OUTGUARD: Detecting In-Browser Covert Cryptocurrency Mining in the Wild

Amin Kharraz[†] Zane Ma[†] Paul Murley[†] Charles Lever[◇]
Joshua Mason[†] Andrew Miller[†] Nikita Borisov[†] Manos Antonakakis[◇] Michael Bailey[†]

[†]University of Illinois Urbana-Champaign [◇]Georgia Institute of Technology

ABSTRACT

In-browser cryptojacking is a form of resource abuse that leverages end-users' machines to mine cryptocurrency without obtaining the users' consent. In this paper, we design, implement, and evaluate OUTGUARD, an automated cryptojacking detection system. We construct a large ground-truth dataset, extract several features using an instrumented web browser, and ultimately select seven distinctive features that are used to build an SVM classification model. OUTGUARD achieves a 97.9% TPR and 1.1% FPR and is reasonably tolerant to adversarial evasions. We utilized OUTGUARD in the wild by deploying it across the Alexa Top 1M websites and found 6,302 cryptojacking sites, of which 3,600 are new detections that were absent from the training data. These cryptojacking sites paint a broad picture of the cryptojacking ecosystem, with particular emphasis on the prevalence of cryptojacking websites and the shared infrastructure that provides clues to the operators behind the cryptojacking phenomenon.

CCS CONCEPTS

• **Security and privacy → Browser security**; *Web application security*;

KEYWORDS

Browser Security; Web Security; Cryptojacking;

ACM Reference Format:
Amin Kharraz[†] Zane Ma[†] Paul Murley[†] Charles Lever[◇] Joshua Mason[†] Andrew Miller[†] Nikita Borisov[†] Manos Antonakakis[◇] Michael Bailey[†] [†]University of Illinois Urbana-Champaign [◇]Georgia Institute of Technology . 2019. OUTGUARD: Detecting In-Browser Covert Cryptocurrency Mining in the Wild. In *Proceedings of the 2019 World Wide Web Conference (WWW'19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3308558.3313665

WWW '19, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6674-8/19/05…$15.00
https://doi.org/10.1145/3308558.3313665

1 INTRODUCTION

The financial success of cryptocurrencies, which are valued at over 280 billion USD in market capitalization[1], has drawn the attention of malicious actors, who attempt to accumulate wealth through illegitimate means. One such method is cryptojacking, which entails hijacking of a victim's processing power to mine cryptocurrency without consent. Cryptojacking has been reported in several forms, including botnets [21, 55] and compromised cloud computing infrastructure [43] that mine cryptocurrencies to benefit the attackers.

Recently, a new strain of cryptojacking has emerged: in-browser cryptojacking, where JavaScript cryptocurrency miners are deployed through websites. In-browser cryptojacking can be enabled primarily by two developments. First, researchers designed specialized proof-of-work mining algorithms (e.g., Cryptonight [6]) that make CPU mining reasonably competitive. Second, in-browser JavaScript performance has been improved through new web standards (e.g., WebAssembly) and general increases in end-user computational power. These advances, coupled with the broad deployment capabilities of the web, have led to reports of in-browser cryptojacking on vulnerable websites [36], public Wi-Fi networks [10], ad-networks [37], and a popular news site [35].

Existing detection mechanisms for in-browser cryptojacking (henceforth simply referred to as cryptojacking) are nascent and primarily centered around blacklisting [1, 12, 49] and CPU usage profiling [45]. Blacklisting is a reactive technique that struggles to keep up with rapidly evolving web threats [32]. CPU usage profiling is imprecise and can mislabel online games, videos, and other processing intensive JavaScript web applications. As we demonstrate in Section 3.4.2, this method suffers from high false positive rates of up to 13% on a small-scale dataset. Recently, L1 cache measurements have been utilized to effectively detect cryptojacking incidents [30]. While this technique is an improvement over the status quo, it is limited by the need for manual, case-by-case reverse engineering of WebAssembly code. Furthermore, reliance on the state of the L1 cache is prone to noise from other running processes and would affect overall detection accuracy.

This paper presents OUTGUARD, an open-source[2] cryptojacking detection system that uses a machine learning classifier to identify cryptojacking at scale. As a first step to building OUTGUARD, we systematically constructed a labeled dataset by scanning the Alexa Top 1M domains with a specially instrumented browser and recording each site's resources, provenance, and JavaScript execution traces.

[1] Market cap for the top 100 cryptocurrencies as of July 2018, according to [11].
[2] Both code and data are available at https://github.com/teamnsrg/outguard

Applying existing cryptojacking detection tools to the scan data yielded a set of 3,006 cryptojacking execution traces from 12 different families. We assembled these cryptojacking traces into two datasets: a balanced dataset with equal numbers of cryptojacking and benign website traces, and an imbalanced dataset that more closely represents the real-world frequency of cryptojacking.

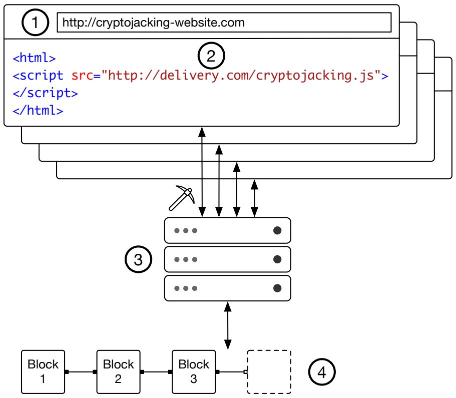
Starting with a wide range of features that pertain to JavaScript execution, network behavior, and DOM characteristics, we carefully analyzed the labeled data to ultimately narrow down to seven features in four feature categories: browser-based, mining-based, load-based, and network-based. We tested two different types of machine learning models, Support Vector Machines (SVM) and Random Forests (RF), and ultimately chose SVM as the classifier for Outguard because it achieves 97.9% TPR and 1.1% FPR.

Effective threat detection systems inevitably prompt malicious actors to adopt detection evasion techniques. We examined the resilience of Outguard to multiple forms of adversarial evasion by removing each feature category and measuring the overall detection accuracy. Our analysis showed that while an adversary can potentially evade some of the features used in the learning process, such evasion techniques severely cripple cryptojacking operations and curtail an adversary's profits. For instance, removing mining-based and network-based features (e.g., WebAssembly) is feasible, but their removal makes cryptojacking less attractive to adversaries.

We deployed Outguard in the wild on the Alexa Top 1M domains and discovered 6,302 cryptojacking sites, of which 3,600 had not been detected by existing techniques. Since in-browser cryptomining, in theory, is not necessarily malicious if user consent is obtained, we performed additional analysis to verify that, in reality, nearly all cryptomining sites did not ask for user consent.Through the lenses of both the Alexa Top 1M domains and passive DNS data, cryptojacking appears predominantly in the long tail of lower popularity sites that promote illicit content such as torrents or copyrighted video streaming. Our analysis also exposed 69 incidents of cryptojacking loaded through advertisement web pages. These incidents constituted unintentional cryptojacking, which occurred without the direct involvement of website owners/publishers.

We examined the delivery of cryptojacking and coordination of cryptojacking monetization. We classified cryptojacking incidents into 14 major families based on their source code origin and found a competitive blend of five families that, in total, accounted for more than 70% of cryptojacking occurrences. We also detected 486 cryptojacking incidents using 24 less well-known cryptomining services. These services provide pool mining services that charge as little as 1% of total mining revenue. Finally, we conducted a campaign analysis and identified 35 cryptojacking campaigns which ranged in size from 2 to 121 websites. These websites deliver different types of cryptojacking libraries such as CoinHive, JSeCoin, and Crypto-loot.

Given the extent of the identified cryptojacking websites, we envision multiple deployment scenarios for disrupting cryptojacking operations in the wild. The output of Outguard could be used to augment blacklists or anti-malware mechanisms in popular browsers. The proposed detection model could also be used as a browser extension that preemptively monitors the content of the visited websites and notifies the user if the website loads a cryptojacking library.



```
<html>
<script src="http://delivery.com/cryptojacking.js">
</script>
</html>
```

Figure 1: Cryptojacking overview. Websites (1) include cryptojacking code from a delivery server (2). Victim browsers execute mining tasks assigned from a mining pool/proxy (3), which ultimately adds new blocks to the blockchain (4).

2 BACKGROUND AND RELATED WORK

2.1 Cryptojacking Workflow

Modern cryptocurrencies are based on blockchain consensus, which was first presented and popularized by Bitcoin [41]. The blockchain is maintained and extended by miners, which typically compete to compute a proof-of-work puzzle that must be satisfied by new blocks. This proof-of-work serves as both a denial-of-service deterrent and an economic indication of commitment to the consensus protocol. Initial proof-of-work protocols, such as Bitcoin's Hashcash, benefit substantially from custom hardware optimizations, and most mining now occurs on custom mining rigs.

The development of Cryptonight [6], a new proof-of-work algorithm that significantly reduces the advantages of custom hardware, positioned GPUs, custom FPGAs/ASICs, and CPUs back into the same ballpark. In fact, Hashcash relies on brute-forcing SHA-256, which is easily parallelized and reliant on computational bandwidth, making it amenable to customized ASIC hardware. Cryptonight, on the other hand, is a memory-bound protocol (2Mb per core) that relies on random access to slow memory and each new block relies on the results of the prior block. One likely side effect of this development is cryptojacking, which allows an adversary to exploit a victim's computing power to mine cryptocurrencies without consent. Figure 1 outlines the cryptojacking workflow, which involves several entities, starting with user-facing cryptojacking websites. These websites are loaded through a user's JavaScript-enabled web browser and embed a cryptojacking JavaScript library, either through intentional design or hacker injection. The cryptojacking library is delivered from a delivery server, which can reside on the same domain as the base website or, more commonly, is loaded through a separate delivery server host.

Once the cryptojacking code has begun execution, it connects to a mining pool proxy in order to receive mining tasks. A proxy server is often employed between the mining pool and cryptojacking miners to limit the number of connections and further subdivide mining tasks assigned by the mining pool. Prior to mining, each mining operator (who may control more than one miner) will have registered a mining account with the mining pool, so that she can receive payments for her cryptojacking efforts. Some form of account identifier is usually transmitted from the cryptojacking client to the mining pool to correctly map mining output to the correct mining account. These communications typically occur via Stratum [2], a JSON RPC protocol for mining pool coordination that is cryptocurrency-agnostic. The mining pool monitors the blockchain and assigns mining tasks to each miner that connects to it. These tasks are a low threshold version of the actual proof-of-work puzzle, so that the mining pool can record and reward incremental mining effort. Occasionally, when one of the completed tasks constitutes a successful block, the mining pool adds the block to the blockchain and claims the associated mining reward for itself. Then, periodically, the mining pool will distribute a portion of this reward among its mining accounts in proportion to the work accomplished; the rest is kept as a mining pool fee. These fees are typically around 10–30%, but can be as low as 1% or 2% [50].

The description provided thus far has focused on the components involved in the cryptojacking workflow. In reality, many of these components are controlled by a shared operator. To elucidate the common mappings between components and operators, we present two scenarios: full-service cryptojacking and do-it-yourself (DIY) cryptojacking. For full-service cryptojacking such as CoinHive, a website owner or hijacker only needs to create a CoinHive account and then insert a script inclusion tag her website. In this scenario, CoinHive operates the delivery server and the mining pool / proxy. For a DIY use case, the publisher may host her own delivery server and have the cryptojacking code connect to a public mining pool.

2.2 Related Work

Techniques seeking to gain financial profit through unauthorized access to a victim computer are numerous. There is a large body of work studying adversarial attempts to profit from end-user machines. Ransomware [25, 27, 29, 44], malicious advertisements [7, 34, 56], click fraud [19], web-based social engineering attacks [5, 8, 9, 26, 33, 51, 53], online scams [28, 39], and Potentially Unwanted Programs (PUPs) [31, 42, 48] are only a few examples of these activities in the wild. In the following paragraphs, we explain more closely related work on cryptojacking operations.

A subset of prior work on cryptojacking [16, 40, 46] mainly focuses on the impacts of unwanted mining on user machines (e.g., CPU utilization), and provides an estimate of network size and revenue for CoinHive. In this paper, we mainly focus on less noisy features that can facilitate automatic detection. Very recently, Konoth et al. [30] and Hong et al. [20] concurrently and independently performed systematic analyses on cryptojacking incidents. Hong et al. [20] proposed two techniques by statically searching for a set of fixed signatures in cryptojacking code as well as identifying patterns in the execution stack. The authors demonstrated the insufficiency of blacklists and provide insight on a set of techniques (e.g., obfuscation) that are currently used by adversaries to bypass blacklists. While there are some similarities between their detection approach and ours (e.g., identifying specific patterns), Outguard mainly relies on dynamic behaviors of cryptojacking libraries that are the core components of in-browser cryptojacking, and that are quite unlikely to be used in the same way in benign programs. We elaborate on the feature set in Section 3, and show that the system still achieves high detection coverage if adversaries utilize different levels of obfuscation techniques.

Code Family	Currency	MinerBlock	Wappalyzer (MS%)	Labeled Dataset (%)
JSEcoin	JSECoin	✓	✓ (30.6%)	1,358 (45.2%)
CoinHive	Monero	✓	✓ (48.2%)	1,124 (37.4%)
CoinHive Captcha	Monero	✓	✓	132 (4.4%)
DeepMiner	Monero	✓	✓	88 (2.9%)
Coinimp	Monero	✓	✓	68 (2.3%)
Crypto-Loot	Monero	✓	✓	55 (1.8%)
ProjectPoi	Monero	-	✓ (19.2%)	43 (1.4%)
Coinlab	Monero	✓	✓	43 (1.4%)
Cloudcoins	Monero	✓	✓	42 (1.4%)
Monerominer	Monero	✓	✓	28 (0.9%)
Coinhave	Monero	-	✓	15 (0.5%)
Monero.cc	Monero	✓	✓	10 (0.3%)
Webmine	Monero	-	✓	-
Inwemo	Monero	-	✓	-
Total	-	10 families	14 families (100%)	3,006 (100%)

Table 1: The training set of cryptojacking libraries we collected. Wappalyzer recognized all the libraries collected, but lacks market share (MS%) data for most of the families at the time of writing. Our labeled dataset of 3,006 websites covers 12 of 14 (85%) cryptojacking libraries.

The prior work most relevant to ours is MineSweeper [30], which introduces a static analysis technique to study the wasm code used by cryptojacking code. While this approach is an improvement over the status quo, it would be desirable to complement it with a more generic approach that requires minimal human intervention. Outguard does not require manual analysis, and does not rely on cryptographic primitive identification. This was a conscious design choice to avoid possible false positive cases, as acknowledged by the authors [30]. MineSweeper also shows that CPU cache L1 event monitoring can be quite useful in detecting cryptojacking websites by analyzing two cryptojacking families with 100% CPU usage. While WebAssembly provides significant freedom to manage the memory layout, an adversary may not be able to selectively load mining operations in the L1 or L3 cache in order to bypass the proposed detection. However, our analysis reveals that cryptojacking libraries impose different L1 cache event loads as a function of the CPU threshold, leaving threshold-based approaches still vulnerable to future evasion. Furthermore, measuring CPU cache events is a very expensive operation, as the entire process of inferring the state of the CPU cache is very sensitive to noise and to the global status of other active applications on the test machine. This makes approaches similar to MineSweeper less likely to scale. Outguard does not have this limitation, and the detection accuracy of Outguard is not impacted by the type of cryptographic function or CPU-related features (see Section 3). Thus, Outguard is a more generic solution to the same problem space.

3 OUTGUARD

Because cryptojacking is still in its infancy, existing detection mechanisms are still sparse and relatively naive: NoCoin [45], minerBlock [4], and CoinBlockerLists [1] rely on open-source, user-updated blacklists to match domains or filenames. Building of an automated cryptojacking detection system in this nascent landscape requires a full construction process: building a ground-truth dataset, creating features, selecting features, training a classification model, and finally evaluating the model under benign and adversarial inputs. In this section, we describe Outguard's architecture, shown in Figure 2, and detail its implementation and performance.

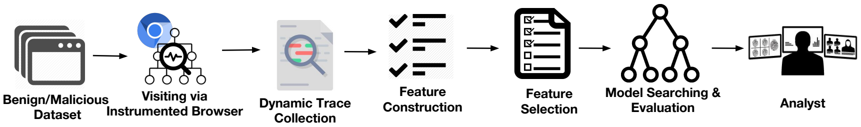


Figure 2: A high-level view of Outguard's architecture.

3.1 Building Ground Truth

Currently, no reliable labeled datasets for cryptojacking websites exist. To begin building our training dataset, we first aggregated the frequently updated public blacklist patterns in CoinBlockerLists [1] as well as the blacklist patterns used in popular browser extensions NoCoin [45] and minerBlock [4]. We scanned the Alexa Top 1M websites using our instrumented Chromium browser and collected 3,006 websites that contained JavaScript libraries or delivery servers matching the blacklist patterns.

To further label the cryptojacking libraries, we used Wappalyzer [52] version 5.2, a web technology detection tool that lists the libraries used by a given website. Wappalyzer labels cryptojacking JavaScript (JS) code by examining its origin. To create our labeled dataset, we incorporated the same methodology by mapping URLs to cryptojacking family names. For example, if the code originated from `http//cdn.cloudcoins.co`, we labeled the code as an instance of the Cloudcoin family. As shown in Table 1, we found 12 cryptojacking families in our labeled data, which cover 100% of the cryptojacking libraries known to MinerBlock. We did not find any instances of two out of the 14 total cryptojacking libraries known to Wappalyzer: *Inwemo* and *Webmine*. Inwemo is a seemingly defunct advertising replacement product [22], and Webmine is a similar product based in the Czech Republic.

To build a ground-truth set of non-cryptojacking websites, we randomly selected websites from the Alexa Top 100K. Even though there have been a few reported cases of high popularity sites exhibiting cryptojacking [13, 35], these websites are generally well-maintained and less likely to be subject to new security incidents (as later indicated in Section 5). The non-cryptojacking dataset covers a variety of real-world websites, including dynamic and high-profile websites such as news, legitimate video streaming, and popular companies' websites as well as static websites. This also contains a wide range of JS usage, including high-usage sites (e.g., cnn.com). We combined the ground truth cryptojacking and non-cryptojacking websites into two training datasets: a balanced dataset and an imbalanced dataset. The balanced dataset (BD) contains equal numbers of 2,000 cryptojacking and 2,000 benign websites. The cryptojacking websites were selected randomly and include members of all the 12 labeled cryptojacking families. The motivation behind the imbalanced dataset (ID) is that, in reality, there are significantly more benign websites than cryptojacking websites, and a balanced distribution may bias the accuracy of the classifier. To analyze the performance of the classifier, we built the ID with an imbalance ratio of 1 to 10, such that it contains 2,700 cryptojacking websites and 27,000 benign pages.

3.2 Feature Selection

After carefully examining a variety of cryptojacking websites in our labeled data, we developed a set of features that are broadly characteristic of cryptojacking libraries. Starting with the raw trace data from our data collection, we considered almost all the related features in rendering and JavaScript engine that could be useful in attributing the runtime behavior of JS code. However, we narrowed down our list to 12 features, as we observed those 12 features can potentially reflect cryptojacking behaviors more accurately. We investigated the following features, which we suspected would be strong indicators of cryptojacking, but that did not prove to be distinctive cryptojacking features.

- *JS engine execution time*: We expected JS execution time to dominate a cryptojacking website, but found that this feature is very noisy; many websites continuously execute JS.
- *JS compilation time*: Cryptojacking did not display significant differences in JS compilation time.
- *Garbage collection*: Garbage collection is often a loose proxy for memory usage, but we didn't find differentiating behavior for cryptojacking libraries.
- *Iframe resource loads*: Cryptojacking code is often loaded in an iframe, but rampant advertising iframes drowned out meaningful data.
- *CPU usage*: Cryptomining relies directly on CPU resources, but many cryptojacking libraries throttle CPU usage and are hard to distinguish from JavaScript-heavy sites.

Ultimately, by applying our domain knowledge and evaluating different potential features, we converged on seven features that represent the JS runtime execution, networking, JS event loads, and cryptomining properties of cryptojacking libraries. These features, with the potential exception of WebSockets, are essential for the profitable operation of cryptojacking campaigns. We show that all seven features are nearly ubiquitous across all 12 families of cryptojacking libraries in our labeled training set. We expand on each feature below.

Web Workers Web workers are a modern JS feature first specified by W3C in 2009 [23] for executing JS files in background threads, independent of a site's main execution thread. Compared to non-cryptojacking usage of web workers for processor-intensive tasks (e.g., image processing during photo upload), we found that cryptojacking libraries are more likely to utilize *multiple* web worker threads for running several mining tasks concurrently. We observed that almost all 12 families utilize at least 3 web workers to perform mining. Outguard's instrument browser dynamically extracts the number of threads as a feature for cryptojacking detection.

Parallel Tasks We observed that benign websites typically create threads for specific purposes (e.g., caching or streaming video).

Our analysis shows that in all the cryptojacking websites, in addition to running multiple web worker threads, cryptojacking libraries tend to run the *same* tasks in each thread, repeatedly invoking the same sequence of method calls. Outguard outputs the number of identical tasks as a numeric value for detection.

WebAssembly WebAssembly, or wasm [54], is a new web standard introduced in 2015 that allows web developers to specify assembly-like code that executes on modern browsers at near-native speeds. The performance of WebAssembly makes it an attractive target for processing intensive cryptomining, and we found that all cryptojacking libraries in our training set use WebAssembly. Outguard detects the usage of WebAssembly by monitoring the exchange of wasm modules between the main execution thread and web workers. This detection technique, which is oblivious to the communication obfuscation found in some variants of CoinHive, results in a boolean feature that indicates the presence or absence of WebAssembly.

WebSockets WebSockets are a low-overhead, full-duplex communication channel over TCP that was standardized in 2011. Our analysis shows that most mining modules in a browser establish a WebSocket connection to a remote mining pool/proxy. In addition to overcoming some limitations of JavaScript APIs in accessing the TCP socket, WebSocket provides a faster communication channel than standard TCP, as it eliminates the sending of additional HTTP headers for each client-side request. We also observed that cryptojacking libraries use WebSocket APIs to receive data from the mining pool which reduces bandwidth usage. Outguard searches for `webSocketCreate` events in its JS execution traces and outputs a numeric feature value for the number of WebSocket connections.

Hash Algorithms As shown in Table 1, Monero is the most common cryptocurrency mined by cryptojacking libraries. Monero mining is a proof-of-work (PoW) task that requires the repeated calculation of the CryptoNight hashing algorithm, which can be calculated efficiently on consumer CPUs. Outguard identifies CryptoNight hashing by searching for specific function names and binary signatures in JS execution call stacks. For instance, *JSECoin* and some *CoinHive* variants call functions named `module._cryptonight_hash` and `CryptonightWASMWrapper.hash`, respectively. On the other hand, CoinHave embeds and executes binary wasm hashing functions inline for which we also developed detection signatures. This detection mechanism outputs a boolean feature.

PostMessage Event Load Web workers constantly have to send the results of mining operations to the main process by sending the job id, the nonce, and the result of each operation. This information is then sent to mining proxies over WebSocket connections in order to manage mining tasks. The communication between the renderer process and the web workers is handled via PostMessage. As the number of web workers is significantly high in cryptojacking websites, and PostMessages are the only form of communication with web workers in most modern browsers, a large number of PostMessage events in site visits can be an indication of cryptojacking. We report the total number of PostMessage events as a numeric feature.

MessageLoop Event Load Chromium performs task management among threads by using MessageLoop. When a WebAssmebly module is loaded into a web worker thread, it starts cryptographic operations inside loops. These operations are pushed to the MessageLoop of the thread to be sequentially processed. While use of MessageLoop does not by itself necessarily mean that a website is performing cryptojacking operations, a significantly large number of MessageLoop events during a short site visit, along with other features, can be an indication of cryptojacking. We report the number of MessageLoop events as a numeric value.

Metric	SVM		Random Forest	
	BD	ID	BD	ID
TPR	96.2%	97.9%	95.6%	97.1%
FPR	0.2%	1.1%	1.3%	2.8%
AUC	97.8%	98.4%	96.9%	98.1%

Table 2: 10-fold cross-validation on Support Vector Machines (SVM) and Random Forest (RF), using balanced (BD) and imbalanced (ID) datasets. We selected SVM as the classifier for Outguard because of its superior 96.2–97.9% TPR and 0.2–1.1% FPR.

3.3 Prototype Implementation

Automatic recording of the dynamic behavior of client-side JavaScript code, such as resource usage, code provenance, and network communication, is a non-trivial challenge. In order to extract these features from websites, we instrumented Chromium and leveraged the Chromium Debugging Protocol [17] to access nearly all the functionality of DevTools [18]. More specifically, the crawler collected browser request and response traces, JavaScript execution traces, and the source code of both inline and dynamically loaded JavaScript code. This set of information was sufficient to attribute specific JavaScript code to the requested computational resources, the functions being executed, any external code referenced by the function, and the execution time spent per function. This trace data includes all features we discussed with respect to the feature selection phase. We deployed the Outguard crawler on 40 virtual machines. For each website, the crawler stayed on the visited website for 45 seconds and interacted with the page by programmatically scrolling to the bottom of the page to simulate user behavior. We empirically found that 45 seconds was sufficient time for a cryptojacking website to load a cryptojacking library, communicate with a mining pool, and run mining tasks. Our analysis on 1,000 randomly selected websites shows that the run-time overhead of the instrumentation layer is approximately 2%.

The second module of Outguard is a classifier that analyzes the collected data and detects cryptojacking websites by incorporating the features described in Section 3.4. To construct the detection model, we tested multiple classification algorithms and found that an SVM classifier produced the best detection results on the training dataset. To construct the classification model, we used the SVM implementation provided by scikit-learn [47]. We evaluate the classifier in Section 3.4.

3.4 Evaluation

In this section, we examine Outguard's detection accuracy on the ground-truth training dataset and compare it with CPU utilization, an existing cryptojacking detection heuristic. To identify the best machine learning algorithm for automatic classification of cryptojacking libraries, we ran a 10-fold cross-validation on our BD and

Rank	Feature	Category	Score Ratio	New?
1	No. Web Workers	E	100%	✓
2	Creating Identical Tasks	E	93.7%	✓
3	PostMessage Event Load	L	91.2%	✓
4	Using WebAssembly	E	85.8%	[20, 30]
5	MessageLoop Event Load	L	82.3%	✓
6	Using Hash Functions	M	76.4%	[20]
7	No. WebSocket Connections	N	48.2%	[30]

Table 3: Feature importance ranking in OUTGUARD. Features are grouped into categories based on runtime execution (E), mining (M), Event Load (L), and network (N) behavior.

ID training datasets by using the scikit-learn [47] implementations of Support Vector Machines (SVM) and Random Forests (RF), based on prior work that showed these two classification algorithms work well on balanced and imbalanced datasets [28]. As shown in Table 2, both models perform above 95% TPR and below 3% FPR. We selected SVM as the default classifier for OUTGUARD because it slightly outperforms the RF model across both training sets.

The SVM model chosen for OUTGUARD is highly selective, achieving TPR rates of 96.2% and 97.9% for our BD and ID datasets, respectively. Our analysis of the detected cases revealed that OUTGUARD achieved a false positive rate (FP) of 1.1% (with 32 false detections) on an imbalanced dataset. Further manual analysis revealed that all the false positive cases were a set of parked domains that inserted 29 JavaScript scripts, on average, including trackers and advertisements. These websites were falsely labeled as cryptojacking because multiple trackers were using WebSocket connections for communications, and some of these scripts created multiple threads for client-side data processing. Our manual analysis of the false negative cases revealed that OUTGUARD was unable to detect some variants of CoinImp because they did not contain indicative function calls in their execution traces, and they used fewer than three web workers.

3.4.1 Feature Analysis. We divided the features we selected in Section 3.2 into four categories in order to reason about subsequent analysis, such as feature ranking and classification evasion, from a feature function perspective. The categories are (1) runtime execution-based features such as WebAssembly usage; web worker quantity, and running of parallel tasks for mining activities; (2) a network-based feature which describes the type of communication between the mining module and the mining pool; (3) event load-based features such as MessageLoop and PostMessage Load, and (4) a mining-based feature that looks for hash algorithm fingerprints.

To determine the significance of each incorporated feature, we performed *recursive feature elimination* (RFE) using OUTGUARD's SVM model on the BD dataset. The ranking procedure begins with all seven features. At each subsequent step, a feature with the minimum weight is removed, and the FP and TP rates are re-calculated to measure the contribution of the removed feature. Table 3 displays the ranks of all features, in descending order of importance. For convenience, we derived a score ratio by dividing each feature score with the maximum score. We found that the five most predominant detection features target the dynamic characteristics of cryptojacking, making the system more resilient to common evasions such as code obfuscation. This matches our intuition that such behaviors contribute to successful mining operation – avoiding these techniques would likely impact revenue generation.

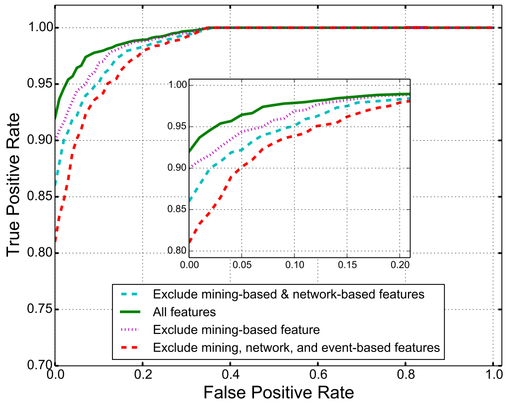


Figure 3: Detection results of OUTGUARD on the labeled dataset when different evasive scenarios were used.

3.4.2 Evaluating the Effectiveness of CPU Utilization. We compared OUTGUARD's classification model against a CPU usage model which is a detection heuristic utilized by several reports [14, 38]. We collected the traces of all websites in our BD training dataset that consumed at least 50% of the total CPU capacity. One might expect a 50% CPU usage threshold to be reasonable for detecting cryptojacking given that we observed cryptojacking libraries from all 12 families establish CPU thresholds of at least 50% of the total CPU capacity. The CPU usage heuristic was applied to the BD. Out of the 2,000 non-cryptojacking websites in BD, CPU usage heuristic misclassified 263 as cryptojacking websites (false positive rate = 13.1%). Manual inspection of a handful of these cases revealed non-cryptojacking websites that introduced high CPU overhead due to a large number of requests and responses, constant JavaScript compilation, and frequent dynamic changes in the DOM during script loads. For example, a visit to cnn.com, a high-profile news website, generated an execution trace with 151 unique domain names that were used to retrieve JavaScript resources and other content (e.g., fonts, images, and styles).

Our comparative analysis demonstrates that it is not uncommon for non-cryptojacking websites to exhibit high CPU utilization, even for websites that typically do not require elevated CPU usage in order to function. CPU utilization can be a helpful feature for analysis of cryptojacking at small scales, but it is a less effective feature for automatic detection at scale. CPU usage is a noisy signal—relying on this heuristic increases the likelihood of false positive cases when hundreds of thousands of websites are being analyzed.

3.5 Adversarial Considerations

Cryptojacking operators, like malware authors, are likely to exercise techniques to evade OUTGUARD by changing their implementation strategies. To address potential evasion scenarios, we evaluated the efficacy of OUTGUARD's SVM model with specific feature categories selectively removed. Figure 3 summaries the results of the experiment. The solid green curve shows the ROC curve of OUTGUARD when it incorporates all the features into the detection model. The dotted purple curve represents the ROC curve in the absence of mining-based features. Evasion of this feature from the detection model is a realistic possibility as our approach to detect hashing algorithm usage is somewhat brittle because it is based on collecting in-line binaries that are statically included in cryptojacking libraries or on finding references to a specific hash function in execution traces. Cryptojacking operators can re-generate cryptojacking binaries by adding dead code or by renaming functions, much as evasive malware does, to bypass mining-based feature. In this scenario, the detection performance of Outguard degrades, but it still achieves a relatively high detection accuracy, with 93% TPR and 1.2% FPR. The mining-based feature is important, but as suggested by the RFE analysis, it is not precariously important, since Outguard still achieves high detection results without it.

In the next step, we excluded the network-based feature with the assumption that a cryptojacking library may forgo a websocket connection to the mining pool/proxy server and connect through other means instead. For instance, the code may use standard TCP connections or JSONP or other communication techniques. In this case, Outguard will rely on event loads and runtime execution features to identify cryptojacking activity. The dashed blue ROC curve illustrates Outguard's performance with mining-based and network-based features excluded. The system still achieves 92% TPR and 1.5% FPR. In the final experiment, we removed the event load features under the assumption that adversaries may try to limit the number of threads, and thereby impact the generated events between the web workers and the host process. The red ROC curve illustrates Outguard's performance. In this case, Outguard relies solely on runtime execution features to identify cryptojacking activity. Use of runtime features (e.g., WebAssembly) in cryptojacking libraries is vital from profitability perspective, as these libraries have to incorporate such browser functionalities to use system resources effectively. Failing to do so would make the in-browser cryptomining not practical or less attractive for generating revenue. So in this final scenario, the classification accuracy has decreased significantly with the loss of networking features, down to 87% with 1.1% FPR.

Note that while an adversary can evade network or event load features, doing so will likely have a significant effect on the efficacy of mining operations, as the mining module requires low-overhead connections in order to communicate with mining pools. Furthermore, as discussed in Section 2, in-browser cryptojacking relies on multiple external parties to operate. Consequently, network-level changes requires compatibility with the rest of the parties involved in the ecosystem (e.g., pools and proxies). Moreover, minimizing of JS event loads to evade the corresponding feature set is not a trivial task, as the entire operation, including task management and task execution, significantly depends on the communication between the host process and the mining threads. Overall, we argue that a tool similar to Outguard can provide robust detection of cryptojacking websites and potentially increase the operational costs of cryptojacking or reduce the generated revenue of such operations.

4 REAL-WORLD DEPLOYMENT

We demonstrated the feasibility of real-world deployment of Outguard by performing an analysis of the Alexa Top 1M websites. Outguard detected 5,873 instances of cryptojacking. Using this data, we characterized the broader cryptojacking ecosystem by examining the websites on which cryptojacking occurs, and the infrastructure supporting its mining and monetization. We later performed a scan of the Alexa Top 600K websites to supplement the case study.

Results	Experiment 1	Experiment 2	Total
Target crawl size	Alexa 1 Million	Alexa 600K	-
URLs visited	3,798,433	1,329,684	5,128,117
Unique cryptojacking domains	5,873	429	6,302
New detections	3,171	429	3,600 (57%)
Overlap with MinerBlock	-	-	3,776 (60%)
Overlap with AntiMiner	-	-	3,684 (58.5%)

Table 4: Number of cryptojacking incidents detected by Outguard. 3,600 (57%) of the detected cryptojacking websites were new detections.

For the real-world application of Outguard, we enabled Google Safe Browsing in our instrumented browser to avoid falsely reporting cryptojacking that might be prevented by default browser protection mechanisms. That allowed us to represent realistically the cryptojacking exposure of an end-user browsing the web. We also modified Outguard to access 3 random links on each Alexa Top webpage to simulate user interaction/navigation on the page. The measurements were conducted over two time ranges: the first scan took place from February 12 to April 15, 2018, on the Alexa Top 1M websites, and the second scan took place from October 20 to October 30, 2018, on the Alexa Top 600K websites. In total, Outguard reported 6,302 cryptojacking websites across 5,128,117 URLs, as shown in Table 4.

4.1 Obtaining User Consent

Mining cryptocurrency in a browser is not inherently malicious, and in-browser cryptomining has been presented as a justifiable business model alternative to advertising and captcha services. The lack of user consent is the distinguishing factor between legitimate in-browser cryptomining from cryptojacking.

We performed a manual analysis of 100 randomly selected websites that contained the JSECoin library by visiting the websites and checking whether the opt-in system was enabled on them. 22% of the websites were not reachable during the manual check (e.g., `mxqproject.com, masir.us`). We found that in the remaining 78% of websites, the JSECoin cryptojacking library was automatically executed without user consent (e.g., `media9.pk, seriesenlinea.net`). Furthermore, on late 2017, CoinHive introduced a version of cryptomining code called AuthedMine, which requires user permission to turn the browser into a Monero miner. We did not find any instance of this library either in the training or in the large-scale deployment of Outguard, which mirrors other reports [24]. This experiment suggests that website operators are likely to be reluctant to use libraries with explicit opt-in forms, and that most detected cases do, in fact, constitute cryptojacking.

4.2 Detection Coverage Evaluation

In total, Outguard detected 6,302 cryptojacking websites. 3,600 of the detected sites in our two experiments were newly detected websites not observed in the training dataset. Since we do not have ground truth labeled data in the large-scale experiment, we cannot provide precision-recall analysis of the detection results. However, manual analysis shows that OUTGUARD falsely reported 14 websites as cryptojacking because they were loading several frames and iframes to embed videos and pop-ups. In addition to creating multiple threads, we observed several WebSocket connections with remote servers to load streaming content in these websites. We consider these cases false positives. The manual analysis of the trace files indicated that the remaining newly detected websites were true positives. We cross-checked the output of OUTGUARD against dedicated security solutions such as MinerBlock [12] and Antiminer [49]. Table 4 summarizes the results of the experiment. Our examination of the filtering lists in these extensions showed that these extensions are not specifically designed to identify cryptojacking websites. Rather, they primarily look for addresses of known mining pool proxies (e.g., `xmrpool.net`). As shown in Table 4, OUTGUARD had approximately 40% more coverage than MinerBlock and 41.5% more coverage than AntiMiner.

While disrupting the communication between the cryptojacking libraries and the mining pools is an effective mitigation technique, this approach is less likely to be very successful in identifying different variants of cryptojacking families. For example, the naive versions of cryptojacking families make queries to a specific address that can easily be detected and blocked. However, we observed several cases of Crypto-loot, JSECoin, and CoinImp, which use legitimate cloud deployment services (e.g., `now.sh`) for this purpose. Unsurprisingly, it is impractical to block a domain name or IP address that belongs to cloud services without affecting a large number of other legitimate applications. Consequently, these extensions are limited to domains that do not map to cloud services, and can therefore be evaded fairly easily. We posit that a solution such as OUTGUARD that monitors the dynamic behavior of websites and JavaScript libraries is more likely to prevent cryptojacking incidents in practice. With that being said, these dedicated security solutions can benefit from using OUTGUARD by automatically updating their corresponding filter lists.

FQDN	Lookups (%)	e2LD+eTLD	Lookups (%)
coinhive.com*†	128.31M (21.5%)	coinhive.com*†	308.09M (51.7%)
cnhv.co*†	34.83M (5.8%)	cnhv.co*†	34.83M (5.8%)
moonbit.co.in†	15.53M (2.6%)	adsptp.com†	19.10M (3.2%)
www.adsptp.com†	13.60M (2.3%)	watchfree.to†	17.30M (2.9%)
1480876790.rsc.cdn77.org†	12.87M (2.2%)	moonbit.co.in†	15.61M (2.6%)
www.rcyclmnr.com†	12.24M (2.1%)	rcyclmnr.com†	13.41M (2.3%)
www.alluc.ee†	9.35M (1.6%)	cdn77.org†	12.87M (2.2%)
www.watchfree.to†	9.12M (1.5%)	alluc.ee†	12.10M (2.0%)
ws008.coinhive.com*	8.69M (1.5%)	crypto-loot.com*†	10.18M (1.7%)
ws013.coinhive.com*	8.68M (1.5%)	recycloped.com†	6.77M (1.3%)
ws014.coinhive.com*	8.68M (1.5%)	pornxs.com†	6.51M (1.1%)
ws012.coinhive.com*	8.67M (1.5%)	faucethub.io†	3.86M (0.6%)
ws011.coinhive.com*	8.67M (1.5%)	baiduccdn1.com†	3.85M (0.6%)
ws009.coinhive.com*	8.67M (1.5%)	graaam.com†	3.83M (0.6%)
ws010.coinhive.com*	8.67M (1.5%)	mycashbar.com†	3.44M (0.6%)
ws030.coinhive.com*	8.39M (1.4%)	otohits.net†	2.67M (0.4%)
www.recycloped.com*	6.77M (1.1%)	jsecoin.com*†	2.53M (0.4%)
ws020.coinhive.com*	6.73M (1.1%)	kickass.cd†	2.23M (0.4%)
ws016.coinhive.com*	6.69M (1.1%)	oyunuoyna.com†	2.18M (0.4%)
ws018.coinhive.com*	6.69M (1.1%)	xxgasm.com†	2.14M (0.4%)
175,202 others	263.83M (44.3%)	*5,807 others*	112.19M (18.8%)
Total	595.67M (100%)	*Total*	595.67M (100%)

Table 5: Grouping the top passive DNS lookups by FQDN and e2LD+eTLD highlights that the expanded passive DNS dataset includes two types of domains: cryptojacking administration domains (*) and cryptojacking website domains(†).

5 CHARACTERIZING CRYPTOJACKING WEBSITES

Armed with the results of OUTGUARD's deployment in the wild, we first seek to answer the question of where cryptojacking occurs. Reports of cryptojacking websites have painted a myriad landscape of sites infected via web vulnerabilities [36], popular websites with cryptojacking advertisements [13], and suspicious e-commerce websites [15]. We will now take a comprehensive look at the 6,302 detected cryptojacking websites, investigating their popularity, categories, and lifetimes.

5.1 Website Popularity and Type

We look up each cryptojacking website's Alexa ranking as a metric for the relative number of visits for each cryptojacking website. 220 websites (3.5%) fall within the top 100K, 602 (9.6%) appear between the top 100K and top 1M, and the remaining 5,480 (87%) are outside of the top 1M. This suggests that cryptojacking occurs primarily in the long tail of lower-popularity websites, as hypothesized in Section 3.1 during the discussion of training data construction.

In order to better address the 5,480 sites outside of the Alexa Top 1M, we also explored passive DNS data as a means to estimate the relative popularity of the cryptojacking domains. Through the lens of one of the largest U.S. ISPs, we collected the contents and daily lookup volume of DNS resource records (RRs) for all successful DNS resolutions. We extracted all passive DNS RRs that match the seed list of 5,873 cryptojacking domains from the initial Alexa Top 1M real-world deployment and also included RRs with subdomains of the seed list. This dataset consisted of 595.6 million RRs collected from April 2016 to March 2018.

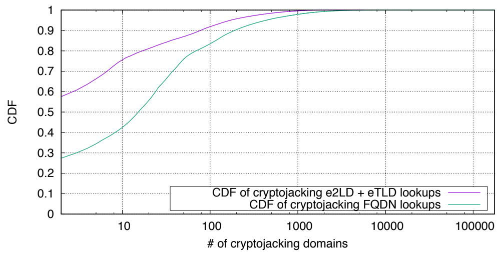


Figure 4: Distribution of passive DNS lookups for cryptojacking FQDNs and e2LD+eTLD. By both metrics, `coinhive.com` accounts for the most lookups, 21.5% and 51.7%, respectively.

We constructed the initial seed domain list for passive DNS data collection from websites that have embeded cryptojacking; however, some domains and their subdomains (e.g. `coinhive.com`) are also used for cryptojacking delivery servers or mining pool/proxy servers that coordinate cryptojacking miners. This muddles the passive DNS dataset with multiple use cases, but grouping passive DNS by effective second-level domain (e2LD) and effective TLD (eTLD), as shown in Table 5, makes it easier to distinct between cryptojacking websites and cryptojacking delivery servers or mining pools.

To characterize the ecosystem of websites that have embeded cryptojacking, we categorized the top 100 e2LD+eTLDs, which accounted for 92% of all passive DNS lookups. Granted, domains dually purposed to be both cryptojacking websites and cryptojacking administration sites have inflated lookup counts, so we examined

Category	#	Category	#
torrent	21	software	2
video	15	forum/chatroom	2
unknown	11	education	2
malicious	9	crypto faucet	2
porn	7	pyrotechnics	1
news	7	mining pool	1
cryptojacking	7	manga	1
gaming	4	image upload	1
music	3	automobile	1
traffic exchange	2	advertising	1

Table 6: Categories for the top 100 most popular cryptojacking e2LD+eTLDs.

only the number of domains for each category and did not evaluate the lookup-weighted category composition. In Table 6 we found that the top cryptojacking domains were typically sites with illicited content such as torrent hosting sites, video download/streaming sites, and malicious websites. Assuming that most of these cryptojacking incidents were not the result of website compromise, these results suggest that cryptojacking is used by criminal actors more successfully than lawful actors.

One particularly interesting mechanism for cryptojacking delivery is inclusion through an advertisement on a website through an embedded iframe. Through this form of indirect delivery, cryptojacking can achieve broad distribution, as demonstrated by DoubleClick's reported inclusion of cryptojacking on Youtube [13]. While we did not detect cryptojacking directly included in advertisements, we did find 69 cryptojacking incidents in landing pages that resulted from clicking on an advertisement. We measured these occurrences by cross-referencing the provenance of websites derived from clicked links with a list of ad domains provided by EasyList (`easylist.to`) and Ghostery (`www.ghostery.com`). We observed several instrances of *crypto–loot* and *JSECoin* libraries in our dataset. These instances were mainly delivered via specific ad-networks such as `adnxs.com` (33%),`spotxchange.com` (23.1%), and `stickyads.com` (14.5%). While the number of advertisement-based cryptojacking incidents is relatively small, they represent a fundamentally different type of cryptojacking: *unintentional* cryptojacking incidents that are dynamically loaded during page loads.

5.2 Website Lifetime

We continuously monitored all cryptojacking websites from our first deployment, which observed 5,873 websites by crawling them every ten days for 3 months to approximate their active lifetime. We labeled a cryptojacking website as inactive if we did not observe any sign of cryptojacking operations in a given trace file. We performed 9 crawls over 3 months from May 20 to July 25, 2018 and found that 32% of the cryptojacking websites became inactive in less than 10 days. These websites were mainly hosted on a set of cloud servers and were likely managed by the same group of operators. While we observed a steady decrease in the number of active cryptojacking operations over time, there were 619 (10.5%) websites—mostly video streaming, gaming, and application download sites—that stayed active for the entire measurement period. An additional 1,552 (26.4%) websites hosted cryptojacking libraries for a moderate duration of 3–9 weeks. In total, 68 (3.1%) of the 2,171 active cryptojacking websites were among the Alexa Top 100K websites, which impact thousands of users. In fact, this analysis suggests that current preventions solutions are unable or unwilling to blacklist active cryptojacking websites including the ones with higher ranking for several weeks.

Cryptojacking Family	Occurrences	Dynamically Loaded
CoinHive	2,206 (35%)	227 (10.3%)
JSEcoin	1,271 (20.2%)	839 (66%)
Crypto-loot	766 (12.2%)	737 (96.3%)
ProsectPoi	348 (5.5%)	228 (65.5%)
CoinHave	321 (5.1%)	170 (53%)
CoinImp	227 (3.6%)	207 (91%)
CoinLab	202 (3.2%)	202 (100.0%)
DeepMiner	176 (2.8%)	162 (92%)
WebMine	139 (2.2%)	111 (80%)
Cloudcoins	57 (0.9%)	43 (76%)
MoneroMiner	53 (0.85%)	7 (13%)
CoinHive Captcha	28 (0.45%)	4(13%)
Inwemo	12 (0.19%)	11 (97%)
Monero.cc	10 (0.15%)	10 (100%)
Previously-Unknown Libraries	486 (7.7%)	214 (44%)
Total	6,302 (100%)	3,172(50%)

Table 7: The number of cryptojacking websites and their associated cryptojacking families in the Alexa Top 1 Million websites in 50% of the cases, the cryptojacking library was loaded from a remote server.

6 OPERATIONAL INSIGHTS

As a second case study, we dig into the operations of cryptojacking. We cluster the identified cryptojacking families, reveal previously unknown CoinHive-like services, extract mining account identifiers to group cryptojacking incidents. This aggregated data begins to reveal the higher-level behavior of the individual and organizational actors that run cryptojacking.

6.1 The Prevalence of Cryptojacking Families

To identify cryptojacking families that each cryptojacking website belongs to, we analyzed the downloaded wasm modules to see whether we can link cryptojacking families together. Although the names of wasm modules might vary for different cryptojacking operators, the content of the wasm module is less likely to differ for a given family. This suggests a relatively stagnant set of wasm binaries, which are typically compiled from C/C++. To assign a label to a given website, we considered two features: (1) the address of the server that hosted the cryptojacking library (similar to Wappalyzer), and (2) the hash of the wasm module used by the library. As displayed in Table 7, we found that these two metrics were sufficient to label 5816 (92.3%) of the detected websites into the 14 known cryptojacking families encountered during training data labeling. We utilized the raw number of discovered cryptojacking domains for each cryptojacking family as a proxy for the family's market share and found a broad spectrum of cryptojacking families with significant deployment. There is no single dominant entity, with the top 3 libraries, CoinHive, JSECoin, and Crypto-loot, accounting for 60.7% of the market. In total, 50% of all cryptojacking sites used non-standard hosts for their cryptojacking libraries, suggesting that the hosting server regexes found in certain blacklists are inherently reactive in their detection capabilities, and even if updated regularly, they can be easily thwarted.

We performed another experiment on the remaining 486 (7.7%) cryptojacking incidents and clustered them based on the address of the remote server that hosted the cryptojacking library. For example, Moonify (e.g., `moonify.min.js`) or Grindcash (`gridcash.js`) are CoinHive-like mining services which were not in our training

New Mining Servies	Occurrences	Main Mining Pool
WebXMR	208	xmrm.pw
CoinPot	81	coinpot.co
Coinblind	32	coinblind.com
Punchhub	21	punchsub.net
VidMiner	18	estream.nu
AJCryptominer	15	ajcryptominer.com
CoinNebula	11	coinnebula.com
Ad-miner	11	ad-miner.com
Cryptonoter	9	cryptonoter.com
Lightminer	8	lightminer.co
gridcash	8	gridcash.net
DigXMR	8	digxmr.com
Minercry	7	minercry.pt
Moonify	7	moonify.io
Grindcash	5	ulnawoyyzbljc.ru

Table 8: A subset of previously unknown cryptojacking services. 486 cryptojacking incidents were conducted by 24 new CoinHive-like services.

Campaign ID	No. of Sites	Main Mining service
pfr5E2eLd0VduZ1wKaesPFnccU9d2GZi	121	coinhive.com
Yq2af5ZaYuELhmPUH524q7sjiaCNCzXr	117	cloudfront.net
FGZZeDbNg5AMpILuz8fKWI4UJqdNIxce	23	xmrpool.xyz
l8Un4BkrkTfKQOEoCRyTtGMfhJFlibNu	16	semipool.com
XcgzWRYl8Lh3r3ElNSznHhP9HAcYtd4g	13	minemonero.pro
Dx8uX9lNo7a3kmGe55vO6bhWMquRd9YG	10	supportxmr.com
15217	7	aalbbh84.info
b7fbc3b2e3d88d9d766dc5ed27db53983ec759be76e2	5	crypto-loot.com
14285	4	bhzejltg.info

Table 9: A subset of cryptojacking campaigns and their corresponding mining pools. 16 out of 35 detected cryptojacking campaigns were using less expensive or free CoinHive-like mining services.

dataset, but Outguard identified instances of these libraries in our large-scale experiment. We identified 24 previously unseen coinhive like mining services. Table 8 shows a subset of these mining services as well as the main mining pool.

6.2 Campaign Analysis

Our analysis show that obtaining the *Site Key* is possible by searching in network traces and JS source code. While mining services allow generating several Site Keys per user, in practice, we found several instances of individual Site Keys being used on multiple websites simultaneously. We found 386 cryptojacking websites under 35 clusters based on unique Site Keys ranging in size from 2 domains to 121 domains. Table 9 illustrates a subset of these Site Keys. Our analysis shows that 16 out of 35 detected campaigns were using less expensive or free CoinHive-like mining services.

To get a flavor for the behavior of a cryptojacking campaign, we took a closer look at the largest CoinHive campaign. The 121 clustered domains were all registered under anonymous WHOIS services and pointed to a set of 23 IP addresses found in 15 ASes, which were mostly cloud-hosting services. The IPs were globally distributed in 7 countries including the US, Germany, Australia, Great Britain, China, Canada, and Switzerland. From Censys.io [3] we were able to determine that all IPs pointed to HTTP webservers running either openresty, nginx, or apached. The campaign's usage of multiple anonymous WHOIS services, globally distributed cloud hosting, and large number of fungible, human-meaningless domains (e.g. 5565925.com, 2zo.xyz, jipa.gdn,etc.) hint at evasion techniques characteristic of other web-based malware or PUPs.

7 DISCUSSION AND LIMITATIONS

In this paper, we construct and evaluate Outguard, a resilient system for detecting cryptojacking that finds nearly twice as many cryptojacking websites as existing blacklist solutions. As we demonstrated, a large fraction of websites have short-lived cryptojacking operations. Therefore, regular scans of the Alexa Top 1M or other domain lists using a system similar to Outguard can automatically update blacklists with the latest cryptojacking domains and remove stale domains.

While the evaluation of Outguard on real-world dataset shows that it works well in practice, it has three potential limitations. First, Outguard is subject to bias due to the degree of completeness of the public blacklists. Like other machine learning approaches, the effectiveness of Outguard depends on the quantity of the dataset that we used to generate the model. The hash function detection mechanism currently relies on static function name/fingerprints that can be circumvented by an evasive cryptojacking operator. We expect this to have limited impact based on our feature ranking analysis. In general, Outguard should be viewed as a lower bound estimate of the the full cryptojacking ecosystem.

Second, note that Outguard is a supervised learning technique. Cryptojacking operators may be able exploit this fundamental limitation and change their strategies to evade some of the features of the detection model. For example, the operators may find alternatives to using web sockets or web worker services. We plan to explore the practicality and profitability costs of these evasions to cryptojacking operators in future work. While not foolproof, Outguard is likely to detect a large number of evasion scenarios as shown in Section 3.5. We believe that Outguard raises the bar for adversaries to avoid cryptojacking detection and increases the development cost required to carry out these operations.

Finally, as mentioned earlier, cryptojacking occurs in many forms beyond in-browser mining. In this paper, we focused on in-browser cryptojacking as it is very easy for cryptojacking operators to develop and deploy. In contrast to other common attacks, in-browser cryptojacking does not require building specific binaries or creating drive-by download websites, which are more likely to be flagged by modern anti-malware solutions. Other forms of cryptojacking operations, such as cryptojacking browser extensions, desktop software, and mobile apps, are outside the scope of this study.

8 CONCLUSION

We find that in-browser cryptojacking libraries use browser resources in a very similar way. We show that if these traits are accurately modeled, it is possible to automatically identify a large number of in-browser cryptojacking operations. Towards that end, we developed a tool called Outguard, which is able to detect a significant number of cryptojacking websites and potentially disrupt the malicious operations of cryptojacking operators. Consequently, we conclude that defending against cryptojacking operations at scale is possible with minimal human intervention.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation (NSF) under grant CNS-1518741 award. We would like to thank the anonymous reviewers for their helpful comments.

REFERENCES

[1] Coinblockerlists. https://github.com/ZeroDot1/CoinBlockerLists. Accessed: 07-29-2018.

[2] Stratum mining protocol. https://en.bitcoin.it/wiki/Stratum_mining_protocol. Accessed: 07-29-2018.

[3] Censys. https://censys.io/, 2018.

[4] Minerblock extension. https://github.com/xd4rker/MinerBlock/blob/master/assets/filters.txt, 2018.

[5] Abraham, S., and Chengalur-Smith, I. An overview of social engineering malware: Trends, tactics, and implications. *Technology in Society 33* (2010), 188–196.

[6] Andersen, D. Mining money with monero ... and cpu vector intrinsics. https://da-data.blogspot.com/2014/08/minting-money-with-monero-and-cpu.html, 08 2014.

[7] Arshad, S., Kharraz, A., and Robertson, W. Include me out: In-browser detection of malicious third-party content inclusions. In *Proc. 20th International Conference on Financial Cryptography and Data Security (FC)* (2 2016), pp. 441–459.

[8] Caballero, J., Grier, C., Kreibich, C., and Paxson, V. Measuring pay-per-install: The commoditization of malware distribution. In *Proceedings of the 20th USENIX Conference on Security* (Berkeley, CA, USA, 2011), SEC'11, USENIX Association, pp. 13–13.

[9] Cialdini, R. B. *Influence: Science and Practice.* Writers of the Round Table Press, Boston, 2009.

[10] Cimpanu, C. Cryptojackers found on starbucks wifi network, github, pirate streaming sites. https://www.bleepingcomputer.com/news/security/cryptojackers-found-on-starbucks-wifi-network-github-pirate-streaming-sites/, 12 2017.

[11] CoinMarketCap. Top 100 cryptocurrencies by market capitalization. https://coinmarketcap.com/, 2018.

[12] CryptoMineDev. MinerBlock. https://chrome.google.com/webstore/detail/minerblock/emikbbbebcdfohonlaifafnoanocnebl?hl=en, 10 2018.

[13] Dan Goodin. Now even youtube serves ads with cpu-draining cryptocurrency miners. https://arstechnica.com/information-technology/2018/01/now-even-youtube-serves-ads-with-cpu-draining-cryptocurrency-miners/, 01 2018.

[14] David Maciejak. Cryptojacking: Digging for Your Own Treasure. https://www.fortinet.com/blog/threat-research/cryptojacking-digging-for-your-own-treasure.html, 2017.

[15] de Groot, W. Cryptojacking found on 2496 online stores. https://gwillem.gitlab.io/2017/11/07/cryptojacking-found-on-2496-stores/, 11 2017.

[16] Eskandari, S., Leoutsarakos, A., Mursch, T., and Clark, J. A first look at browser-based cryptojacking. *CoRR* (2018).

[17] Google Chromium. Chrome debugging protocol viewer. https://chromedevtools.github.io/devtools-protocol/, 2017.

[18] Google Development. Extending DevTools. https://developer.chrome.com/extensions/devtools, 2018.

[19] Haddadi, H. Fighting online click-fraud using bluff ads. *ACM SIGCOMM Computer Communication Review 40* (2010), 21–25.

[20] Hong, G., Yang, Z., Yang, S., Zhang, L., Nan, Y., Zhang, Z., Yang, M., Zhang, Y., Qian, Z., and Duan, H. How you get shot in the back: A systematical study about cryptojacking in the real world. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (2018), ACM, pp. 1701–1713.

[21] Huang, D. Y., Dharmdasani, H., Meiklejohn, S., Dave, V., Grier, C., McCoy, D., Savage, S., Weaver, N., Snoeren, A. C., and Levchenko, K. Botcoin: Monetizing stolen cycles. In *Proc. Network and Distributed System Security Symposium (NDSS)* (2014).

[22] Hunt, P. Inwemo. https://www.producthunt.com/posts/inwemo, 11 2018.

[23] Ian Hickson. W3C Working Draft: Web Workers. https://www.w3.org/TR/2009/WD-workers-20090423/, 4 2009.

[24] Jerome Segura. The state of malicious cryptomining. https://blog.malwarebytes.com/cybercrime/2018/02/state-malicious-cryptomining/, 2018.

[25] Kharraz, A., Arshad, S., Mulliner, C., Robertson, W., and Kirda, E. UNVEIL: A large-scale, automated approach to detecting ransomware. In *Proceeding of 25th USENIX Security Symposium* (2016), pp. 192–203.

[26] Kharraz, A., Kirda, E., Robertson, W., Balzarotti, D., and Francillon, A. Optical Delusions: A Study of Malicious QR Codes in the Wild. In *Proceeding of 44th International Conference of IEEE/IFIP Dependable Systems and Networks (DSN)* (2014), pp. 192–203.

[27] Kharraz, A., Robertson, W., Balzarotti, D., Bilge, L., and Kirda, E. Cutting the gordian knot: A look under the hood of ransomware attacks. In *Proceeding of 12th International Conference of Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)* (2015), pp. 3–24.

[28] Kharraz, A., Robertson, W., and Kirda, E. Surveylance: Automatically detecting online survey scams. In *Proceeding of IEEE Symposium on Security and Privacy (S&P)* (2018), pp. 70–86.

[29] Kolodenker, E., Koch, W., Stringhini, G., and Egele, M. Paybreak: Defense against cryptographic ransomware. In *Proceeding of ACM on Asia Computer and Communications Security (Asia CCS)* (2017), pp. 599–611.

[30] Konoth, R. K., Vineti, E., Moonsamy, V., Lindorfer, M., Kruegel, C., Bos, H., and Vigna, G. Minesweeper: An in-depth look into drive-by cryptocurrency mining and its defense. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (2018), ACM, pp. 1714–1730.

[31] Kotzias, P., Bilge, L., and Caballero, J. Measuring pup prevalence and pup distribution through pay-per-install services. In *Proceeding of 25th USENIX Security Symposium* (2016).

[32] Kührer, M., Rossow, C., and Holz, T. Paint it black: Evaluating the effectiveness of malware blacklists. In *Proceeding of 17th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)* (2014), pp. 1–21.

[33] Kwon, B. J., Mondal, J., Jang, J., Bilge, L., and Dumitras, T. The dropper effect: Insights into malware distribution with downloader graph analytics. In *Proceeding of 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2015), pp. 1118–1129.

[34] Li, Z., Zhang, K., Xie, Y., Yu, F., and Wang, X. Knowing your enemy: understanding and detecting malicious web advertising. In *Proceedinf of 2012 ACM SIGSAC conference Computer and Communications Security (CCS)* (2012), pp. 674–686.

[35] Lindsey O'Donnell. Cryptojacking attack found on los angeles times website. https://threatpost.com/cryptojacking-attack-found-on-los-angeles-times-website/130041/, 02 2018.

[36] Lindsey O'Donnell. Cryptojacking Campaign Exploits Drupal Bug, Over 400 Websites Attacked. https://threatpost.com/cryptojacking-campaign-exploits-drupal-bug-over-400-websites-attacked/131733/, 2018.

[37] Liu, C., and Chen, J. C. Malvertising campaign abuses googleâĂŹs doubleclick to deliver cryptocurrency miners. https://blog.trendmicro.com/trendlabs-security-intelligence/malvertising-campaign-abuses-googles-doubleclick-to-deliver-cryptocurrency-miners/, 01 2018.

[38] Malwarebytes Inc. A look into the global drive-by cryptocurrency mining phenomenon . https://go.malwarebytes.com/rs/805-USG-300/images/Drive-by_Mining_FINAL.pdf, 2017.

[39] Miramirkhani, N., Starov, O., and Nikiforakis, N. Dial one for scam: A large-scale analysis of technical support scams. In *Proceeding of Network and Distributed System Security Symposium (NDSS)* (2017).

[40] Musch, M., Wressnegger, C., Johns, M., and Rieck, K. Web-based cryptojacking in the wild. *arXiv preprint arXiv:1808.09474* (2018).

[41] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. http://www.bitcoin.org/bitcoin.pdf, 2009.

[42] Nelms, T., Perdisci, R., Antonakakis, M., and Ahamad, M. Towards measuring and mitigating social engineering software download attacks. In *Proceedings of 25th USENIX Security Symposium* (2016).

[43] Newman, L. H. Hack brief: Hackers enlisted tesla's public cloud to mine cryptocurrency. https://www.wired.com/story/cryptojacking-tesla-amazon-cloud/, 02 2018.

[44] Nolen Scaife, Henry Carter, P. T., and Butler, K. CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data. In *IEEE International Conference on Distributed Computing Systems (ICDCS)* (2016).

[45] Rafael Keramidas. No Coin. https://github.com/keraf/NoCoin/blob/master/src/js/background.js, 2018.

[46] Rüth, J., Zimmermann, T., Wolsing, K., and Hohlfeld, O. Digging into browser-based crypto mining. *arXiv preprint arXiv:1808.00811* (2018).

[47] scikit-learn. scikit-learn: machine learning in Python. https://github.com/scikit-learn/scikit-learn, 2018.

[48] Thomas, K., Crespo, J. A. E., Rasti, R., Picod, J.-M., Phillips, C., Decoste, M.-A., Sharp, C., Tirelo, F., Tofigh, A., Courteau, M.-A., Ballard, L., Shield, R., Jagpal, N., Rajab, M. A., Mavrommatis, P., Provos, N., Bursztein, E., and McCoy, D. Investigating commercial pay-per-install and the distribution of unwanted software. In *USENIX Security Symposium* (2016).

[49] tunghobrens. AntiMiner – No 1 Coin MinerBlock. https://chrome.google.com/webstore/detail/anti-miner-no-1-coin-mine/ibhpgkhoicjhklmbhdoeikeggbeejonj?hl=en, 2018.

[50] Tuwiner, J. Bitcoin mining pools. https://www.buybitcoinworldwide.com/mining/pools/. accessed: 01-29-2019.

[51] Vadrevu, P., Rahbarinia, B., Perdisci, R., Li, K., and Antonakakis, M. Measuring and detecting malware downloads in live network traffic. In *Proceeding of European Symposium on Research in Computer Security* (2013), Springer, pp. 556–573.

[52] Wappalyzer. MinerBlock. https://www.wappalyzer.com/download. accessed: 01-23-2019.

[53] Whaley, B. Toward a general theory of deception. *The Journal of Strategic Studies 5* (1982), 178–192.

[54] World Wide Web Consortium. WebAssembly Specifications. https://webassembly.github.io/spec/, 2018.

[55] Wyke, J. Zeroaccess botnet – mining and fraud for massive financial gain. https://www.sophos.com/en-us/medialibrary/PDFs/technical%20papers/Sophos_ZeroAccess_Botnet.pdf, 2012.

[56] Zarras, A., Kapravelos, A., Stringhini, G., Holz, T., Kruegel, C., and Vigna, G. The dark alleys of madison avenue: Understanding malicious advertisements.

In *Proceeding of 2014 ACM Internet Measurement Conference (IMC)* (2014), pp. 373–380.