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:
Stanislav Dashevskyi, Yury Zhauniarovich, Olga Gadyatskaya, Aleksandr Pilgun, and Hamza Ouhssain. 2020. Dissecting Android Cryptocurrency Miners. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy (CODASPY '20), March 16–18, 2020, New Orleans, LA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3374664.3375724

*This research was done while Stanislav was at SnT, University of Luxembourg.
†This research was done while Olga was at SnT, University of Luxembourg.
‡This research was done while Hamza was at University of Luxembourg.

CODASPY '20, March 16–18, 2020, New Orleans, LA, USA

ACM ISBN 978-1-4503-7107-0/20/03…$15.00
https://doi.org/10.1145/3374664.3375724

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 today[1] and that may further proliferate during the next crypto boom. Due to the rising price of the Monero coin, in Summer 2019 the cryptomining malware was revitalized[2].

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

[1]https://www.helpnetsecurity.com/2019/04/10/cryptomining-still-dominates/
[2]https://www.zdnet.com/article/crypto-mining-malware-saw-new-life-over-the-summer-as-monero-value-tripled/

victims. Even smart TV appliances can now be infected with mining Android apps[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[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[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 BRENNTDROID tool that can be used to detect miners dynamically and to check if an app indeed mines cryptocurrencies.

[3]http://blog.netlab.360.com/adb-miner-more-information-en/
[4]https://blog.trendmicro.com/trendlabs-security-intelligence/cryptocurrency-mining-botnet-arrives-through-adb-and-spreads-through-ssh/
[5]The dataset is available upon request at https://standash.github.io/android-miners-dataset/

```
<script src="https://coinhive.com/lib/coinhive.min.js"/>
<script>
  var miner = new CoinHive.Anonymous('SITE_KEY');
  miner.start();
</script>
```

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[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

[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.coinimp.com/, https://crypto-loot.com/

```
./minerd --url stratum+tcp://eu.multipool.us:7777 --userpass USERNAME:PASSWORD
```

Listing 2: Binary miner initialization example

methods are called from the managed code. Listing 2 shows an example of another approach, when a miner binary (a *MinerD*[7] executable) is called from a shell.

3 DATASET COLLECTION

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 set 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 incomprehensive. 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 the 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.

[7] The source code is available at https://github.com/pooler/cpuminer
[8] Our data collection had started before Google decided to remove all mining apps from Google Play on 07/27/2018.
[9] http://virustotal.github.io/yara/
[10] https://www.virustotal.com/
[11] https://koodous.com/
[12] https://www.virustotal.com/en/documentation/private-api
[13] https://www.virustotal.com/intelligence/help/file-search/

Category	# samples (%)
JavaScript	594 (77.95%)
JavaScript illicit	563 (73.88%)
Binary	134 (17.59%)
Binary illicit	51 (6.69%)
Miner-related	34 (4.46%)
Total	762

Table 1: The distribution of mining apps in our dataset

Android permission	Protection level	#Miners (%)
INTERNET	N (D if API<23)	728 (100.00%)
ACCESS_NETWORK_STATE	N	374 (51.37%)
WAKE_LOCK	N (D if API<17)	351 (48.21%)
WRITE_EXTERNAL_STORAGE	D	300 (41.21%)
RECEIVE_BOOT_COMPLETED	N	258 (35.44%)
READ_EXTERNAL_STORAGE	D (N if API<23)	203 (27.88%)
c2dm.permission.RECEIVE	N	149 (20.47%)
ACCESS_WIFI_STATE	N	138 (18.96%)
VIBRATE	N	135 (18.54%)
READ_PHONE_STATE	D	128 (17.58%)

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

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 lists the top 10 most occurring system event subscriptions. Most of the miners have subscribed to the `BOOT_COMPLETED` event, which means that they will attempt to resume their work as soon as the device has been booted. This system event is highly indicative of malicious apps [39]. We see also that a significant number of miners tries to monitor the battery consumption and the network connection status.

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).

Android system event	#Miners (%)
android.intent.action.BOOT_COMPLETED	536 (73.63%)
android.intent.action.QUICKBOOT_POWERON	169 (22.18%)
android.intent.action.MY_PACKAGE_REPLACED	161 (22.11%)
com.android.vending.INSTALL_REFERRER	159 (21.84%)
com.google.android.c2dm.intent.RECEIVE	146 (20.05%)
com.htc.intent.action.QUICKBOOT_POWERON	122 (16.76%)
android.net.conn.CONNECTIVITY_CHANGE	95 (13.05%)
android.intent.action.ACTION_POWER_DISCONNECTED	84 (11.26%)
android.intent.action.ACTION_POWER_CONNECTED	84 (11.26%)
android.intent.action.BATTERY_LOW	68 (8.52%)

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*[14] 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],

[14]This API has been released by CoinHive as well. Unlike the original mining API, it requires explicit user consent for mining.

Library	Type	URL	#Apps
CoinHive Android SDK	Java	https://github.com/theapache64/coin_hive_android_sdk	437
CoinHive API	JS	https://coinhive.com/lib/coinhive.min.js	139
CPUMiner	Binary	https://github.com/pooler/cpuminer https://github.com/mdelling/cpuminer-android	42
MinerD	Binary	https://github.com/MiniblockchainProject/Minerd	26
CGMiner	Binary	https://github.com/reorder/cgminer_keccak https://github.com/Max-Coin/cgminer	17
XMRig	Binary	https://github.com/xmrig/xmrig	6
Authedmine API	JS	https://authedmine.com/media/miner.html	3
C0nw0nk	JS	https://github.com/C0nw0nk/CoinHive	1
UNKNOWN			57

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

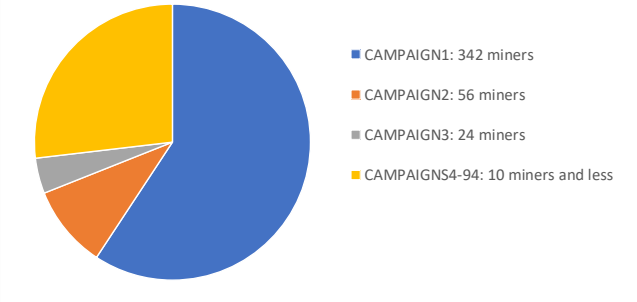


Figure 1: The sizes of mining campaigns

we try to identify the *mining campaigns* by grouping the sets of miners that are likely to belong to the same origin, and therefore may share the profits from the mined cryptocurrency. As most of the miners from our dataset reuse the same third-party mining libraries, it is difficult to identify the same origin by looking at similar code patterns in the apps. Therefore, we assume that two miners belong to the same campaign only if they share the same mining credentials used for authentication with the mining services (e.g., the cryptocurrency *wallet id* and/or CoinHive *site key*).

Figure 1 shows the distribution of the sizes of the mining campaigns found within our sample. Overall, we found **94** unique mining campaigns, with the largest campaign enclosing **342** miners, two smaller campaigns enclosing **56** and **24** miners respectively, and **91** small campaigns of **10** miners and less. The rest of the apps are *benign* miners that did not contain any mining credentials, or *illicit* miners for which we could not retrieve these credentials. Therefore, we could not attribute such miners to a specific mining campaign.

At this stage, we cannot conclude whether the small mining campaigns that we found are indeed small in the wild, as this requires further large-scale data collection and analysis. However, the two relatively large campaigns suggest that in the wild there may be many Android apps created or, more likely, repackaged by the same malicious developer that is actively trying to maximize their mining profit. Indeed, it is relatively inexpensive to repackage an already existing app [38] and add the mining code. We have seen many examples that support this conclusion (see Section 5).

Notably, a security researcher has already spotted our largest mining campaign[15]. 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

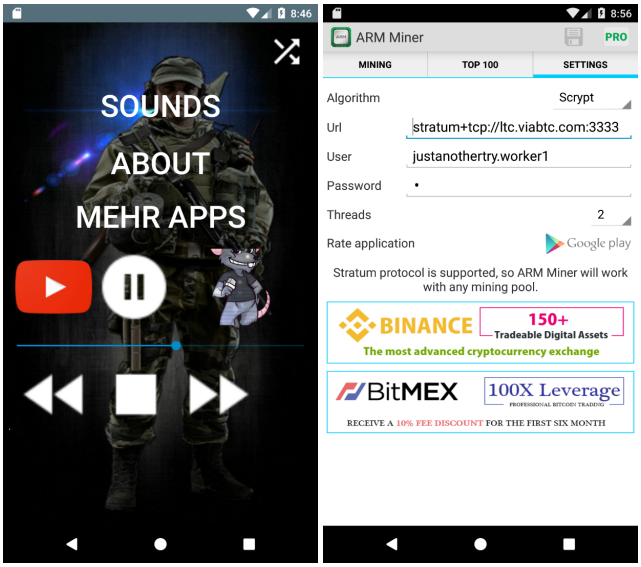
The screens in Figures 2a and 2b demonstrate examples of two mining applications. The first app is an illicit miner[16]. It looks like an app that was created just for fun and provides very basic functionality playing a funny song. However, invisibly it mines the Monero cryptocurrency in a hidden Web browser. The second example is a legal miner created specifically for mining Bitcoin on ARM devices[17]. 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

[15] https://twitter.com/fs0c131y/status/950082654891802630
[16] SHA256 `a3f376a5c74e1fe112786b4ad450a6b3976226e2164b106653483522adf6bced`
[17] SHA256 `727cd092ed478453c2f19d180e1aa8fd22e43dc9cf24772c5ae2ca36cf9dbc4e`



(a) Illicit miner **(b) Legitimate miner**

Figure 2: Screenshots of miner apps

applications can serve as confusion points since various static indicators would suggest the presence of mining code (see Section 5.1). The rest of 22 miner-related apps are *scam*. They do not have any useful functionality, yet claim to be legitimate mining apps. Their monetization comes from either showing paid ads, or tricking their users into paying for an "upgraded" version of the app. For example, the "basic" version may claim that it does not support transferring the mined funds, until a sufficient amount of a cryptocurrency is mined. In particular, 2 of these apps employ a trick to improve their ratings: from the start they promise the user 50,000 Satoshis (0.0005 Bitcoins) "for free" if the user rates the app.

Such apps correspond to another possible source of confusion for automated detection approaches: while an app claims it is a cryptocurrency miner and could be immediately considered as a positive data point (e.g., by classification approaches), it neither contains the mining code, nor manifests the runtime behavior typical to the mining apps (see Section 5.2).

4.4 VirusTotal Analysis Results

We checked our dataset using the VirusTotal API. We downloaded the latest, at the time of writing, extended analysis reports for each app in our dataset. If a report was not found, i.e., a sample had not been uploaded to VirusTotal before, we submitted the app on our own. We were the first who found and uploaded at least **5** samples to VirusTotal[18]. Among previously seen samples, an app from our dataset was checked by VirusTotal at the earliest in October 2013, while the most recent one was uploaded in March, 2019.

All applications from our dataset have been checked by at least 1 out of *77 antivirus products* aggregated on the platform. Interestingly, **16** miners from our dataset 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 **10** 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 VirusToal 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

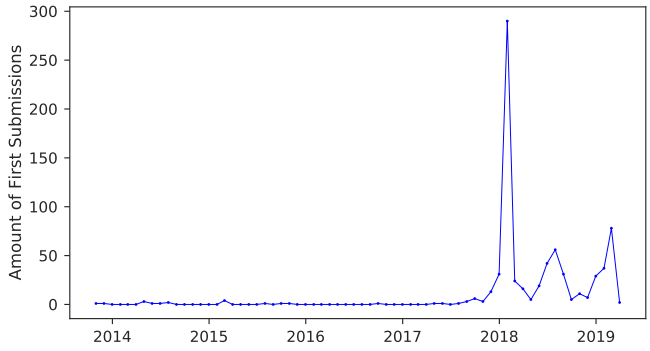
5.1 Static Indicators

We now describe the heuristics that we identified and used when performing manual analysis of potential miner apps. These heuristics can be used as static indicators for pinpointing potential Android miners across large amounts of apps.

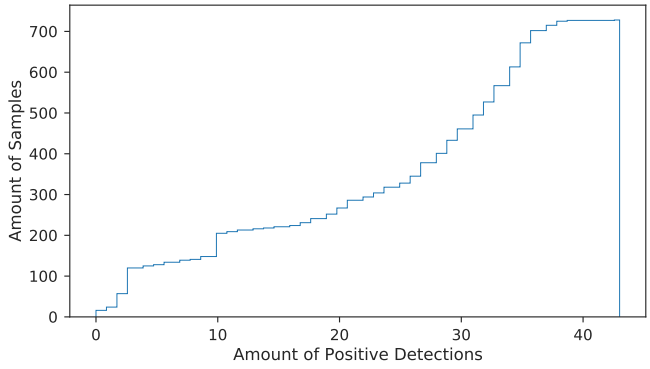
[18]We have not collected this statistics from the start, therefore, we can confirm only 5 cases.

SHA256	Illicit	First seen date	Last seen date	Amount of submissions	Amount of unique sources
aa200375c8422f3e034b122aa45e59a289b6c356b2301c4651189c27a895d9b0	✖	2013-10-13	2015-03-14	4	2
76ae303c82d8233414694ff803c2a22bd82dc1ff1bab1341f9932a238b6b0efc	✖	2017-07-28	2017-07-28	1	1
f8f936810980d14ab41abb91d4fb0bba32c083e6846623d4320ea45053e8ea6d	✖	2017-09-27	2017-09-27	1	1
d735cf3732d00ce43d1a36bc77123770d5d611f09d39b8576e3698a9a2ebda87	✖	2017-12-01	2017-12-01	1	1
c491cbabb604a59c99e5be0e1808f43e1d21b94be524c4d1c759c8bbbd452509	✔	2018-01-09	2018-01-09	1	1
609941fdf62a6f9d186a7714bd4238e0d2c531badd96a69dbb2dce2b4f1d5248	✖	2018-02-28	2018-07-18	8	6
be11f2929b4383f1bcf020c8d7d8b4ef0172c5c5a4e468271ebb87f4b14db876	✖	2018-03-02	2019-04-24	4	3
c471ca1989d7fc7662ea3ba5bf0bcc79d8790fe4770acaaabd10dafadb7ee362	✖	2018-05-03	2018-05-03	2	2
630cf7f1728e8a592aa016171be1f7852f70baaed5267398417f8d91b9d14acb	✖	2018-05-29	2018-09-08	2	2
f61e31ee2f27f2815e7720cad5920b750d10e01788cd78f7fbd81ba1c31dfeb3	✔	2018-07-13	2018-07-13	1	1
7acb35a690d02a34a404cae9ccd3f9b25558e43fd143514c7b42f225aa3663a3	✖	2018-08-05	2018-10-13	2	2
17b56ef3a43c6cd4245113ada9a9ff0364754fc6947d05e9f9acb8e6630f9d27	✖	2018-08-23	2018-08-23	1	1
2e5ba00cc3caa0a4801f2b0580829cee0577e4b05719e86ea7e5690c961d5dae	✔	2019-01-30	2019-01-30	1	1
33db4abf2526b4bda22559e41052ea12362c25e96fd0ebd49becd470694e57de	✔	2019-02-02	2019-04-05	2	2
cb6546a785af3aa2dfee434e25e66ca8691e56909a64e3e317a91fb4e2d5bd1b	✔	2019-02-10	2019-02-10	1	1
ec8433cd5a06aaafe251361ec304dbc438272a5fef49cf7c5c45a63caecff375	✔	2019-03-02	2019-03-02	1	1

Table 5: Samples not detected by the VirusTotal scanners



(a) Date distribution



(b) Positive detections CDF

Figure 3: *VirusTotal* submissions data

Scanner 1	Scanner 2	Correlation
AntiVir	Commtouch	1.000
	ByteHero	1.000
ByteHero	Commtouch	1.000
Agnitum	Commtouch	1.000
Commtouch	Norman	1.000
ByteHero	Norman	1.000
TACHYON	nProtect	1.000
Agnitum	Norman	1.000
	AntiVir	1.000
AntiVir	Norman	1.000
Agnitum	ByteHero	1.000
AVG	Avast	0.997
Kaspersky	ZoneAlarm	0.990
BitDefender	Emsisoft	0.984
	GData	0.927

Table 6: Top 15 highly correlated scanner pairs

Scanner	Final score	True positives	False negatives	Failed / no data
Sophos	514	621	107	0
CAT-QuickHeal	478	603	125	0
DrWeb	474	601	127	0
ESET-NOD32	394	561	167	0
Ikarus	346	512	166	50
Avira	285	505	220	3
McAfee	266	497	231	0
SymantecMobileInsight	256	408	152	168
ZoneAlarm	254	489	235	4
Kaspersky	252	489	237	2

Table 7: Top 10 VirusTotal scanners evaluated on our dataset

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 smali 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

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

Listing 3: CoinHive Android SDK initialization example

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(...)` call – this is quite similar to how the browser-based mining works in the Web [18]. 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 `minerd` 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 code 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[19].

We used regular expressions based on the patterns of mining credentials for various cryptocurrencies. As we were looking for strings like `NDMtBC8iLiUkEjUzKC8mYSQzMy4zYXxh`, we computed the Shannon Entropy metric [19] 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[20] 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[21] 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*[22] 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

[19] E.g., file:///android_asset/engine.html?site_key=....
[20] We compiled a large list of known mining domains from various sources such as https://github.com/hoshsadiq/adblock-nocoin-list/blob/master/nocoin.txt.
[21] The link is still available at the time of writing: https://raw.githubusercontent.com/cryptominesetting/setting/master/setting.txt
[22] https://github.com/cryptominesetting/setting

```
{
  "chEnable":true,
  "maEnable":false,
  "alternativeMine":false,
  "secondaryAlternativeMine":false,
  "chargingOn":true,
  "chargingOff":true,
  "screenOff":true,
  "screenOn":true,

  "ma":0.8,
  "ch":0.7,

  "alternativeLink":"http://crymore.ga",
  "secondaryAlternativeLink":"",

  "nativeMinerPool":"pool.supportxmr.com:3333",
  "wallet":"44V8ww9soyFfrivJDfcgmT2gXCFPQDyLFXyS7mEo2xT[...]",
  "nativeMinerThread":1,
```

Listing 4: Illicit miner configuration served online

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

Listing 5: Remote CoinHive initialization script example

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.

5.2 Dynamic Detection

We developed an approach and a prototype called BrenntDroid that leverage 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 the mining process 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.

5.2.1 Features. For each application, we collected 15 different metrics: 1) Battery Current; 2) Battery Power; 3) CPU Branch Misses; 4) CPU Clock; 5) CPU Context Switches; 6) CPU Cycles; 7) CPU Cycles/Instruction; 8) CPU Instructions; 9) CPU Page Faults; 10) CPU Task Clock; 11) CPU Utilization Percent; 12) Memory Usage; 13) Rx Bytes (Total); 14) Tx Bytes (Total); and 15) Temperature.

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 (Skew); 7) Unbiased standard error of the mean (Sem); 8) Standard deviation (Std); 9) Mean absolute deviation (Mad); 10) Coefficient of variation (CV).

We used filtering techniques that perform cleaning based only on the internal properties of the dataset, without considering our application classes. We applied two feature selection techniques to eliminate excessive variables. First, we removed the features that have low variance in our dataset (threshold=0.1) using the scikit-learn library [23]. Second, we eliminated highly correlated features (Pearson correlation coefficient is more than 0.9). After these procedures, only 67 features remained (see Figure 4 for the list).

To detect the strongest features in our dataset, we exploited the internal property of tree-based algorithms that calculate feature importances as a part of their training procedure. We trained a Random Forest classifier [23] on our dataset. Figure 4 lists the extracted features and shows their importance.

According to our analysis, two most important features are Maximum CPU Utilization % and Average Battery Power. The corresponding Kernel Density Estimation (KDE) plots are shown in Figures 5a and 5b. In Figure 5a, we can spot a huge spike around 100% for miners. This confirms that miners try to utilize all CPU resources on the device. At the same time, we also see some spikes around 30% tick. Thus, some miners in our dataset throttled their mining capability, or used a subset of all available CPU cores. In general, miners' CPU utilization gravitates to discrete values. At the same time, the Maximum CPU Utilization % KDE for benign applications is almost uniformly distributed along the X axis. The more intensive tasks a processor executes, the more power it requires. During this experiment, the phone was attached to a computer through a USB cable. Thus, as a side-effect during the dataset collection the phone was also charging. Figure 5b shows that when a benign application was executed the phone was charging (the extremum value is on the positive side), while the miners were consuming so much energy that the battery was draining, even though the phone was attached to a power supply.

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].

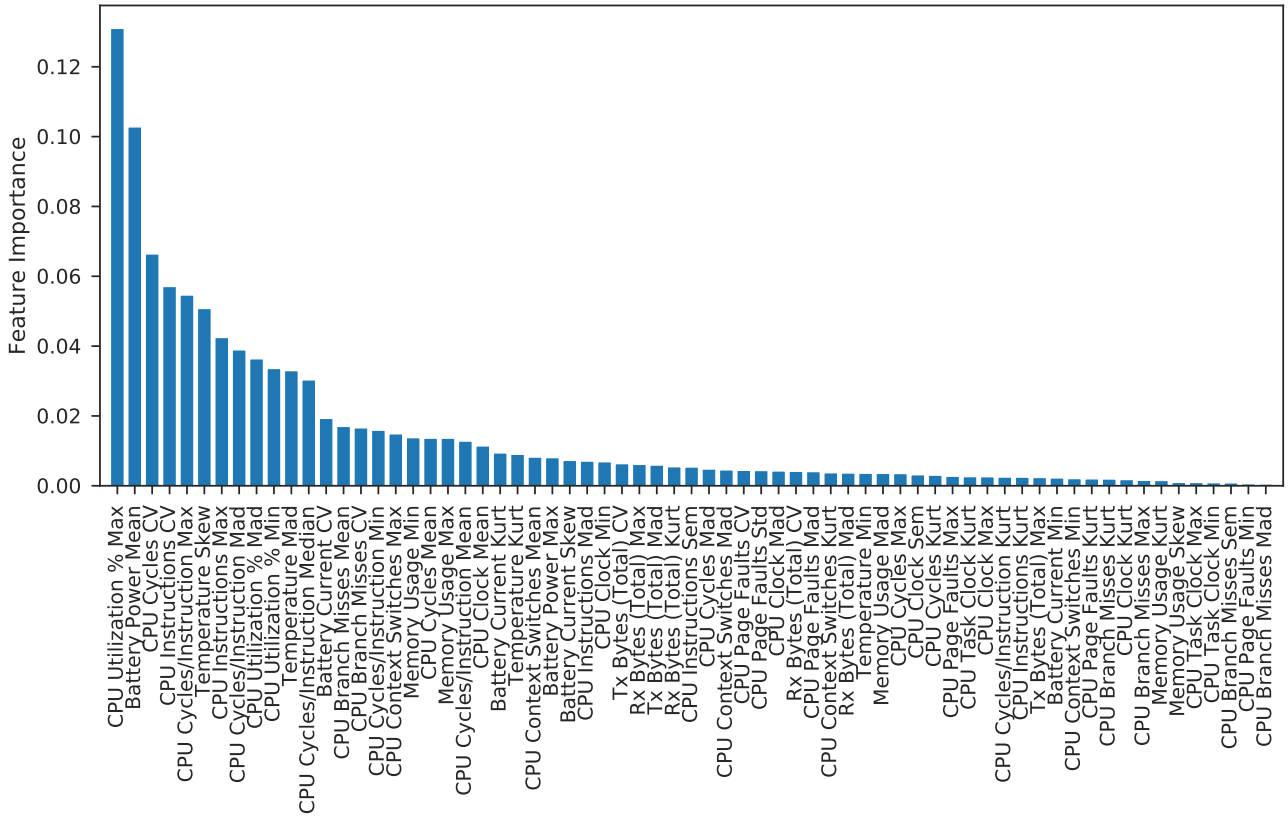
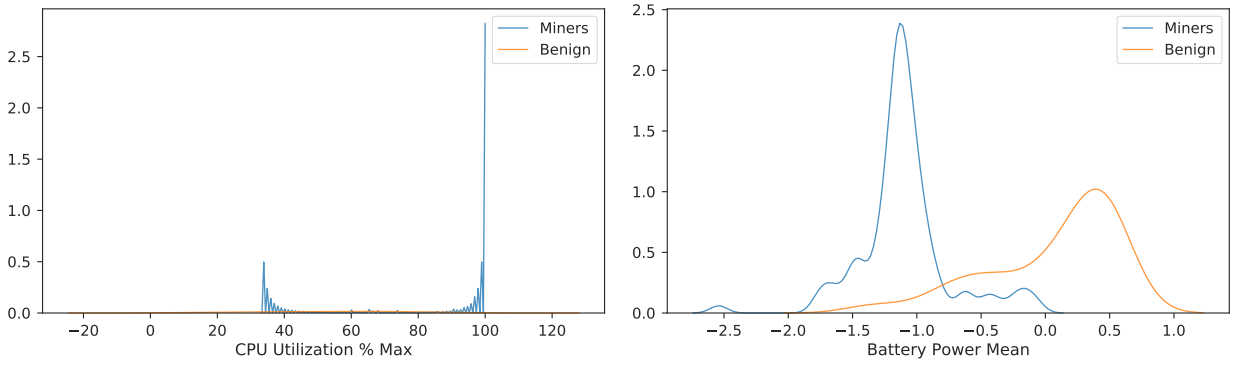


Figure 4: Feature importance



(a) Maximum CPU Utilization % KDE

(b) Battery Power Average KDE

Figure 5: Two most important dynamic features

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

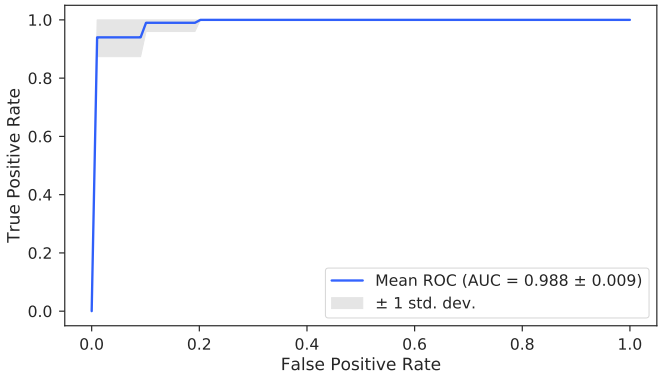


Figure 6: ROC curve

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 cryptojacking [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 analyzed 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,

[23] https://www.bbc.com/news/world-europe-49494927
[24] https://cryptobriefing.com/is-web-mining-still-a-thing/

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 0.988±0.009. Our prototype called BrenntDroid 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 BrenntDroid 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

[1] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *Proceedings of the Network and Distributed System Security Symposium*. 23–26.

[2] Gerardo Canfora, Eric Medvet, Francesco Mercaldo, and Corrado Aaron Visaggio. 2016. Acquiring and Analyzing App Metrics for Effective Mobile Malware Detection. In *Proceedings of the ACM International Workshop on Security And Privacy Analytics*. 50–57.

[3] Domhnall Carlin, Philip O'Kane, Sakir Sezer, and Jonah Burgess. 2018. Detecting Cryptomining Using Dynamic Analysis. In *Proceedings of the Annual Conference on Privacy, Security and Trust*. 1–6.

[4] Luca Caviglione, Mauro Gaggero, Jean-François Lalande, Wojciech Mazurczyk, and Marcin Urbański. 2016. Seeing the Unseen: Revealing Mobile Malware Hidden Communications via Energy Consumption and Artificial Intelligence. *IEEE Transactions on Information Forensics and Security* 11, 4 (2016), 799–810.

[5] James Clay, Alexander Hargrave, and Ramalingam Sridhar. 2018. A Power Analysis of Cryptocurrency Mining: A Mobile Device Perspective. In *Proceedings of the Annual Conference on Privacy, Security and Trust*. 1–5.

[6] Coinhive. 2019. Discontinuation of Coinhive. https://coinhive.com/blog/en/discontinuation-of-coinhive

[7] Mauro Conti, Ankit Gangwal, Gianluca Lain, and Samuele Giuliano Piazzetta. 2019. Detecting Covert Cryptomining using HPC. arXiv:1909.00268

[8] Cyber Threat Alliance. 2018. The Illicit Cryptocurrency Mining Threat. https://www.cyberthreatalliance.org/wp-content/uploads/2018/09/CTA-Illicit-CryptoMining-Whitepaper.pdf

[9] Stanislav Dashevskyi, Yury Zhauniarovich, Olga Gadyatskaya, Aleksandr Pilgun, and Hamza Ouhssain. 2019. Dissecting Android Cryptocurrency Miners. (2019). arXiv:1905.02602

[10] Dragos Draghicescu, Alexandru Caranica, Alexandru Vulpe, and Octavian Fratu. 2018. Crypto-Mining Application Fingerprinting Method. In *Proceedings of the International Conference on Communications*. 543–546.

[11] Randi Eitzman, Kimberly Goody, Bryon Wolcott, and Jeremy Kennelly. 2018. How the Rise of Cryptocurrencies Is Shaping the Cyber Crime Landscape: The Growth of Miners. https://www.fireeye.com/blog/threat-research/2018/07/cryptocurrencies-cyber-crime-growth-of-miners.html

[12] Shayan Eskandari, Andreas Leoutsarakos, Troy Mursch, and Jeremy Clark. 2018. A First Look at Browser-based Cryptojacking. arXiv:1803.02887

[13] Xing Gao, Dachuan Liu, Daiping Liu, and Haining Wang. 2016. On Energy Security of Smartphones. In *Proceedings of the ACM Conference on Data and Application Security and Privacy*. 148–150.

[14] Geng Hong, Zhemin Yang, Sen Yang, Lei Zhang, Yuhong Nan, Zhibo Zhang, Min Yang, Yuan Zhang, Zhiyun Qian, and Haixin Duan. 2018. How You Get Shot in the Back: A Systematical Study about Cryptojacking in the Real World. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 1701–1713.

[15] Danny Yuxing Huang, Hitesh Dharmdasani, Sarah Meiklejohn, Vacha Dave, Chris Grier, Damon McCoy, Stefan Savage, Nicholas Weaver, Alex C Snoeren, and Kirill Levchenko. 2014. Botcoin: Monetizing Stolen Cycles. In *Proceedings of the Network and Distributed System Security Symposium*.

[16] Xuxian Jiang and Yajin Zhou. 2012. Dissecting Android Malware: Characterization and Evolution. In *Proceedings of the IEEE Symposium on Security and Privacy*.

[17] Kaspersky. 2017. Loapi – This Trojan is Hot! https://www.kaspersky.com/blog/loapi-trojan/20510/

[18] Radhesh Krishnan Konoth, Emanuele Vineti, Veelasha Moonsamy, Martina Lindorfer, Christopher Kruegel, Herbert Bos, and Giovanni Vigna. 2018. MineSweeper: An In-depth Look into Drive-by Cryptocurrency Mining and Its Defense. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 1714–1730.

[19] Wanli Ma, John Campbell, Dat Tran, and Dale Kleeman. 2010. Password Entropy and Password Quality. In *Proceedings of the International Conference on Network and System Security*.

[20] Marius Musch, Christian Wressnegger, Martin Johns, and Konrad Rieck. 2018. Web-based Cryptojacking in the Wild. (2018). arXiv:1808.09474

[21] Panagiotis Papadopoulos, Panagiotis Ilia, and Evangelos P. Markatos. 2018. Truth in Web Mining: Measuring the Profitability and Cost of Cryptominers as a Web Monetization Model. (2018). arXiv:1806.01994

[22] Sergio Pastrana and Guillermo Suarez-Tangil. 2019. A First Look at the Crypto-Mining Malware Ecosystem: A Decade of Unrestricted Wealth. In *Proceedings of the Internet Measurement Conference*. 73–86.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[24] Qualcomm Technologies, Inc. 2019. Snapdragon Profiler. https://developer.qualcomm.com/software/snapdragon-profiler

[25] Julian Rauchberger, Sebastian Schrittwieser, Tobias Dam, Robert Luh, Damjan Buhov, Gerhard Pötzelsberger, and Hyoungshick Kim. 2018. The Other Side of the Coin: A Framework for Detecting and Analyzing Web-Based Cryptocurrency Mining Campaigns. In *Proceedings of the International Conference on Availability, Reliability and Security*. Article 18.

[26] Jan Rüth, Torsten Zimmermann, Konrad Wolsing, and Oliver Hohlfeld. 2018. Digging into Browser-based Crypto Mining. In *Proceedings of the Internet Measurement Conference*.

[27] Muhammad Saad, Aminollah Khormali, and Aziz Mohaisen. 2018. End-to-End Analysis of In-Browser Cryptojacking. (2018). arXiv:1809.02152

[28] Aleieldin Salem, F. Franziska Paulus, and Alexander Pretschner. 2018. Repackman: A Tool for Automatic Repackaging of Android Apps. In *Proceedings of the International Workshop on Advances in Mobile App Analysis*. 25–28.

[29] Sophos Labs. 2018. CoinMiner and Other Malicious Cryptominers Targeting Android. https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/sophos-coinminer-and-other-malicious-cryptominers-tpna.pdf

[30] Kimberly Tam, Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, and Lorenzo Cavallaro. 2017. The Evolution of Android Malware and Android Analysis Techniques. *Comput. Surveys* 49, 4 (Jan. 2017).

[31] Liam Tung. 2017. Android Security: Coin Miners Show up in Apps and Sites to Wear out your CPU. https://www.zdnet.com/article/android-security-coin-miners-show-up-in-apps-and-sites-to-wear-out-your-cpu/

[32] Wenhao Wang, Benjamin Ferrell, Xiaoyang Xu, Kevin W Hamlen, and Shuang Hao. 2018. SEISMIC: SEcure In-lined Script Monitors for Interrupting Cryptojacks. In *Proceedings of the European Symposium on Research in Computer Security*. 122–142.

[33] Wei Wang, Xing Wang, Dawei Feng, Jiqiang Liu, Zhen Han, and Xiangliang Zhang. 2014. Exploring Permission-induced Risk in Android Applications for Malicious Application Detection. *IEEE Transactions on Information Forensics and Security* 9, 11 (2014), 1869–1882.

[34] Lifan Xu, Dongping Zhang, Nuwan Jayasena, and John Cavazos. 2016. HADM: Hybrid Analysis for Detection of Malware. In *Proceedings of the SAI Intelligent Systems Conference*.

[35] Yury Zhauniarovich, Maqsood Ahmad, Olga Gadyatskaya, Bruno Crispo, and Fabio Massacci. 2015. StaDynA: Addressing the Problem of Dynamic Code Updates in the Security Analysis of Android Applications. In *Proceedings of the ACM Conference on Data and Application Security and Privacy*. 37–48.

[36] Yury Zhauniarovich and Olga Gadyatskaya. 2016. Small Changes, Big Changes: An Updated View on the Android Permission System. In *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses*. 346–367.

[37] Yury Zhauniarovich, Olga Gadyatskaya, and Bruno Crispo. 2013. DEMO: Enabling Trusted Stores for Android. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 1345–1348.

[38] Yury Zhauniarovich, Olga Gadyatskaya, Bruno Crispo, Francesco La Spina, and Ermanno Moser. 2014. FSquaDRA: Fast Detection of Repackaged Applications. In *Proceedings of the IFIP Annual Conference on Data and Applications Security and Privacy*. 130–145.

[39] Ziyun Zhu and Tudor Dumitraundefined. 2016. FeatureSmith: Automatically Engineering Features for Malware Detection by Mining the Security Literature. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 767–778.