Cryptoblocker: Dynamically Detecting in-Browser Cryptocurrency Mining

David Gibson
Harvard University
Cambridge, Massachusetts
davidgibson@college.harvard.edu

Thomas Chang
Harvard University
Cambridge, Massachusetts
tchang@college.harvard.edu

ABSTRACT

Client-side execution of Javascript presents a tempting vector for launching attacks, and recent rises in the value of cryptocurrencies provide an easy way to monetize extremely simple attacks. We present *Cryptoblocker*, an extension for Google Chrome aimed at detecting mining behavior and allowing users to defend against mining attacks. It utilizes two layers of defense: blacklists, which are provided by current extensions aiming to solve the same problem, and dynamic analysis of CPU usage patterns. This second defense vector of dynamic CPU usage analysis presents a powerful new protection mechanism for ordinary users against cryptojacking attacks. In this paper, we discuss our implementation of these defense mechanisms in a simple, easy to use extension that adds little to no noticeable overhead.

1 INTRODUCTION

A key feature of Cryptocurrencies are computationally intensive hash based puzzles, which are solved to verify blocks of transactions (mining a block). Solving one of these puzzles before other miners yields a small payment, which provides an incentive for miners to spend a large amount of computational power to mine blocks before other miners[3]. One method of acquiring more computing power is by embedding mining JavaScript code in a website, so that visitors to the website will execute client side mining code in their web browser. While some pages ask for user consent before mining, many websites will embed mining code and harvest client side computing power without the user's knowledge. The practice of using client browsers for cryptocurrency mining is known as cryptojacking. Not only is cryptojacking tricking clients into making money for the script owner, cryptojacking can be harmful for a client's computer because extended periods of high load can cause the CPU to be subject to excessive heat for a long period of time[13].

This paper describes *Cryptoblocker* , a Google Chrome browser extension that detects and kills tabs that are mining cryptocurrency. When a potential cryptojacking process is found, *Cryptoblocker* will inform the user and prompt them to either kill the offending tab or to resume normal browsing. *Cryptoblocker* protects against cryptojacking by using current blacklisting approaches as well as dynamic analysis of CPU usage. An important objective for our dynamic analysis is that it should detect mining behavior without interrupting normal browsing activity. This objective means that our extension must not flag computationally intensive behavior that occurs from normal browsing.

Another goal of *Cryptoblocker* is to provide minimal overhead to normal web browsing. The most computationally intensive points of *Cryptoblocker* occur at the initial loading of a page, so we analyze the difference in load times caused by our extension. We compare load times on many popular and complex websites to show that *Cryptoblocker* incurs minimal overhead.

2 BACKGROUND

2.1 Mining Cryptocurrency

Cryptocurrency mining extends the block chain, which dictates the distribution of currency among all of a cryptocurrency's users. To add a block to the block chain, a miner must take a set of transactions and hash them with a nonce to generate a hash that meets a certain set of conditions[3]. The randomness of hash functions requires a miner to simply guess nonces that will hash to the appropriate conditions, and the large range that hash functions can output forces many hashes to be computed.

2.2 Cryptojacking

Browser based cryptocurrency mining exists in two forms: opt-in and cryptojacking. For opt in sites, browser based mining has been framed as an alternative stream of revenue to promote an ad free experience[9][6][4]. Typically in this scenario, the website will present the client with a prompt or terms of service informing the user that their site will be using the client's web browser to mine cryptocurrency, after which the client can either agree to mine or return to a version of the site with ads. An example of a site that subscribes to this methodology is the news outlet `salon.com` (although Salon has publicly stated that they use browser based mining their homepage does not present the user with and option and immediately starts mining). Another example of an opt-in mining website is `thehopepage.org`, a website run by UNICEF Australia that asks clients to mine cryptocurrency for charity.

The second kind of browser mining websites are cryptojacking sites that do not disclose to the client they are mining cryptocurrency and usually attempt to hide their behavior. While this illicit mining typically occurs on smaller, shadier websites, cryptojacking also occurs on large domains. Recently the popular video hosting site `worldstarhiphop.com` was found cryptojacking but has since removed its mining script [12]. In a recently published list 16 websites in the top 10,000 Alexa ranked pages were found to contain known cryptojacking scripts [15].

In either of these two cases, however, the mining website will employ some existing API to handle the mining code. The most popular API service in this field is Coinhive, but there are many other APIs such as Crypto-Loot, CoinImp, Minr, and deepMiner[10]. These APIs provide mining scripts to embed with a SITE-KEY in a website. All clients visiting the page will run the embedded script and mine for the planted SITE-KEY, which maps to the website's wallet. These APIs typically mine the currency Monero, a cryptocurrency optimized to run on consumer machines. [1]

2.3 Google Chrome Dev Channel

Currently, Google Chrome only allows extensions access to general CPU usage statistics for the whole system. In theory these could be tracked to attempt to identify mining behavior, but this signal is simply too noisy in practice, given the infeasibility of differentiating between browser mining and the many other legitimate behaviors that could occur on a system and change system-wide CPU usage. Support for process-by-process CPU usage statistics is included in the Dev Channel build of Google Chrome, however [2]. The Dev Channel is a prototype build used for testing future features and offers extended developer support. Our extension utilizes these process-by-process CPU usage statistics extensively, and as such can currently only be run on the Chrome Dev Channel.

3 RELATED WORK

In the literature of Cryptojacking there have been several proposed defenses. Although only one method is widespread, research has been conducted that shows more advanced analysis can operate at a high degree of accuracy. Our primary contribution is an extension that performs dynamic analysis without the aid of an external application. Current dynamic analysis either requires an additional local monitoring application or knowledge of what type of script is being run. Local monitoring applications require extensive set-up and incur overheads in both performance and especially convenience, limiting them to research purposes and making them largely infeasible for widespread consumer use. Our extension is entirely stand alone and has the potential to identify mining code without prior knowledge of the semantics of the script that it is trying to detect.

[1]https://coinhive.com/documentation/simple-ui

3.1 Available Extensions

The first and only widely available methodology for protecting against Cryptojacking is blacklisting. Currently there are a number of other extensions available on the Chrome extension store that use blacklisting[14][8]. Blacklisting keeps a list of URLs that are known to distribute mining code and flags any requests that attempt to retrieve scripts from these sites. While simple, this method is effective because a large amount of mining code originates from browser mining APIs. Blacklisting can effectively disable an API's URLs, which would then require the attacker to keep an unknown proxy for receiving their scripts. Blacklisting essentially creates a race between attacker and defender. An attacker must be able to find new proxies or hosts to keep malicious scrips faster than a defender can identify them. For the most part blacklists are community managed ventures. For example one major list, CoinBlockersList, is a project by the GitHub user zerodot1[16]. While acquiring new proxies and hosts can be costly for attackers, the small community effort for managing blacklists can still be overwhelmed and should not serve as a long term solution.

3.2 Network Analysis

One method of advanced dynamic analysis is examining network patterns. Saad, Khormali, and Mohaisen [13] showed that the network protocol cryptojackering APIs use to communicate computed hashes can be reverse engineered and detected. After figuring out the network protocol of the CoinHive API they were able to create an extension that detects CoinHive traffic even when the CoinHive scripts are obfuscated or sourced from a proxy to avoid blacklisting. By analyzing packet exchanges the extension can identify a hash exchange protocol for an API and then stop the script from mining.

3.3 Dynamic Analysis from System Statistics

The second method of more advanced dynamic analysis involves using system statistics to determine whether a cryptomining script is running. Musch et al. [11] developed a Node.js application that crawls the web and detects miners by examining many heuristics of likely miners. When a likely miner is detected further dynamic analysis is conducted to confirm that a script is a miner. One notable characteristic of this Node.js crawler is that it can find high usage in one specific JavaScript function. Because mining code spends most of its time hashing, knowing that a script is using one function in particular is a stronger indicator that a script is mining. Once a mining script has been confirmed through a longer dynamic analysis the code of the offending script can be isolated and used as a feature for static analysis in the future. While these advanced heuristic methods are very thorough, they require an external application, which bars them from seeing widespread consumer use.

3.4 Static Analysis

Although not much static analysis has been done for cryptojacking specifically, static analysis of malicious JavaScript is a well researched topic. For the application *Zozzle*, machine learning methods are applied to classify malicious JavaScript from features derived from a JavaScript abstract syntax tree[5]. Some difficulties with acquiring these features are deobfuscating code and making an appropriately large dataset without already having a classifier. The primary advantage to using static methods is the speed of classification and lack of a requirement to execute code. In analyzing malicious malware it can be important to not let code execute, so being able to classify code without running it is highly advantageous.

4 THREAT MODEL

To identify when cryptocurrency mining is happening we identify static and dynamic characteristics of mining scripts. Statically,the origin and identifying strings of a script can be used to pin it as a mining script. Dynamically, the CPU usage pattern of mining scripts can be contrasted with the usage pattern of computationally intensive tasks undertaken during normal browsing.

4.1 Embedded Scripts and Obfuscation

One way to identify cryptojacking scripts is by looking for known strings in JavaScript code and known distributors of cryptojacking code. Because most cryptojacking sites use a cryptojacking API like Coinhive, the majority of malicious mining JavaScript is structured with similar keywords and is retrieved from those services API URLs. These characteristics mean that cryptojacking scripts can be identified by searching for known classes like "Coinhive.Anonymous" or by looking for scripts that are requested from Coinhive servers. We employ the latter method as a blacklist to deny websites from retrieving scripts from known cryptojacking distributors.

Code obfuscation is used to evade identification from known strings. More advanced scripts will obfuscate their scripts as blocks of hex instructions making this kind of simple static analysis impossible [10]. While obfuscation can be reverse engineered, determining the semantics of obfuscated code is a challenging task[13]. Additionally an attacker could evade detection from a blacklist by hosting the scripts on their own servers and using their own servers as a proxy[13]. In this situation, the cryptojacking scripts are sourced from the webserver itself so a API service cannot be easily detected. To overcome these shortcomings, we use dynamic analysis.

4.2 CPU Usage Pattern

Mining cryptocurrency is a computationally intensive task. Because this statement is unavoidable for cryptojackers, mining scripts must cause a significant increase to CPU load to be effective. To characterize the CPU usage patterns of cryptojackers we found a few websites with verified mining scripts and compared their CPU usage to normal browsing. In Figure 1 we plot the CPU usage of websites when viewed as the active tab (unless specified as background) with no user input and no other tabs open. Initially all pages have high CPU usage due to page load but they all stabilize after a short period of time. After loading, all miners stabilize to significantly higher average CPU loads than intensive browsing activities like streaming video. Within miners there are two different usage patterns: high consistent load and high variable load. `oixrysoi.net` and `salon.com` examples are miners that consistently consume a large amount of CPU usage. This behavior is in line with mining code that is written in a tight loop but limited to use a specific amount of CPU. The other CPU usage patten can be seen in `filmyhd.cz`. This website has extremely high usage and is also extremely variable. A key observation is that the mining scripts all use much more CPU than normal browsing, which allows differentiation between mining behavior.

Another difference between normal browsing and miner behavior is the behavior of a tab when it is inactive. YouTube has significantly lower CPU usage when it is an inactive tab. While this is especially true for video playback because it does not have to render to the screen, idling inactive tabs lower their CPU usage as well. Mining code, however, uses the CPU whether the tab is active or not because hashes still demand the same amount of processing power. Figure 2 illustrates this difference contrasting the CPU usage of `oixrysoi.net` and YouTube when watching a video.

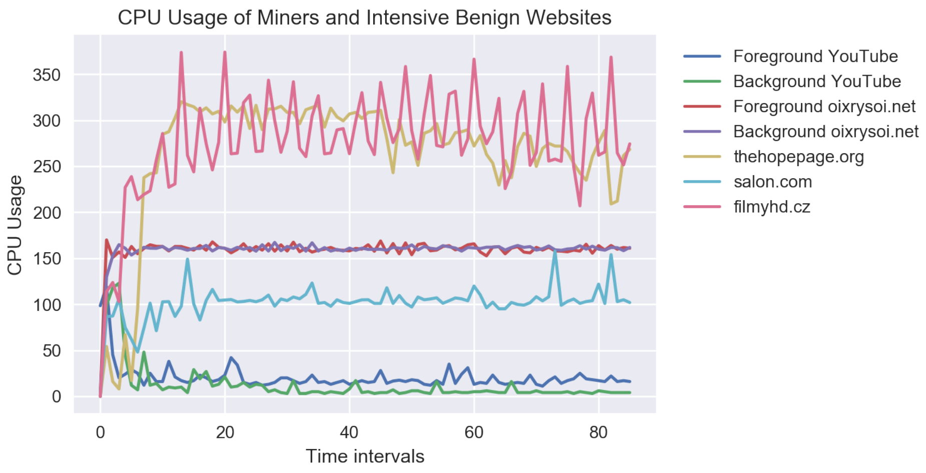

Figure 1: CPU Usage plots of known miners and intensive regular browsing.

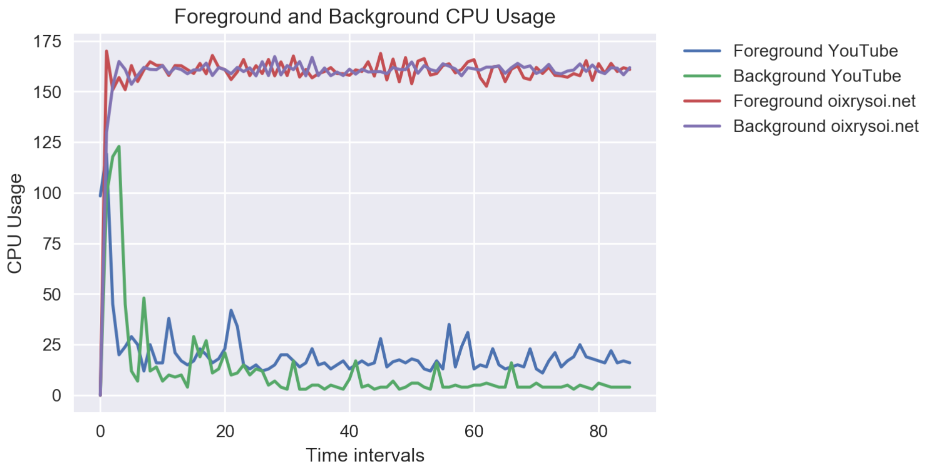

Figure 2: CPU Usage plots of Foreground and Background Tabs for Miners and Benign browsing.

5 IMPLEMENTATION

Our implementation consists of two layers of defense: A blacklist of known mining script sources, and a heuristic-based dynamic analysis of Chrome's per-process CPU usage

5.1 Blacklist

The first layer of defense against cryptojacking is a simple blacklist of known sources of mining JavaScript code. Every request made by the browser is compared against a static list of known malicious sources, and *Cryptoblocker* denies any requests made to sources on the list. Even though blacklists can be avoided relatively easily by either hiding script sources behind proxies or simply using inline JavaScript, many websites do not even bother with these simple anti-detection measures. A blacklist alone is therefore enough to stop a large amount of cryptojacking at little cost to the user. When the effectiveness of blacklists is combined with their minimal overhead, they become the single best defense against cryptojacking. Our blacklist was sourced from a list of over 14,000 cryptojacking sources curated by Github user ZeroDot1 [16].

5.2 Dynamic CPU Usage Analysis

Our second layer of defense is a set of heuristics designed to identify suspicious CPU usage patterns. *Cryptoblocker* uses the `processes` API found in the Chrome Dev Channel to record CPU usage on a per-Chrome-processes basis. The `processes` API offers measurements of the CPU usage of each Chrome process updated roughly once per second, which we use to perform our dynamic analysis. These CPU

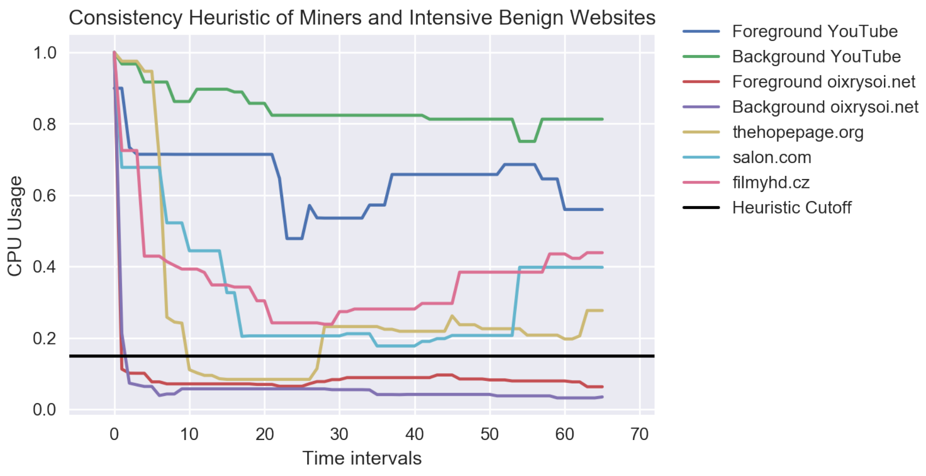

Figure 3: Consistency test statistic for tracked websites.

usage measurements are given as percentages of a single CPU core used by all threads in a process, and can thus exceed 100 in certain cases with multiple cores and multiple threads.

All miners must use a certain amount of computational power to execute their algorithms, and we leverage this fact in our heuristics. For our window of analysis of 15 measurements, if the measurements are simply too low (which we define by the presence of a single measurement lower than 5), we deem the processes as clean. It is possible that *Cryptoblocker* could therefore miss an extremely conservative and throttled miner, but such a miner would offer little benefit to the owner and present little harm to the user, so we choose to focus our attention on more high profile mining scripts. A clever adverserial attacker might be able to avoid detection by frequently reducing usage to levels around 0, but such fine grained control over client side CPU usage is difficult. This pattern of behavior would also be more similar to many benign processes, and therefore much harder to detect with our methods, so we accept *Cryptoblocker* 's inability to flag these specific mining scripts. Given the threat model, we thus identified several patterns of CPU usage as suspicious: extremely high use by a single processes, high use by a background process, and low variance in use. All three of these patterns are almost nonexistant in benign processes, and at least one of them is present in almost every mining process.

5.2.1 General High CPU Usage. Extremely high CPU use by any processes is a good indicator of potential mining behavior, but many benign processes also use high amounts of CPU resources, such as video streams and graphics. We therefore had to set our usage threshold quite high, and only flagged a process as suspicious if it recorded 15 measurements of CPU usage over 150 in a row. The 150 threshold was based off our evaluations of several benign but intensive CPU behaviors, such as video streaming, as well as off of measurements of the usage of actual mining scripts. As shown in Figure 1, this catches some but not all of the plotted mining scripts, while staying well above the usage patterns of normal intensive browsing behaviors. This is a quite conservative threshold, and only flags extremely overt miners who do not use any throttling parameters.

5.2.2 High Background CPU Usage. Our second heuristic involves monitoring the CPU usage of background processes, and is designed to catch more conservative mining behavior. The vast majority of benign processes experience significantly decreased CPU usage when their respective tab is inactive, given that a large amount CPU usage is related to visually displaying the contents of an active tab. Miners, on the other hand, almost always continue their computation regardless of whether their tab is active or not. To check for this, we set a lower CPU usage threshold for processes corresponding to inactive tabs, and flag a process as suspicious if it has eight consecutive usage measurements of 50 or more while its tab is inactive. Since most miners do not decrease CPU usage while running in the background, this lower threshold of 50 as compared to the active tab threshold of 150 is enough to catch the majority of other miners such as that of `salon.com`.

5.2.3 Consistent Usage Pattern. Our final heuristic monitors the consistency of CPU usage. Because most benign processes have a wide range of usage and most miners have a relatively consistent pattern, we can use the range of CPU usage measurements in a given window of time to detect potential mining behavior. For every window of 15 consecutive usage measurements, we check the range of the data in the

Table 1: Average Page Load Times (n=32)

	Base	Extension	% increase	p-value
google.com	0.225s	0.230s	0.024s	0.360
facebook.com	0.904s	0.981s	0.078s	0.185
youtube.com	1.556s	1.572s	0.010s	0.486

window then divide by the maximum usage measurement in the window in order to normalize the range measurement. If this value ever drops below 0.15, indicating extremely consistent usage patterns, we alert the user of suspicious activity. This threshold was conservatively chosen, but as shown in Figure 3, most mining scripts hover around this threshold. Simply dropping below it once is enough to flag suspicious behavior warnings, and so this heuristic does end up catching many miners.

6 EXPERIMENTAL RESULTS

6.1 Overhead

A key goal of *Cryptoblocker* was to add little to no computational overhead, as measured by changed page load times. An noticeable increase in page load on every website visited by a user would be a prohibitive cost and would destroy the general useability of the extension. To test overhead, we used `Selenium` to repeatedly load specific webpages with a range of various load times with and without the *Cryptoblocker* extension, and analyzed the results for potential differences in page load times (Table 1). We found no evidence of a significant difference in page load times, and the added load time from the extension was minimal. In the worst case, it increased load times by 7% for a single website, but this appears to be an outlier as the other websites tested showed extremely minimal increases in page load times.

To test for false positives and true negatives, we visited the top 100 websites as determined by Alexa Ranking [1], spending 30 seconds on each page. Some examples of these websites are `google.com` and `amazon.com`. They are extremely high profile and we operated under the assumption that none of them contained mining scripts. After testing, none of these sites had suspicious behavior patterns as determined by our dynamic analysis heuristics. One of the sites, `mail.ru`, triggered our blacklist and had a request blocked [2], but the request appeared to be to an ad server, and usage of the page was unhindered. We also manually tested several behaviors with known high CPU usage requirements, including watching Youtube videos and streaming video content, none of which triggered mining behavior alerts.

Testing for true positives and false negatives was substantially more challenging, mainly because we had a difficult time collecting a set of sites known to host miners. We identified two sites, `salon.com` and `thehopepage.org`, which acknowledged using Javascript based miners. Our extension was able to both block their mining code using the blacklist, and also identify and kill the offending processes through dynamic heuristic analysis when we disabled the blacklist in order to analyze their specific mining behavior. We identified four websites that attempted to engage in true cryptojacking by hiding mining behavior [3], and*Cryptoblocker* 's heuristic analysis was successful in flagging all four. All four sites engaged in extremely shady behavior and were only accessed through a virtual machine as a safety precaution.

7 DISCUSSION

The most notable missing element in this paper is a more thorough analysis of *Cryptoblocker* 's effectiveness in finding true positives in known cryptojacking websites. This mainly resulted from our inability to find a sizeable test set of such websites. Curating a set of websites known to contain mining scripts turned out to be oe of the most difficult parts of the project. Approximately 80% of all web-based cryptocurrency miners use Coinhive's implementation [7]. However, all Coinhive-based mining scripts we found on websites failed to execute properly. They almost universally failed to connect to `coinhive.com` proxy servers in order to communicate computation results, and crashed rather than fully executing. This made our search far more difficult and was a primary cause of our struggle to find cryptojacking websites.

One convenient aspect of this particular classification problem is the relatively small cost of false positives. Unlike in cases such as cancer detection, where extremely small numbers of false positives can be prohibitive, a false positive in the task of cryptojacking identification is relatively painless. If the false positive comes from an obviously known benign site known to be performing significant computation, the user can simply ignore it. If the alert comes from a shady site that is more likely to actually be hosting a cryptocurrency miner, the cost of simply using a different site is usually quite low. Most websites have plentiful available alternatives, and stopping visiting one despite a potentially false positive diagnosis is reasonable and not an extreme inconvenience for the most part.

7.1 Future Work

Most future improvements to *Cryptoblocker* revolve around finding a suitable set of known cryptojacking websites. In particular, this would allow us to pursue a machine learning approach to identifying mining behavior in addition to or

[2]to https://prg.smartadserver.com/prebid/v1

[3]`oixrysoi.net`, `ekinomaniak.tv`, `filmyhd.cz`, and `megapastes.com`

perhaps instead of the heuristics we currently use. This problem of classifying CPU usage logs into benign or malicious behavior is well suited to a machine learning approach, as it is likely there are many patterns present in the data that our heuristics miss. The prospective problem formulation is also quite clear, with well defined inputs and outputs and an obvious way to collect them, given the existence of a set of known cryptojacking websites. Furthermore, this specific type of approach has been underrepresented in related projects, with most machine-learning approaches to the problem of malicious JavaScript focusing on static analysis of code rather than dynamic logs of CPU usage [5].

8 CONCLUSION

In conclusion, we have presented *Cryptoblocker* , a Chrome browser extension designed to protect users from cryptojacking exploits. *Cryptoblocker* differs from current Chrome extensions by implementing process-by-process CPU usage heuristics to detect cryptocurrency mining scripts in addition to utilizing a blacklist. Furthermore, it offers all its functionality with next to no computational overhead, having no noticeable effect on page load times.

REFERENCES

[1] [n. d.]. https://www.alexa.com/topsites

[2] [n. d.]. Chrome Dev. https://www.google.com/chrome/dev/?platform=win64

[3] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. 2015. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. *2015 IEEE Symposium on Security and Privacy* (2015), 104–121.

[4] Eric Chong. 2018. The Growing Trend of Coin Miner JavaScript Infection. https://www.fortinet.com/blog/threat-research/the-growing-trend-of-coin-miner-javascript-infection.html

[5] Charlie Curtsinger, Benjamin Livshits, Benjamin G. Zorn, and Christian Seifert. 2011. ZOZZLE: Fast and Precise In-Browser JavaScript Malware Detection. In *USENIX Security Symposium.*

[6] Jeff Edwards. [n. d.]. How to Detect and Stop Cryptomining on Your Network. https://blog.ipswitch.com/how-to-detect-and-stop-cryptojacking-on-your-network

[7] Shayan Eskandari, Andreas Leoutsarakos, Troy Mursch, and Jeremy Clark. 2018. A First Look at Browser-Based Cryptojacking. *2018 IEEE European Symposium on Security and Privacy Workshops* (2018), 58–66.

[8] keraf. [n. d.]. NoCoin. https://github.com/keraf/NoCoin/

[9] Maria Korolov. 2018. How to detect and prevent crypto mining malware. https://www.csoonline.com/article/3267572/encryption/how-to-detect-and-prevent-crypto-mining-malware.html

[10] Troy Mursch. 2018. How to find cryptojacking malware. https://badpackets.net/how-to-find-cryptojacking-malware

[11] Marius Musch, Christian Wressnegger, Martin Johns, and Konrad Rieck. 2018. Web-based Cryptojacking in the Wild. *CoRR* abs/1808.09474 (2018).

[12] Pixalate. [n. d.]. Websites with Coinhive JavaScript. http://info.pixalate.com/websites-with-coinhive

[13] Muhammad Saad, Aminollah Khormali, and Aziz Mohaisen. 2018. End-to-End Analysis of In-Browser Cryptojacking. *CoRR* abs/1809.02152 (2018).

[14] xd4rker. [n. d.]. MinerBlock. https://github.com/xd4rker/MinerBlock

[15] Xu Yang. 2018. The List of Top Alexa Websites With Web-Mining Code Embedded on Their Homepage. https://blog.netlab.360.com/the-list-of-top-alexa-websites-with-web-mining-code-embedded-on-their-homepage/

[16] ZeroDot1. [n. d.]. CoinBlockersList. https://zerodot1.gitlab.io/CoinBlockerListsWeb/index.html