In *Proc. of USENIX Security Symposium*, pages 1–15, 2008.
[29] J. D. P. Rodriguez and J. Posegga. CSP & Co. Can Save Us from a Rogue Cross-Origin Storage Browser Network! But for How Long? In *Proc. of ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2018.
[30] J. D. P. Rodriguez and J. Posegga. Rapid: Resource and api-based detection against in-browser miners. In *Proc. of Annual Computer Security Applications Conference (ACSAC)*, 2018.
[31] A. Rossberg. Webassembly core specification. W3C First Public Working Draft, Feb. 2018. URL https://www.w3.org/TR/2018/WD-wasm-core-1-20180215.
[32] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1986.
[33] Scott Helme. Protect your site from Cryptojacking with CSP + SRI. Website https://scotthelme.co.uk/protect-site-from-cryptojacking-csp-sri/, Feb. 2018.
[34] "Seigen", M. Jameson, T. Nieminen, "Neocortex", and A. M. Juarez. Cryptonight hash function. CryptoNote Standard 008, Mar. 2008. URL https://cryptonote.org/cns/cns008.txt.
[35] R. Tahir, M. Huzaifa, A. Das, M. Ahmad, C. Gunter, F. Zaffar, M. Caesar, and N. Borisov. Mining on someone else's dime: Mitigating covert mining operations in clouds and enterprises. In *Proc. of International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, pages 287–310, 2017.
[36] The Monero Project. Monero: Home. Website https://getmonero.org/, May 2018.
[37] UNIX International. Dwarf debugging information format. Revision 2.0.0, 1993.
[38] N. van Saberhagen. Cryptonote v2.0. Technical report, CryptoNote, Oct. 2013.
[39] W. Wang, B. Ferrell, X. Xu, K. W. Hamlen, and S. Hao. Seismic: Secure in-lined script monitors for interrupting cryptojacks. In *Proc. of European Symposium on Research in Computer Security (ESORICS)*, 2018.
[40] Y.-M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2006.
[41] Wired. Your Browser Could Be Mining Cryptocurrency for a Stranger. Website https://www.wired.com/story/cryptojacking-cryptocurrency-mining-browser/, Sept. 2017.