Dissecting Android Cryptocurrency Miners

Stanislav Dashevskyi† Forescout Technologies Eindhoven, Netherlands stanislav.dashevskyi@forescout.com
Yury Zhauniarovich Independent Researcher Uzda, Belarus yury@zhauniarovich.com
Olga Gadyatskaya‡ LIACS, Leiden University Leiden, Netherlands o.gadyatskaya@liacs.leidenuniv.nl
Aleksandr Pilgun SnT, University of Luxembourg Esch-sur-Alzette, Luxembourg aleksandr.pilgun@uni.lu
Hamza Ouhssain‡ ARHS Developments Sanem, Luxembourg hamza.ouhssain@arhs-developments.com

ABSTRACT
Cryptojacking applications pose a serious threat to mobile devices. Due to the extensive computations, they deplete the battery fast and can even damage the device. In this work, we make a step towards combating this threat. We collected and manually verified a large dataset of Android mining apps. In this paper, we analyze the gathered miners and identify how they work, what are the most popular libraries and APIs used to facilitate their development, and what static features are typical for this class of applications. Further, we analyzed our dataset using VirusTotal. The majority of our samples is considered malicious by at least one VirusTotal scanner, but 16 apps are not detected by any engine; and at least 5 apks were not seen previously by the service.

Mining code could be obfuscated or fetched at runtime, and there are many confusing miner-related apps that actually do not mine. Thus, static features alone are not sufficient for miner detection. We have collected a feature set of dynamic metrics both for miners and unrelated benign apps, and built a machine learning-based tool for dynamic detection. Our BRENNTDROID tool is able to detect miners with 95% of accuracy on our dataset.

CCS CONCEPTS
• Security and privacy → Malware and its mitigation: Mobile platform security; • Social and professional topics → Malware / spyware crime; • General and reference → Empirical studies.

ACM Reference Format:

1 INTRODUCTION
The recent wave of cryptocurrencies contributed to the debut of a new malware class called cryptominers, cryptojackers, or simply miners. After infecting a device, these malicious applications start solving computationally hard puzzles that support the cryptocurrency network, getting rewards for their work that are accumulated on the miner developer’s account. The ease of monetization and the anonymity factors enabled the quick growth of the mining malware. In 2017, the skyrocketing price of cryptocurrencies caused by the enormous attention to these technologies has played a role in the cryptojacking proliferation [11]. Not surprisingly, miners have quickly gained popularity and appeared among the top security threats in 2018 [8, 29]. Security researchers have also paid attention, with many papers focusing on browser-based and binary mining [12, 14, 18, 20–22, 25–27] appearing recently.

Due to the cryptocurrencies boom, end-user demand for mining applications has emerged. Prior to July 2018 everybody could simply find mining applications on Google Play and attempt to generate a few cryptocoins on the mobile device. Yet, as the smartphone-based mining no longer generates interesting profits for the benign user [5, 27], the interest to these apps has significantly diminished. Google has removed mining apps from Google Play, but they are still available on alternative markets. The “crash” of the cryptocurrency market in the end of 2018 forced the operation of several mining services to shut down. For instance, recently the popular service CoinHive has announced discontinuation of its service [6]. Still, there are many alternatives, like CryptoLoot and JSEcoint, which are important cyber threats today1 and that may further proliferate during the next crypto boom. Due to the rising price of the Monero coin, in Summer 2019 the cryptominer malware was revitalized2.

The Android ecosystem itself is huge, comprising not only mobile devices but also wearable technology, TVs and cars. It is therefore a lucrative target for adversaries due to the large number of potential applications.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

1https://www.helpnetsecurity.com/2019/04/10/cryptomining-still-dominates/
victims. Even smart TV appliances can now be infected with mining Android apps\textsuperscript{3}. Security industry reports mention that mining capabilities are being introduced to existing malware families or added into repackaged Android applications [8, 11, 29]. Attackers are constantly looking into different ways to deliver malware to Android-based devices. For instance, recently cryptomining malware was distributed in 21 different countries via open Android Debug Bridge (ADB) ports\textsuperscript{4}.

Indeed, mobile mining has certain advantages for the attackers. By running in the background and when the user is not present, the mining code packaged as an app can be more persistent than website visits. The cost of creating and distributing a miner is negligible, given the ease of application repackaging [28] on Android and the availability of mining libraries [18]. But miners are particularly dangerous for mobile devices. The extensive computations performed during the mining process drain the battery and increase the temperature of the device, potentially causing irreversible damage. For example, a malicious family called Loapi causes the mobile device’s battery to overcook within 48 hours after the infection [17]. Therefore, there is a need to study the Android miner phenomenon and to be able to detect such applications.

Contributions. To the best of our knowledge, our paper presents the first large study of Android miners. We make the following contributions.

- We have collected a large dataset of mining Android apps, which includes both Web-based and binary-based cryptocurrency miners. As our focus is on the Android mining phenomenon, our dataset contains malicious mining applications, and also honest miners that declare their mining activity upfront and could be solicited by the users. We also include properly labelled samples of non-mining applications that can confuse the basic detection approaches (scam, wallet apps, etc.). Our dataset has been fully confirmed by manual analysis. We share our labelled dataset and the metadata with the community\textsuperscript{5}.
- We share insights on how (JavaScript) and binary Android miners work, how the mining code is injected, and what are the most popular libraries/APIs for mining. Particularly, we have identified 8 common mining libraries that are used in 671 miners.
- At least 5 miners from our dataset have previously never been uploaded to the popular VirusTotal service. We have also found 16 apps, including both malicious and honest miners, that are not detected by any VirusTotal scanner. Finally, we have ranked the antivirus engines at VirusTotal based on our dataset.
- Using our verified dataset as the ground truth, we performed dynamic analysis of the miners and compared the results with randomly selected benign applications. We identified a set of dynamic metrics that are the most efficient for accurate classification results, achieving 95% of accuracy and the AUC score of 0.988±0.009. Based on our findings, we propose the BRENNT-DROID tool that can be used to detect miners dynamically and to check if an app indeed mines cryptocurrencies.

\textsuperscript{3}http://blog.netlab.360.com/adl-miner-more-information-en/
\textsuperscript{5}The dataset is available upon request at https://standash.github.io/android-miners-dataset/

\begin{verbatim}
1 <script src="https://coinhive.com/lib/coinhive.min.js"></script>
2 <script>
3 var miner = new CoinHive.Anonymous("SITE_KEY");
4 miner.start();
5 </script>
\end{verbatim}

\textbf{Listing 1: JavaScript miner initialization example}

2 ANDROID CRYPTOCURRENCY MINING

There exist two approaches for mining cryptocurrencies on mobile devices: (1) the mining code is embedded into a Web page that can be executed via a Web browser (we refer to them as JavaScript miners from now on); (2) the mining code is packed into a binary that can be executed by a device (we will call them binary miners). The Web-based approach to cryptojacking is typically used on malicious websites, while the binary approach is favored by the authors of the traditional computer malware. Importantly, both these approaches can be used within Android apps. We refer the interested reader to our technical report that provides more details about the integration of Web content and native code into Android apps [9].

The miners that explicitly ask the user consent for mining (legitimate) and those that attempt to hide the mining process (illicit) could be created using either of the methods. Besides, there are also scam miners – applications that only pretend to mine cryptocurrencies, but do not actually deliver. JavaScript miners typically rely upon drive-by mining services, e.g., CoinHive, CoinImp or CryptoLoot\textsuperscript{6}, that provide the necessary infrastructure to mine cryptocurrencies such as Monero. This is particularly attractive for regular users, as Monero can be mined using the CPU, instead of the expensive GPU or other specialized hardware [18]. There exist other “lightweight” cryptocurrencies, e.g., Litecoin and Ethereum, that can be mined using commodity hardware. However, according to various reports, Monero dominates them [8, 12, 27]. To facilitate cryptocurrency mining with comparatively weak hardware, mining service providers support creating mining pools, when several devices combine their computational power to perform mining collectively.

Listing 1 shows an example of an initialization script that, when embedded into a Web page, starts mining Monero once the page is loaded. The “SITE_KEY” needs to be substituted by a key value connected to the cryptocurrency wallet that will receive the mining reward. In this example, the anonymous version of the CoinHive API is used: the real wallet public key is proxied through the site keys associated to it, and the “identity” of the wallet behind the miner cannot be inferred. Such a simple mining initialization script is particularly popular in malicious website mining, as it is takes minimal effort to embed it, and provides anonymity [14]. The script in Listing 1 can be invoked via the WebView component that supports loading Web pages and JavaScript code in Android apps. Therefore, Android apps can use the same mechanism for mining crypto as regular websites.

Binary Android miners can load a native mining library via the System.loadLibrary(...) interface. Then the declared native

\textsuperscript{6}https://coinhive.com/ (this service is down since Spring 2019, but it has been active during the early stages of this work), https://www.coinmp.com/, https://crypto-loot.com/
While building our dataset, we performed many iterations of the following steps: (1) find a large sample of potential Android miners using string search, and download them; (2) perform manual analysis to find evidence that these apps are miners and discard false positives; (3) update the search strings used at the step (1) with new patterns discovered at the step (2).

We seeded our dataset by collecting several samples of Android miners using hashes reported in relevant security industry blog posts and white papers, e.g., the SophosLabs report [29]. We have also collected mining apps from different Android stores (Google Play\(^8\), F-droid, etc.). This initial dataset has been used to create a set of strings and rules in the YARA notation\(^9\) indicating mining payload in the code and metadata.

Our initial set of miner-related strings and YARA rules contained only a few generic keywords, e.g., “Monero”, generic mining API calls such as `CoinHive.Anonymous()`, and domain names of the popular mining pools, e.g., “mine.xmrpool.net”. Such strings are useful for finding some potential miners, but they are insufficient.

Automatic string pattern search against many diverse apps is inevitably prone to errors. As we aimed to build a reliable dataset that could be used as the ground truth for detecting Android miners, we manually analyzed each app with at least a single match to the miner-related strings. We were looking for characteristics of the mining activity and the intended user interactions: (a) the mining code (e.g., the code that initializes mining and the mining libraries); (b) how mining can be triggered by a user (by interacting with an app in a certain way or by simply launching its main activity); (c) supported cryptocurrencies; (d) the declared functionality of the app and whether it tries to “hide” its intentions. Additionally, we searched for more string patterns that can be used to extend our dataset. Discovering a new pattern, we added it to our set of miner-related strings, and re-ran the string search against the apps that had no previous matches and a new batch of apps that we were downloading.

As a source of potential miners, we used the popular platforms VirusTotal\(^10\) and Koodous\(^11\). On VirusTotal, we used the Private API\(^12\) to search and download the apps from our original dataset. We also collected the app metadata used and compiled a list of cryptocurrency-related malware families. We used the file search functionality\(^13\) to download all Android apps that have been detected by at least one antivirus engine and belong to at least one malware family from our list. Additionally, we downloaded apps with known miner-related strings (e.g., mining pools and domain names listed in [14, 18]). Koodous allows to create YARA rulesets, search for matching Android apps, and download them. We created our own YARA ruleset for potential Android miners based on the list of the identified miner-related strings. We also searched for the YARA rules to detect miners that had been already written by the community and incorporated them into our ruleset.

Manual analysis. To confirm that an app is a miner we performed a thorough manual analysis. We decompiled each app with apktool, matched the set of miner-related strings against the decompiled files, and examined the app starting from the files where we found matches. We investigated resource files with extensions .html, .js, and .xml, native libraries (.so), and executable files shipped with the app. Once we had located the `mining initialization code` (e.g., a JavaScript code fragment that inserts the mining credentials into a mining library and starts the mining, or a `smali` code fragment that calls a native mining library) and/or the mining code (e.g., a library that implements the mining functionality), we looked for the entry points in the app that triggered the mining. For example, we found at least 22 cases when the mining initialization code is placed directly into the `MainActivity` class, or located in a subclass of the `Application` class – in such cases, the mining starts immediately upon the app startup. In this process, we have identified and collected other static indicators suggesting that the app under analysis is a miner. We describe them in Section 5.1.

Using the string patterns and rules, we have downloaded in total 17159 potential mining apps using both VirusTotal and Koodous. After the manual analysis, we have obtained the dataset of 728 Android miners. During the collection phase, we might have missed miners, e.g., if they used advanced hiding techniques like dynamic code updates [35]. Thus, we admit that our dataset may be incomplete.

However, to the best of our knowledge, currently this is the only publicly available dataset of mobile miners.

### 4 DATASET DESCRIPTION

**JavaScript vs binary miners.** Table 1 reports the numbers and proportions of JavaScript and binary miners in our dataset, and the numbers of illicit miners among them. We also identified a small subset of miner-related apps that are similar to miners, but do not contain any mining code, and can be points of confusion for automatic miner detection approaches (we discuss them in more detail further in this Section). The distribution of apps in Table 1 suggests that the most popular way to create mining apps is with JavaScript. The majority of the miners in our sample are illicit (614 illicit miners).

Only 39 miners from our sample can be characterized as generic malware, i.e., integrating traditional malicious behaviors (confirmed by manual analysis and via VirusTotal). 30 of these miners actively force users to allow them administrative privileges, and monitor if they receive the role of the device administrator, and whether users try to revoke this role.

---

1. \(^1\) https://koodous.com/
2. \(^2\) https://www.virustotal.com/en/documentation/private-api
3. \(^3\) https://www.virustotal.com/intelligence/help/file-search/
4. \(^4\) https://github.com/pooler/cpuminer
5. \(^5\) Our data collection had started before Google decided to remove all mining apps from Google Play on 07/27/2018.
6. \(^6\) http://virustotal.github.io/yara/
7. \(^7\) https://www.virustotal.com/
8. \(^8\) https://koodous.com/
10. \(^10\) https://github.com/pooler/cpuminer
12. \(^12\) https://www.virustotal.com/intelligence/help/file-search/
Wang et al. [33] and Jiang and Zhou [16] that looked into malware—without user consent [36]. Comparing the statistics in the works of Android malware [1, 30], and it is interesting to see whether the illicit
not match it to any of the original libraries (mostly the case for legitimate
the fact that they might be using a custom mining library that we
mining code for the remaining apps from our sample that rely on them. In total, these libraries
identified in our sample. Table 4 lists these libraries and the number
4.1.2 Mining libraries.
connection status.
miners tries to monitor the battery consumption and the network
of malicious apps [39]. We see also that a significant number of
the device has been booted. This system event is highly indicative
Most of the miners have subscribed to the
READ_SMS
malicious samples only, e.g.,
generally do not request permissions that are very prevalent in
evident that the only permission needed for Android miners to
requested Android permissions across our sample of miners. It is
same system events as generic malware. Table 2 lists the top 10
miner population uses the same permissions and listens to the
4.1 Miner Characteristics
4.1.1 Permissions and monitored system events. Permissions and
system events subscriptions are widely used as features to detect Android malware [1, 30], and it is interesting to see whether the miner population uses the same permissions and listens to the same system events as generic malware. Table 2 lists the top 10 requested Android permissions across our sample of miners. It is evident that the only permission needed for Android miners to properly function is the INTERNET. This permission is currently not considered dangerous, and is granted by the Android system without user consent [36]. Comparing the statistics in the works of Wang et al. [33] and Jiang and Zhou [16] that looked into malware-specific permissions with Table 2, we can conclude that miners generally do not request permissions that are very prevalent in malicious samples only, e.g., READ_SMS.
Table 3: Top 10 system event subscriptions by miners

Notably, for JavaScript miners, the plain JavaScript CoinHive API library is not the most used one: 437 miners integrate the CoinHive Android SDK library, which is a wrapper for convenient JavaScript-to-Java bindings to the CoinHive API in Android apps. An example of the CoinHive API usage is shown in Listing 1. We found that the Authedmine API has been used only in 3 cases. We further identified the usage of several desktop cryptomining software projects in binary miners: CPUMiner, CGMiner, XMRig, and MinerD. These projects have been specifically compiled for Android as libraries/executables by the authors of the miners. We found on GitHub several versions of CPUMiner and CGMiner used by the miners.

We observed that in many cases the third-party libraries have been used “as is”, but in some cases the original library was changed by the miner authors. For instance, the original CoinHive Android SDK library has had large modifications in at least 64 illicit miners from our sample. In all these cases, the changes were non-significant to the core functionality of the library (perhaps, made only for evading detection): e.g., package name has been changed, several classes not related to mining have been removed, variable names have been changed, etc. For example, in 8 of these cases, the engine.html file, which is the core of the library, has not been modified (confirmed by hashes of the files against the original file provided by the library). In 56 cases the engine.html file has been renamed into coinhive.html and modified, yet the original mining functionality was intact.

Overall, these observations favor the intuition that, given the small hash rate for smartphone-based mining [27], the malicious actors would not spend resources on implementing the mining functionality from scratch, but rather use the libraries that are already available.

By looking at the used third-party libraries and the code of the miners from our sample we identified that 586 miners targeted the Monero cryptocurrency, 5 miners targeted Ethereum, 5 miners targeted Litecoin, and 3 miners had been created to test the capability of mobile devices for mining Bitcoin. 91 binary miners in our sample rely on third-party mining libraries that can be used to mine multiple cryptocurrencies (Monero, Litecoin, Ethereum, Bitcoin, and others).

4.1.3 Mining campaigns. Similarly to the previous works on in-browser cryptojacking by Konoth et al. [18] and Hong et al. [14],

Table 2: Top 10 Android permissions used by miners (Protection levels: D - dangerous, N - normal)

Table 1: The distribution of mining apps in our dataset

<table>
<thead>
<tr>
<th>Category</th>
<th># samples (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript</td>
<td>594 (77.95%)</td>
</tr>
<tr>
<td>JavaScript illicit</td>
<td>563 (73.88%)</td>
</tr>
<tr>
<td>Binary</td>
<td>134 (17.59%)</td>
</tr>
<tr>
<td>Binary illicit</td>
<td>53 (6.69%)</td>
</tr>
<tr>
<td>Miner-related</td>
<td>34 (4.46%)</td>
</tr>
<tr>
<td>Total</td>
<td>762</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Android permission</th>
<th>Protection level</th>
<th>#Miners (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERNET</td>
<td>N (D if API=23)</td>
<td>728 (100.00%)</td>
</tr>
<tr>
<td>ACCESS_NETWORK_STATE</td>
<td>N</td>
<td>374 (51.37%)</td>
</tr>
<tr>
<td>WAKE_LOCK</td>
<td>N (D if API=17)</td>
<td>351 (48.21%)</td>
</tr>
<tr>
<td>WRITE_EXTERNAL_STORAGE</td>
<td>D</td>
<td>300 (41.21%)</td>
</tr>
<tr>
<td>RECEIVE_BOOT_COMPLETED</td>
<td>N</td>
<td>258 (35.44%)</td>
</tr>
<tr>
<td>READ_EXTERNAL_STORAGE</td>
<td>D (N if API=23)</td>
<td>203 (27.88%)</td>
</tr>
<tr>
<td>c2dm.permission.RECEIVE</td>
<td>N</td>
<td>149 (20.47%)</td>
</tr>
<tr>
<td>ACCESS_WIFI_STATE</td>
<td>N</td>
<td>138 (18.96%)</td>
</tr>
<tr>
<td>VIBRATE</td>
<td>N</td>
<td>135 (18.54%)</td>
</tr>
<tr>
<td>READ_PHONE_STATE</td>
<td>D</td>
<td>128 (17.58%)</td>
</tr>
<tr>
<td>android.intent.action.BATTERY_LOW</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>android.intent.action.QUICKBOOT_POWERON</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>android.intent.action.MY_PACKAGE_REPLACED</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>android.permission.INSTALL_LOCATION</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>android.permission.WAKE_LOCK</td>
<td>N (D if API&lt;17)</td>
<td></td>
</tr>
<tr>
<td>android.permission.ACCESS_NETWORK_STATE</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>android.net.conn.CONNECTIVITY_CHANGE</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>com.htc.intent.action.QUICKBOOT_POWERON</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>android.net.conn.CONNECTIVITY_CHANGE</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>android.intent.action.ACTION_POWER_DISCONNECTED</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>android.intent.action.ACTION_POWER_CONNECTED</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>android.intent.action.BATTERY_LOW</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

4.1.2 Mining libraries. 8 third-party mining libraries have been identified in our sample. Table 4 lists these libraries and the number of apps from our sample that rely on them. In total, these libraries are used by 671 miners. We could not identify the origin of the mining code for the remaining 57 miners. This was either due to the fact that they might be using a custom mining library that we could not identify (mostly the case for legitimate miners), or the library has been heavily changed and obfuscated so that we could not match it to any of the original libraries (mostly the case for illicit miners).
Notably, a security researcher has already spotted our largest mining campaign\(^5\). Our dataset contains more apps than it was originally reported (342 versus 291). Moreover, at least 24 of apps from our sample that belong to this campaign do not share similar code (unlike the apps seen by the researcher), suggesting that the campaign might be even bigger in the wild. In our sample there are other 64 miners that correspond to 17 mining campaigns, for which mining credentials have been reported in white papers and blog posts by other researchers. We have collected 173 miners that correspond to 76 campaigns that have not been previously reported. Particularly, the second largest illicit campaign shown on Figure 1 has not been reported before.

4.2 Examples of Mining Applications

The screens in Figures 2a and 2b demonstrate examples of two mining applications. The first app is an illicit miner\(^6\). It looks like an app that was created just for fun and provides very basic functionality playing a funny song. However, invisibly it mines the Monero cryptocurrency in a hidden Web browser. The second example is a legal miner created specifically for mining Bitcoin on ARM devices\(^7\). Users need to configure their own mining credentials and run the miner. It is a binary miner that exploits a standalone executable minerd. Both apps have been previously hosted on Google Play.

4.3 Miner-related apps.

We have encountered 34 Android apps that we refer to as miner-related. While these apps do not perform any mining, they are riddled with keywords, links, and mining credentials relevant to the real mining apps. Such apps may pose additional challenges for automated miner detection approaches, and it is therefore important to consider their presence in the wild. We include these apps as a separate category in the dataset, because they are valuable confusing data points. Below we briefly describe them. More information about these apps is available in the technical report\[^9\].

12 of these apps have useful functionality: e.g., they either monitor the value of cryptocurrencies, serve as cryptocurrency wallets, or simply ask for donations in cryptocurrencies (for apps of the latter case we found a match for a cryptocurrency wallet). These

\[^5\]\url{https://twitter.com/fs0c131y/status/950082658991802630}

\[^6\]\url{SHA256:a3f376a5c74e1fe112766bad458a6b3976226e2164b106538322adf6bced}

\[^7\]\url{SHA256:727cd892ed4f7432c7f9d8d0e1aad6d2e436c6c2f4772c5ae2ca36c79dc6e}

---

### Table 4: Third-party mining libraries used by the miners from our sample

<table>
<thead>
<tr>
<th>Library</th>
<th>Type</th>
<th>URL</th>
<th>#Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoinHive SDK</td>
<td>Java</td>
<td><a href="https://github.com/theadpach64/coin_hive_andorid_sdk">github.com/theadpach64/coin_hive_andorid_sdk</a></td>
<td>437</td>
</tr>
<tr>
<td>CoinHive API</td>
<td>JS</td>
<td><a href="https://coinhive.com/lib/coinhive.min.js">https://coinhive.com/lib/coinhive.min.js</a></td>
<td>139</td>
</tr>
<tr>
<td>CPUMiner</td>
<td>Binary</td>
<td><a href="https://github.com/pooler/cpuminer-android">https://github.com/pooler/cpuminer</a></td>
<td>42</td>
</tr>
<tr>
<td>MinerD</td>
<td>Binary</td>
<td><a href="https://github.com/Mimbloclockchainproject/Miner">https://github.com/Mimbloclockchainproject/Miner</a></td>
<td>26</td>
</tr>
<tr>
<td>CGMiner</td>
<td>Binary</td>
<td><a href="https://github.com/reorder/cgminer_keccak">https://github.com/reorder/cgminer_keccak</a></td>
<td>17</td>
</tr>
<tr>
<td>XMRig</td>
<td>Binary</td>
<td><a href="https://github.com/Max-Coin/cgminer">https://github.com/Max-Coin/cgminer</a></td>
<td>6</td>
</tr>
<tr>
<td>Authedmine API</td>
<td>JS</td>
<td><a href="https://authedmine.com/media/linger.html">https://authedmine.com/media/linger.html</a></td>
<td>3</td>
</tr>
<tr>
<td>CoinW0nK</td>
<td>JS</td>
<td><a href="https://github.com/0wnw0nK/CoinHive">https://github.com/0wnw0nK/CoinHive</a></td>
<td>1</td>
</tr>
</tbody>
</table>

**UNKNOWN** 57
We checked our dataset using the VirusTotal API. We downloaded while the most recent one was uploaded in March, 2019. The rest of 22 miner-related apps are scam 77 antivirus products 1 out of cases. We aggregated on the platform. Interest-ingly, 16 apps from our list are not detected by any antivirus product. Table 5 reports SHA256 hashes of these miners and additional data extracted from VirusTotal reports. 10 out of these 16 apps are legitimate miners, and 6 of them are illicit. The apps from this table are not new: the oldest dates back to 2013. However, even the illicit miners among these apps are still not recognized as malicious or unwanted. For 4 apps this can be explained by the fact that the mining script is stored in the encrypted form and decrypted at runtime, and 2 of the undetected apps use obfuscation. Based on these results, we cannot not draw firm conclusions whether mining functionality is deemed malicious by the VirusTotal scanners, as legitimate miners are also detected as malware. Our analysis has not revealed evidence of generic malicious functionality for legitimate miners.

We aggregated the samples by months when they have been first spotted on VirusTotal. Figure 3a shows the application submission timeline based on the VirusTotal data. Our dataset is relatively new: the majority of apps have been first spotted on VirusTotal in 2018. Second, the figure shows that before the last quarter of 2017 there were few new miner submissions, while in the beginning of 2018 we observe a huge spike. Before 2017 the interest to the cryptocurrencies was relatively low. However, in 2017 the price of cryptocurrencies started to grow exponentially attracting the attention of criminals. As a result, in the end of 2017 antivirus companies started to consider Android miners as harmful applications [8, 31]. This indicates that miner developers are profit-driven. They quickly adopt the techniques that bring revenue and lose interest when the area is drained, while security companies are mostly reactive.

Figure 3b shows the Cumulative Distribution Function (CDF) representing the amount of antivirus scanners that detected each application from our dataset. On average, a sample in our list is marked as malicious by 22 scanners. Maximum, the miner in our dataset is detected by 43 different scanners. This shows that even old, well-known samples are not recognized by all scanners. In Figure 3b, we can spot three high steps at 3, 10 and 35 detections (63, 57 and 59 new samples correspondingly). These steps could have appeared because some scanners have similar detection engines, or they share antivirus databases. Table 6 proves this hypothesis. It shows that the results of some scanner pairs are either identical or highly correlated.

We have ranked the VirusTotal scanners according to their ability to detect miners. We assigned +1 point to each true positive and -1 point to each false negative; if a scanner failed to scan a sample or VirusTotal does not have the data, we gave 0 points. The final score is calculated as sum of these points. Table 7 reports the Top 10 VirusTotal scanners based on this score. Note that the rating is based only on the illicit miners dataset. More details about the VirusTotal analysis of our dataset are available in the technical report [9].

## 5 DETECTING ANDROID MINERS

### 5.1 Static Indicators

We now describe the heuristics that we identified and used when performing manual analysis of potential miner apps. These heuris-tics can be used as static indicators for pinpointing potential Android miners across large amounts of apps.
5.1.1 Library indicators. As mentioned in Section 4, the vast majority of our miners use third-party mining libraries, and often the code of these libraries is used without any changes. Therefore, detecting these libraries will indicate a potential miner with a high degree of certainty. For libraries written in JavaScript we took note of the distinctive code patterns and strings. For libraries written in Java (e.g., CoinHive Android SDK shown in Table 4) we collected distinctive components of the library, such as package names, classes, and small code patterns. For native libraries and executables we captured their filename and the SHA256 hash value, as well as specific string patterns that can be present inside them. For instance, most of the native mining libraries listed in Table 4 have a distinctive help menu that lists the available mining parameters and settings. When we see an unknown binary file that could be a mining library, we can obtain its hexdump using command line tools such as `xxd` and compare string patterns inside the binary file against the
We observed that many Web 
applications contain JavaScript mining code, which is often stored as a library, such as libcpuminer.so. In some cases, the JavaScript mining code is stored as a wallet (e.g., engine.html) element. The UI element through the JavaScript resources of an app, and is loaded into an Android application. HTML pages can also help when the initialization code is obfuscated to a certain degree. For example, in case of the CoinHive Android SDK library, we looked at the values of the parameters passed either to the CoinHive Java class (Listing 3), or the parameters passed directly into the engine.html file.

We used regular expressions based on the patterns of mining credentials for various cryptocurrencies. As we were looking for strings like NDMtBChB1LiUKeJ1u2KC8mYSQ2My4zYXxh, we computed the Shannon Entropy metric for strings and only kept the strings for which this metric exceeded a certain threshold (we empirically selected the value of 4.33). We checked the retrieved mining credentials and added them to our string search.

### Listing 3: CoinHive Android SDK initialization example

```java
1 public class App extends Application {
2     @Override
3     public void onCreate() {
4         super.onCreate();
5         Coinhive.getInstance();
6         init("YOUR SITE") // mining credentials
7         setNumberOfThreads(4) // CPU threads
8         setThrottle(0.2) // CPU throttle
9     }
10 }
```

string patterns retrieved from known mining libraries. While this approach cannot beat sophisticated obfuscation techniques, it may be still helpful to uncover a large set of miners where the library (or its parts) is used as is. We found that this simple heuristic was quite powerful, allowing us to find many miners that were difficult to spot otherwise.

We illustrate our heuristics with the CoinHive Android SDK library (Table 4). The library implements a convenient Java wrapper around the CoinHive JavaScript API. Thus, the library can be added directly to an Android project as a dependency, and a CoinHive miner instance can be created and launched from within the Java code, as shown in Listing 3. The CoinHive Java class contains JavaScript-to-Java bindings to the file called engine.html located in the resources/ folder of the SDK. This file includes the plain CoinHive API JavaScript library (Table 4). Therefore, JavaScript miners that rely on this library can be detected by searching for code patterns specific for the mining library (e.g., the smali code that corresponds to the miner initialization code shown in Listing 3), and/or for the code patterns of the engine.html file.

The miner initialization code for Web-based mining services, such as CoinHive API, is typically inserted into benign HTML/JavaScript resources of an app, and is loaded into an Android WebView UI element through the WebView.loadUrl(...). This is quite similar to how the browser-based mining works in the Web. In some cases, the JavaScript mining code is stored as a string constant inside the smali code and is passed directly into a WebView element.

The authors of binary miners typically place the mining libraries (e.g., libcpuminer.so) or standalone executables (e.g., the miner ELF executable) under the res/raw/ or the assets/ folders of an app archive. These libraries are invoked either via the Android System.loadLibrary(...-interface, or by spawning separate application processes for executables.

### 5.1.2 Mining credentials

We also looked for mining credentials (e.g., cryptocurrency wallet and site key identifiers) passed into the mining initialization code – the presence of known mining credentials immediately indicates that they are most likely miners. In fact, 577 illicit miners from our sample have hardcoded mining credentials. Therefore, to quickly pinpoint new miners we built a collection of such identifiers.

We used several heuristics to retrieve the mining credentials. We observed that many JavaScript and binary miners share similar miner initialization code patterns. Therefore, after the known third-party library has been located, it is easier to identify the code that initializes the mining and recover the mining credentials (this can also help when the initialization code is obfuscated to a certain degree). For example, in case of the CoinHive Android SDK library, we looked at the values of the parameters passed either to the CoinHive Java class (Listing 3), or the parameters passed directly into the engine.html file.

We used regular expressions based on the patterns of mining credentials for various cryptocurrencies. As we were looking for strings like NDMtBChB1LiUKeJ1u2KC8mYSQ2My4zYXxh, we computed the Shannon Entropy metric for strings and only kept the strings for which this metric exceeded a certain threshold (we empirically selected the value of 4.33). We checked the retrieved mining credentials and added them to our string search.

### 5.1.3 Mining domains and keywords

To search through the apps for which we could not find known mining credentials or code patterns, we used potential mining domains and simple keywords such as miner, bitcoin, stratum, monero, hashrate, etc. While the keyword search allows to find new previously unseen miners, using it alone is prone to large amounts of false-positives. For example, many non-miner apps that we encountered contain an adblock functionality that actively tries to block known cryptocurrency mining domains (and thus, there will be a match to the known miner domain list).

### 5.1.4 Evasion techniques

We observed that some illicit miners applied evasion techniques. The code fragments that initialize the mining process and contain the mining credentials may be not shipped with the miner app itself, or this code could be encrypted within an app to be decrypted at runtime.

For example, we encountered cases when an illicit miner loads the mining credentials from a remote server upon the startup. The download link is present within the code of the miner, but it has been obfuscated. However, when we launched the app in the Android emulator, the logcat utility allowed us to see which link the app is trying to connect to, and that it downloads a JSON file. The file contains the mining credentials. We provide an excerpt of this configuration file in Listing 4. In this Listing, we can see the settings for the mining script, including the (shortened) wallet address, the preferred mining pool, and some configurations that allow to start mining when the device is charging and not charging. By examining the public GitHub repository where the link is hosted, we found several other files that had similar structure but different mining wallets. We added these wallets to our miner-related strings, however we have not found any apps that use them yet. This could be due to some other ways of hiding the mining payload that we are not yet aware of, or possibly the owner of this repository is creating illicit cryptocurrency miners for other application platforms.

We observed another approach for hiding credentials. The application resources contain an .html resource file with a link to the CoinHive mining script, yet, there seem to be no code that initializes the mining process. Upon closer inspection of other links embedded into the HTML pages, we identified a set of links to JavaScript code located at suspicious websites. Several of these links contained the

---

19E.g., file:///android_asset/engine.html?site_key=...
20We compiled a large list of known mining domains from various sources such as https://github.com/hoshsadiq/adblock-nocoin-list/blob/master/nocoin.txt.
21The link is still available at the time of writing: https://raw.githubusercontent.com/cryptominesetting/setting/master/setting.txt.
22https://github.com/cryptominesetting/setting
We developed an approach and a prototype called BrenntDroid. Thus, it is necessary to perform runtime mining detection, not only on the internal properties of the dataset, but also on any obfuscated or embedded code that may be involved.

5.2 Dynamic Detection

We developed an approach and a prototype called BrenntDroid that leverages machine learning for detecting Android miners using dynamic features. To build this prototype, we selected a sample consisting of 200 Android applications: 100 miners and 100 benign apps. For the miners class, we selected 100 apks from our dataset that start mining immediately after they have launched. This is a valid assumption because our tool is supposed to constantly monitor the device and, thus, can detect the moment when an app starts mining, while dormant miners do not cause damage for the user. For the benign class, we randomly selected 100 apps from the local Google Play store among the "Trending", "Top Apps", and "Top Grossing" groups. These apps include various categories such as "Gaming", "Education", "Sports" and "Shopping", and their numbers of downloads range from 100 to more than 500M.

For each of these apps, we collected a set of dynamic parameter values generated by the corresponding application using the Snapdragon Profiler [24]. We exercised each app from our dataset for 300 seconds on LG Nexus 5 powered by Snapdragon 800 running the Lineage OS 14.1 operating system (based on Android 7.1), and collected data about the low-level system events.

We used filtering techniques that perform cleaning based only on the collected data about the low-level system events. For each of the metric timeseries, we calculated 10 simple statistical values: 1) Minimum (Min); 2) Maximum (Max); 3) Average (Mean); 4) Median (Median); 5) Unbiased kurtosis (Kurt); 6) Unbiased skew (Sew); 7) Unbiased standard error of the mean (Sem); 8) Standard deviation (Std); 9) Mean absolute deviation (Mad); 10) Coefficient of variation (CV).

We evaluated our model using the 10-fold stratified cross-validation with all selected features (see Figure 6 for the Receiver Operating Characteristic (ROC) curve). In our experiment, we achieved 95% of accuracy with the Area Under Curve (AUC) score equal to 0.988±0.009. Our model proves that it is possible to build a very accurate runtime detection tool. At the same time, we admit that collection of dynamic features is connected with large power consumption overhead. This means that implementation of BrenntDroid to run on a user device could be impractical. Moreover, the obtained machine learning model is valid in our testbed and may not transferable to other devices. Still, the developed approach could be applied as one of the tests during the application vetting process at an app store like Google Play [37].

5.2.2 Detection results. We evaluated our model using the 10-fold stratified cross-validation with all selected features (see Figure 6 for the Receiver Operating Characteristic (ROC) curve). In our experiment, we achieved 95% of accuracy with the Area Under Curve (AUC) score equal to 0.988±0.009. Our model proves that it is possible to build a very accurate runtime detection tool. At the same time, we admit that collection of dynamic features is connected with large power consumption overhead. This means that implementation of BrenntDroid to run on a user device could be impractical. Moreover, the obtained machine learning model is valid in our testbed and may not transferable to other devices. Still, the developed approach could be applied as one of the tests during the application vetting process at an app store like Google Play [37].

#### Listing 4: Illicit miner configuration served online

```
1 "chEnabled":true,
2 "ma":false,
3 "alternativeMine":false,
4 "secondaryAlternativeMine":false,
5 "chargingOn":true,
6 "chargingOff":true,
7 "screenOff":true,
8 "screenOn":true,
9 "alternativeLink"="http://srymore.go",
10 "secondaryAlternativeLink"="",
11 "nativeMinerPool":"pool.supportxmr.com:3333",
12 "wallet":"44V8ww9soyFfrivJDfcgmT2gXCFPQDyLdsFyS7mEo2xT[...]",
13 "alternativeLink":"http://crymore.ga",
14 "secondaryAlternativeLink":"",
15 "nativeMinerPool":"pool.supportxmr.com:3333",
16 "wallet":"44V8ww9soyFfrivJDfcgmT2gXCFPQDyLdsFyS7mEo2xT[...]",
17 "nativeMinerThread":"",
18 "ch":0.7,
19 "ma":0.8,
```

#### Listing 5: Remote CoinHive initialization script example

```
1 var miner = new CoinHive.Anonymous('..');
2 miner.start();
```

initialization code (shown in Listing 5). Therefore, for improving static Android miner detection, it is important to inspect remote resources.

The majority of scam miners in our dataset are obfuscated, probably, to hinder inspection and to make repackaging more difficult. Thus, it is necessary to perform runtime mining detection, not only in the cases, when the mining code is not shipped within the Android app and/or is heavily obfuscated, but also to identify scam miners that do not mine.
5.3 Discussion

We have shared our approaches to identify mining code, our set of static miner indicators, and we have presented a technique based on dynamic performance-related features for detecting mining apps on Android. These approaches have certain limitations.

Static indicators we collected are relevant to our dataset, and new generations of cryptojacking malware apps will likely include different tokens. Moreover, cryptomining libraries evolve or die, and new cryptocurrencies and novel mining algorithms emerge constantly. Yet, our ideas focus on the Android platform specifics that will not change with the cryptomining evolution. It is likely that CPU mining on Android will continue to exist in the JavaScript and the binary flavors. We have also seen evidence that the low marginal profit from mobile mining drives the attackers to implement the most straightforward and easy solutions. Thus, our suggestions on
manual miner analysis and static indicator collection will likely be valid for some time in the future.

Our dynamic detection approach is based on CPU performance profiling. We have demonstrated its practical viability on our dataset. As cryptomining libraries may and will change in the future, our machine learning-based approach will require retraining. However, our approach itself will be relevant, until the underlying core idea of cryptomining, which involves solving computationally hard puzzles quickly (proof-of-work), will change to other consensus mechanisms.

Our dataset shared with the community contains many apps that rely on the CoinHive library for mining (note that many different mining libraries are also present in our dataset). While CoinHive was shut down in 2019, Monero miners are still a threat\(^{23}\). Moreover, at the time of writing, CoinHive is still the dominant browser-based cryptomining library\(^{24}\). Therefore, it is important to be able to detect apps with this library, and our results and the apps with this library are still relevant for the community.

6 RELATED WORK

To the best of our knowledge, ours is the first work reporting on Android cryptomining applications. Previously, cryptojacking has been investigated in the context of traditional binary malware \([15, 22]\), and there have been several papers focusing on browser-based cryptomining \([12, 18, 20, 25–27, 32]\).

**Browser-based cryptojacking.** Eskandari et al. \([12]\) applied keyword-based search to the website code, and have reported finding more than 30K occurrences of the CoinHive library and some occurrences of its alternatives, such as CryptoLoot and JSECoin. Konoth et al. \([18]\) have found 20 active crypto-mining campaigns and 28 crypto-mining services in Alexa’s Top 1 Million websites. They have used keyword-based search in web traffic logs, followed by manual analysis. Similarly, Musch et al. \([20]\) have applied CPU usage profiling and presence of web assembly code and several WebWorkers as cryptojacking indicators. Hong et al. \([14]\) have proposed a run-time mining detection tool CMTracker integrating hash computation-based and stack structure-based profilers.

Saad et al. \([27]\) used public services to acquire a list of websites with mining code embedded. Using these sites as the ground truth, they have developed dynamic mining script profiles with respect to CPU usage, battery drain and network usage. Machine learning-based approach to browser-based miner detection has also been outlined in Carlin et al. \([3]\), where opcode traces have been used as features, and in Draghicescu et al. \([10]\), where the CPU allocation features and the threads and socket connections have been captured.

**Binary-based cryptojacking.** Malicious Bitcoin cryptominers have been investigated by Huang et al. \([15]\) already in 2014. Division into campaigns and profits generated by recent binary-based cryptominers have been analysed by Pastrana and Suarez-Tangil \([22]\). The data collection approach used in \([22]\) is similar to ours, as the authors crawled public services for malicious samples and then applied static and dynamic analysis heuristics to select only miners.

In contrast to the aforementioned works, ours focuses on mining apps in the Android ecosystem. Our results show that the Android platform is plagued by both Web-based and binary cryptojackers. We collected and analysed not only malicious cryptominers, as in \([22]\), but also bona fide miners. We have also reported about the phenomenon of scam miners that only pretend to be mining cryptocurrencies, while, at best, only serving ads to the users.

**Dynamic mining detection** Recently, Clay et al. \([5]\) have evaluated the CPU consumption required for mining on Android devices. As mentioned, Saad, Khormali and Mohaisen \([27]\) reported on using CPU usage and battery level on several devices, including an Android phone, to discriminate mining web scripts from non-mining ones (that were emulated with JavaScript disabled in the browser).

Our dynamic detection approach relies on evaluation of many dynamically profiled features of Android applications, including CPU usage and battery drain caused by computation-intensive mining code. In contrast to \([3, 27]\), our solution for dynamic miner detection is based on comparison of mining apps with benign but fully functional ones.

A generic, platform-agnostic approach to covert cryptomining detection based on hardware performance counters profiles and machine learning techniques has been proposed by Conti et al. \([7]\). This work addresses creating generic mining profiles composed of processor’s events that are common to the majority of cryptomining algorithms. Our approach uses another set of features for the dynamic detection task. It would be interesting to investigate comparative advantages of each feature sets in an implementation tailored to a mobile platform.

**Android malware detection.** As our analysis of VirusTotal results and security industry reports show \([11]\), mining functionality can be delivered as a part of malicious payload. There exist a large body of work that focuses on Android malware detection, e.g., \([1, 16, 30, 33, 34, 39]\), to name just a few. Several approaches for detecting Android malware based on energy consumption fingerprints have been proposed and evaluated, e.g., \([2, 4, 13]\). Yet, these works focused on detection of malware behaviors other than mining.

7 CONCLUSIONS

Cryptojacking poses a serious threat to mobile devices. In order to better comprehend this threat, we collected a dataset of 728 Android mining apps and dissected them. To the best of our knowledge,
this is the first work that looks into cryptojacking applications on Android. Our analysis confirms that the public knowledge in this area is largely insufficient. For example, we found 173 illicit miners from 76 mining campaigns that have been not previously reported. With the clean dataset available, we executed a set of miners and compared the results with benign applications. We identified a set of dynamic features that contribute the most to accurate classification. Based on our dataset, we achieved 95% of accuracy with the AUC score of 0.98±0.009. Our prototype called BRENNTYDROID can be used to detect miners at runtime.

In the future, we will study illicit cryptominers that use heavy code obfuscation. We also plan to extend BRENNTYDROID with techniques for detecting static and dynamic evasion methods, such as CPU throttling.

Acknowledgements. This research was supported by Luxembourg National Research Fund through grants C15/IS/10404933/COMMA and AFR-PhD-11289380-DroidMod.

REFERENCES


