DISSECTING ANDROID CRYPTOCURRENCY MINERS

A PREPRINT

Stanislav Dashevskyi
University of Luxembourg

Yury Zhauniarovich
Perfect Equanimity

Olga Gadyatskaya
University of Luxembourg

Aleksandr Pilgun
University of Luxembourg

Hamza Ouhssain
University of Luxembourg

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. We analyzed the collected miners and identified 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. We have also analyzed our dataset with 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 tool dubbed BRENNTDROID is able to detect miners with 95% of accuracy on our dataset.

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 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 a quick growth of the mining malware. In 2017, the skyrocketing price of cryptocurrencies caused by the enormous attention to these technologies has played a huge role in the cryptojacking proliferation [10]. Not surprisingly, miners have quickly gained popularity and appeared among the top security threats in 2018 [8, 34]. Security researchers have also paid attention, with many papers focusing on browser-based mining [16, 11, 24, 30, 20, 31, 25, 29] and binary mining [26] appearing in the last year.

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 coins on the mobile device. Yet, as the smartphone-based mining no longer generates interesting profits for the benign user [31, 6], the interest to these apps has significantly diminished. Now Google has removed mining applications 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 most popular service Coinhive has announced discontinuation of its service [7]. Still, there are many alternatives, like CryptoLoot and JSEcoint, which are important cyber threats today and that may further proliferate during the next crypto boom[1,2].

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 number of potential victims. Even smart TV appliances

[1] https://blog.checkpoint.com/2019/04/09/march-2019s-most-wanted-malware-cryptomining-still-dominates-despite-coinhive-closure/

[2] https://blog.malwarebytes.com/cybercrime/2019/05/cryptojacking-in-the-post-coinhive-era/

can now be infected with mining Android apps[3]. Recent security industry reports mention that mining capabilities are being introduced to existing malware families or added into repackaged Android applications [8, 34, 10].

Indeed, mobile mining has certain advantages for the attackers. The mining code packaged as an application (app for short) can be *more persistent* than website visits, by running in the background and when the user is not present. The cost of creating and distributing a miner is *negligible*, given the ease of application repackaging [32] on Android and the availability of mining libraries [20]. 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, what can render the device unusable. For example, a malicious family called `Loapi` causes the mobile devices to explode [34]. Therefore, there is an urgent need to study the Android miner phenomenon and to be able to detect such applications.

Contributions. To the best of our knowledge, our paper is the first systematic 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 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[4].
- 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** mining applications.
- 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 a tool called BRENNTDROID that can be used to detect miners dynamically and to check if an app indeed mines cryptocurrencies (to combat scam).

2 Background

2.1 Android Applications

Android apps are distributed as archive files (`.apk`) that consist of the compiled Java code (`.dex`), the application manifest (`AndroidManifest.xml`) containing the requested Android permissions and subscriptions to system events, application resource files (e.g., Web-based resources such as `.js` and `.html` files), and native libraries (`.so` files).

The contents of `.apk` archives can be extracted with *apktool* [40] that, among other things, disassembles the `.dex` files and transforms the disassembled Java classes into *smali* [33] – a low-level human-readable code representation for Java code in Android platform.

Android has wide support for embedding Web-based content into its applications [21]. To achieve this, Android apps support `WebView` – a technology that packages the basic functionality of web browsers (e.g., webpage rendering and JavaScript) into a Java class that can be instantiated within an Android app and function similarly to a basic web browser. In particular, this technology allows the developers of Android apps to embed Web content into the application resources and interact with this content from the Java code and vice-versa.

```
1 WebView view = new WebView(this);
2 view.getSettings().setJavaScriptEnabled(true);
3 view.loadUrl("https://google.com");
4 view.loadUrl("file://assets/page.html");
```

Listing 1: WebView example

[3] `http://blog.netlab.360.com/adb-miner-more-information-en/`

[4] The dataset is available upon request at `https://standash.github.io/android-miners-dataset/`.

The Android `WebView` is a subclass of the standard `View` class; its basic usage is shown in Listing 1: line 1 instantiates an object of the `WebView` class, line 2 enables the execution of JavaScript code (disabled by default), and lines 3 and 4 load Web-based content into the `WebView` element. Note that not only the remote Web content can be loaded (line 3), but also the resource files shipped together with the app (line 4).

```
1 public class NativeLibrary {
2    static {
3       System.loadLibrary("native_lib");
4    }
5    public static native void doSmthng(int param);
6 }
```

Listing 2: Native library load example

```
1 String command = "echo 'hello world'";
2 Runtime localRuntime = Runtime.getRuntime();
3 localRuntime.exec(command);
```

Listing 3: Running a shell command from an Android app

Listing 2 shows how a native library can be called via the `System.loadLibrary(...)` interface: a Java wrapper class has to be created in the Android app, where the `.so` library has to be loaded (line 3) and the corresponding native methods declared (line 5). Listing 3 illustrates how a shell command can be executed from the Java code of an Android app.

2.2 Android Cryptocurrency Miners

There exist two different 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).

Both of the two mining approaches can be used to create either *legitimate* or *illicit* miners: the miners that belong to the former category explicitly ask the user consent for mining, while the latter attempt to hide the fact that they are mining. Also, the Web-based approach is typically used by the authors of malicious websites, while the binary approach is favored by the authors of the traditional computer malware. It is important to stress, that both of these approaches can be used within Android apps.

Software that mines cryptocurrencies typically rely upon drive-by mining services such as *CoinHive*[5] 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 expensive GPU or other specialized hardware [20]. There exist other "lightweight" cryptocurrencies, such as Litecoin and Ethereum that can be mined using the commodity hardware. However, according to various reports, Monero significantly dominates them [8, 31, 11]. Also, 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. Therefore, when multiple devices are combined into a single mining pool, mining the lightweight cryptocurrencies becomes profitable.

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

Listing 4: JavaScript miner initalization example

[5] https://krebsonsecurity.com/2018/03/who-and-what-is-coinhive/

Listing 4 shows an example of an initialization script that, when embedded into a Web page, starts mining the Monero cryptocurrency once that page is loaded. The "SITE_KEY" needs to be substituted by the actual hash of the public site key, which is connected to a certain cryptocurrency wallet tat willh receive the mining reward (the *mining credentials*). In this example, the *anonymous* version of the CoinHive API has been used: any device that loads the script will perform the mining, however the reward will be received only by the owner of the site key. Moreover, a wallet can have multiple public site keys associated to it, and the "identity" of the wallet behind the miner cannot be inferred. Such a simple mining initalization script is particularly popular in *malicious* website mining, as it is takes no effort for embedding it, and provides anonymity [16].

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

Listing 5: Binary miner initalization example

Similarly, the script on Listing 4 can be invoked via the *WebView* element 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.

Listing 5 shows an example of invoking a binary miner from a shell (a *MinerD*[6] executable). Android apps can also include such miner executables and call them in this fashion by spawning a separate application process.

2.3 Our Terminology

To summarize, in this paper, we distinguish the following categories of cryptocurrency miners:

- *Illicit miners* are applications that try to perform *stealth* mining, i.e. they do not warn users explicitly about the mining process, or *selfish* mining, i.e., they mine cryptocurrencies explicitly, but redirect the profits to the attacker, not the user.
- *Legitimate* or *benign* miners are applications that both declare their mining activities and the user is the sole benefactor of the mining process.
- *Miner-related* apps are applications that match many mining string patterns, but do not perform cryptocurrency mining (e.g., wallet apps).
- *Scam* miners are applications that pretend to perform mining activity, but do not actually perform any mining.
- *Javascript* miners are applications that include the web-based mining payload.
- *Binary* miners are applications that include binary mining payload.

3 Dataset Collection

We started by collecting several samples of Android miners. These applications have been found based on relevant security industry blog posts and whitepapers, e.g., the SophosLabs report [34], that had explicitly mentioned the hashes of illicit Android miners. We have also collected several miners from different Android app stores (Google Play[7], F-droid, etc.). This initial dataset has been used to create a set of strings and rules in the YARA notation[8] indicating mining payload in the code and metadata.

The initial set of miner-related strings and YARA rules contained only a few generic keywords, such as "Monero", "Litecoin", generic mining API calls such as CoinHive.Anonymous(), CRLT.Anonymous(), and CoinHive.User(), and domain names of the popular mining pools such as "us.litecoinpool.org" "mine.xmrpool.net". Yet, while such strings are already useful for finding *some* potential miners, they are insufficient.

During already the first iterations of the dataset collection, we realized that a fully automatic search against many diverse apps is inevitably prone to errors. This was unacceptable, as we aimed to build a reliable collection of apps that could be used as the ground truth for detecting Android miners. Therefore, 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

[6]The source code is available at https://github.com/pooler/cpuminer

[7]This research had started before Google decided to remove all mining apps from Google Play on $07/27/2018$.

[8]http://virustotal.github.io/yara/

intended interactions with user: (a) the mining code (e.g., the code that initializes mining and the mining libraries); (b) how the 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. We also aimed to find more string patterns that can be used to extend our sample of miners. Once we had found 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 have been downloading.

To summarize, 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 a manual analysis to find the 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 repeated these steps multiple times, increasing our dataset of Android miners and improving its quality.

3.1 Main Application Sources

To find a large set of *potential* miners, we used the popular platforms *VirusTotal*[9] and *Koodous*[10]. Services provided by these platforms are quite different, but they both allow to search for Android apps that match specific criteria, and to download them. Below, we briefly introduce these platforms and describe how we utilized them.

VirusTotal checks user-submitted binaries (including Android apps) against several popular anti-virus engines. Currently, *VirusTotal* allows not only to perform scans in the black-box manner, but also to search over the dynamic data of apps (e.g., the URLs that apps try to connect to), and to perform string pattern search over its application database.

We first used the *Private API*[11] of *VirusTotal* to search and download the apps from our original dataset by their hashes, and to collect their metadata. From this metadata we extracted the information about the malware families that these apps potentially correspond to, and manually compiled a list of the families related to cryptocurrency mining (as reported by the anti-virus engines used by *VirusTotal*). Next, we used the *file search functionality*[12] of *VirusTotal*, and downloaded all Android apps that have been recently detected by at least one of the anti-virus engines and belong to at least one malware family from our list. Additionally, we checked the dynamic information of apps against a set of known miner-related strings (e.g., mining pools and domain names listed in [20, 16]), and downloaded the matching apps as well.

Koodous is collaborative platform that allows to download Android apps, analyze them, and share the analysis results. *Koodous* performs static and dynamic analyses of apps using the state-of-art Android analysis tools like Androguard[13] and the Cuckoo sandbox[14]. Users can also write custom YARA rules[15] for finding and downloading the matching apps. We used this functionality to create our own YARA rulesets for searching and downloading potential Android miners from *Koodous*, based on the set of miner-related strings we identified during our manual analysis over the sample retrieved from *VirusTotal* (see Section 3.2). We also searched for the YARA rules that had been already written by the community to detect miner apps, and incorporated them into our ruleset as well.

3.2 Manual Analysis

To confirm that an app is a cryptocurrency miner we performed its thorough manual analysis. During the manual analysis we treated every app as follows: we decompiled it 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 paid special attention to the 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. While performing this process, we have identified and collected other static indicators that suggest that the app under question is a miner. We describe them in more detail in Section 5.1.

[9] `https://www.virustotal.com/`

[10] `https://koodous.com/`

[11] `https://www.virustotal.com/en/documentation/private-api`

[12] `https://www.virustotal.com/intelligence/help/file-search/`

[13] `https://github.com/androguard/androguard`

[14] `https://cuckoosandbox.org/`

[15] `https://yara.readthedocs.io/en/v3.4.0/writingrules.html`

Using the above static indicators (see Section 5.1 for more details), we have downloaded in total 17159 Android apps using both *VirusTotal* and *Koodous*. After the manual analysis step, we obtained the dataset of **728** Android miners (we describe the dataset in Section 4).

4 Dataset Description

Javascript vs binary miners. Table 1 lists the percentages of Android apps in our dataset: the proportions of *javascript* and *binary* miners in the sample, 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, and that the majority of the miners in our sample are *illicit* (**614** illicit miners).

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 Android apps in our sample

Android permission	#Miners (%)
android.permission.INTERNET	728 (100.00%)
android.permission.ACCESS_NETWORK_STATE	374 (51.37%)
android.permission.WAKE_LOCK	351 (48.21%)
android.permission.WRITE_EXTERNAL_STORAGE	300 (41.21%)
android.permission.RECEIVE_BOOT_COMPLETED	258 (35.44%)
android.permission.READ_EXTERNAL_STORAGE	203 (27.88%)
com.google.android.c2dm.permission.RECEIVE	149 (20.47%)
android.permission.ACCESS_WIFI_STATE	138 (18.96%)
android.permission.VIBRATE	135 (18.54%)
android.permission.READ_PHONE_STATE	128 (17.58%)

Table 2: Top 10 Android permissions used by 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

Top Android permissions and system events used. Permissions and system events subscriptions are widely used as features to detect Android malware [2, 36], and it is interesting to see whether the miner population uses the same permissions and listens to the same system events as generic malware.

Wang et al. [39], Jiang and Zhou [19], and Feldman et al. [12] have previously compared Android malware and benign apps in terms of requested permissions. 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 `android.permission.INTERNET`, which is the only permission requested by **286** miners. This permission is not considered dangerous anymore, and is granted by the Android system without user consent [43]. Comparing the statistics in the works of Wang et al. [39] and Jiang and Zhou[19] 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, else called filtered intents. Most of the miners have subscribed to `android.intent.action.BOOT_COMPLETED`, 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 [45]. We see also that a significant number of miners tries to monitor the battery consumption and the network connection status.

Only **39** miners from our sample can be characterized as generic malware with respect to their behavior, while **30** of these miners are actively forcing users to allow them administrative privileges, and monitor whether they receive the role of the device administrator, and whether users try to revoke this role.

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

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

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

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, for *javascript* miners, the plain JavaScript *CoinHive API*[16] library is *not* the most used one: **437** miners integrate the *CoinHive Android SDK* library, which is a wrapper that provides convenient JavaScript-to-Java bindings for the CoinHive API in Android apps. At the same time, we found that the *Authedmine*[17] API has been used only in **3** cases. We 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 several versions of *CPUMiner* and *CGMiner* available on GitHub[18] used by the miners.

We observed that in many cases the third-party libraries have been used "as is", however in some cases the original library is changed by the authors of miners. 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 been totally unchanged (we checked the 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 [31], 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 were able to determine that **586** miners target the Monero cryptocurrency, **5** miners target Ethereum, **5** miners target Litecoin, and **3** miners have 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).

Mining campaigns. Similarly to previous works Konoth et al. [20] and Hong et al. [16], we tried to identify the *mining campaigns* by grouping the sets of miners that are likely to share the same origin, and therefore may share the benefits from the mined cryptocurrency. As most of the miners from our sample reuse the same third-party mining libraries, it is difficult to identify the same origin by looking at the 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 consider such miners to be a part of a mining campaign.

[16] An example of its usage is shown in Listing 4

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

[18] `https://github.com`

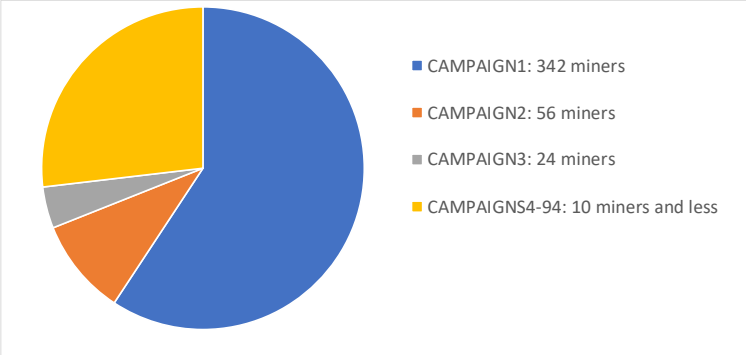


Figure 1: The sizes of mining campaigns

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 her mining profit. Indeed, it is relatively inexpensive to repackage an already existing app and insert only the mining code. We have seen many examples that support this conclusion, see Section 5.

When searching for the mining credentials used in the biggest mining campaign that we identified, we found that a security researcher has already reported[19] this mining campaign. Still, our sample 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 another **64** miners that correspond to **17** mining campaigns which mining credentials have been reported in whitepapers and blogposts by other researchers. Yet, we have collected **173** miners that correspond to **76** campaigns that have not been previously reported. In particular, the second largest *illicit* campaign shown on Figure 1 is has not been reported before.

Miner-related apps. We have also encountered **34** Android apps that we refer to as *miner-related*. While these apps do not perform any cryptocurrency 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 we have built, because they are valuable confusing data points. Below we briefly describe them.

12 of these apps have useful functionality: e.g., they either monitor the value of cryptocurrencies, or 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 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, and 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 should be immediately considered as a positive data point (e.g., by classification approaches from the area of machine learning), they neither contain the mining code, nor manifest the runtime behavior typical to the mining apps (see Section 5.2).

Table 5 summarizes top 10 permissions requested by miner-related applications, and Table 6 shows top 10 system events that miner-related apps subscribe to. The requested permissions and monitored system events in these apps largely coincide with the statistics reported for the mining sample in Tables 2&3. This further shows that lightweight, keyword-based detection approaches for miners may produce many false-positives. At the same time, permissions and system events may indicate that miner-related apps, especially of the *scam* type, are supplied by malicious actors. Yet, our sample of miner-related applications in the dataset is relatively small; thus a larger investigation of a larger population of such applications is warranted.

[19] `https://twitter.com/fs0c131y/status/950082654891802630`

Android permission	# Apps (%)
android.permission.INTERNET	34 (100.00%)
android.permission.ACCESS_NETWORK_STATE	34 (100.00%)
android.permission.WAKE_LOCK	22 (64.70%)
com.google.android.c2dm.permission.RECEIVE	19 (55.88%)
android.permission.ACCESS_WIFI_STATE	15 (44.11%)
android.permission.WRITE_EXTERNAL_STORAGE	13 (38.23%)
android.permission.CAMERA	12 (35.29%)
android.permission.RECEIVE_BOOT_COMPLETED	11 (32.35%)
android.permission.VIBRATE	9 (26.47%)
android.permission.ACCESS_COARSE_LOCATION	8 (23.52%)

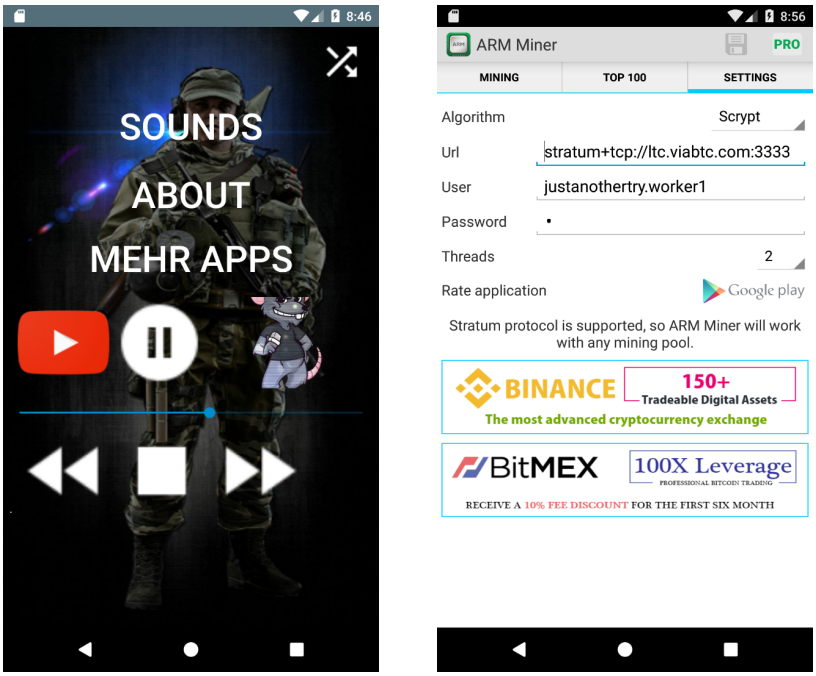
Table 5: Top 10 permissions in miner-related apps

Android system event	# Apps (%)
com.android.vending.INSTALL_REFERRER	19 (55.88%)
android.intent.action.BOOT_COMPLETED	16 (47.05%)
com.google.android.c2dm.intent.RECEIVE	16 (47.05%)
android.net.conn.CONNECTIVITY_CHANGE	7 (20.58%)
android.intent.action.QUICKBOOT_POWERON	5 (14.70%)
android.intent.action.PACKAGE_ADDED	5 (14.70%)
android.intent.action.ACTION_POWER_CONNECTED	4 (11.76%)
android.intent.action.ACTION_POWER_DISCONNECTED	4 (11.76%)
android.intent.action.MY_PACKAGE_REPLACED	4 (11.76%)
android.net.wifi.WIFI_STATE_CHANGED	4 (11.76%)

Table 6: Top 10 system events used by miner-related apps

4.1 Examples of Mining Applications

The screens in Figures 2a and 2b demonstrate examples of two mining applications. The first app is an illicit miner[20]. 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[21]. In this miner 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.



(a) Illicit miner (b) Legal miner

Figure 2: Screenshots of miner apps

4.2 VirusTotal Analysis Results

We checked each sample from our mining dataset using the VirusTotal service. Using their API, we downloaded the VirusTotal extended analysis reports for each app in our dataset, obtaining the latest report version for the time of writing. If a report was not found, i.e., a sample had not been uploaded to VirusTotal before, we submitted the application on our own.

[20] SHA256 `a3f376a5c74e1fe112786b4ad450a6b3976226e2164b106653483522adf6bced`
[21] SHA256 `727cd092ed478453c2f19d180e1aa8fd22e43dc9cf24772c5ae2ca36cf9dbc4e`

We were the first who found and uploaded at least 5 samples to VirusTotal[22]. 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. Figure 3 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 scanner*. Maximum, a sample in our dataset is detected by *43 different scanners*. This shows that even old, well-known samples are not recognized by all scanners.

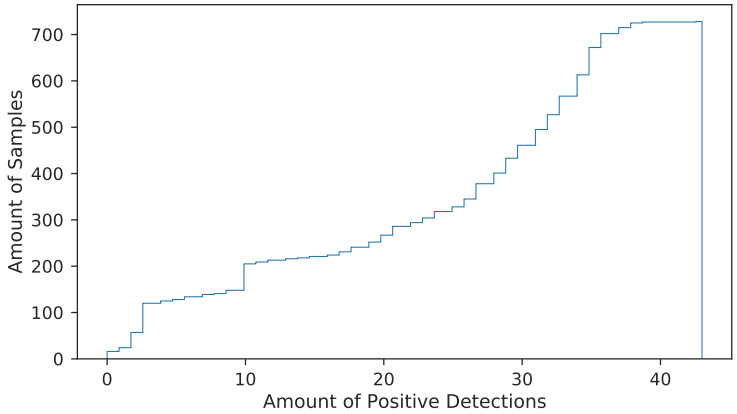


Figure 3: CDF of positive detection by VirusTotal scanners

In Figure 3, 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. Indeed, Table 7 proves this assumption. It shows that the results of some scanner pairs are either identical or highly correlated.

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
Emsisoft	GData	0.922
BitDefender	MAX	0.894
Emsisoft	MAX	0.886
GData	MAX	0.862
Arcabit	BitDefender	0.857

Table 7: Top 20 highly correlated scanner pairs

Interestingly, **16** miners are not detected by any antivirus product. Table 8 reports SHA256 hashes of these miners and the accompanied data that we extracted from VirusTotal reports. We determined **10** out of these 16 apps as *legitimate* miners, and **6** of them as *illicit*. The miners from this table are not new: the oldest is dated 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 is being decrypted at runtime, and **2**

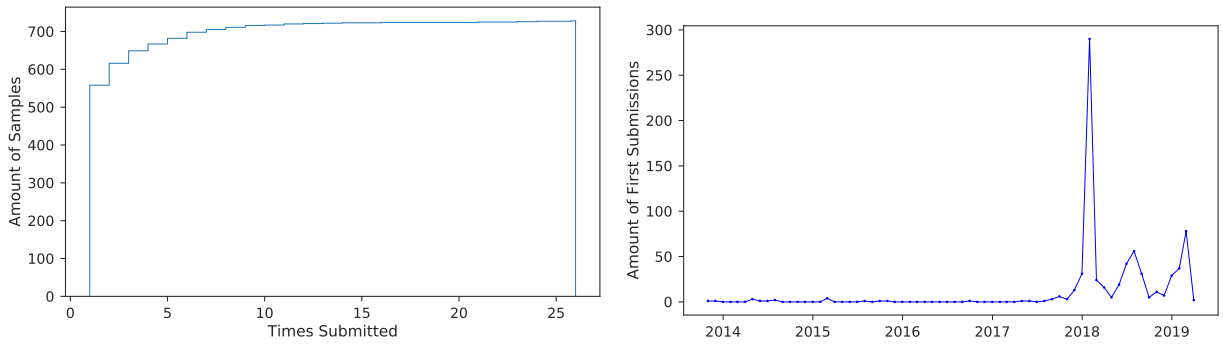
[22]We have not collected this statistics from the start, therefore, we can confirm only 5 cases.

of the undetected apps use obfuscation. Based on these results, we cannot not draw firm conclusions whether mining functionality is deemed malicious by VirusTotal scanners, as **10** legitimate miners are also detected as malware. It could be also that our miners contain some other malicious payloads, even though our analysis have not revealed such evidence for legitimate miners (but there were "also-malicious" illicit miners).

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 8: Samples not detected by the VirusTotal scanners

Figure 4a shows how many times a sample from our dataset has been submitted to VirusTotal. On average, this value is around *1.85*. Indeed, 558 samples have been uploaded to VirusTotal only once, while the most frequently submitted sample has been uploaded 26 times.



(a) CDF of sample submissions

(b) Distribution of new submissions to VirusTotal

Figure 4: Sample submissions to *VirusTotal*

Using the information on the date when a sample was submitted to VirusTotal first time, we evaluated if our dataset is relatively new. To achieve this goal, we aggregated the samples by months when they have been first spotted on VirusTotal. Figure 4b shows the timeline when the apps from our list were submitted to VirusTotal for the first time. Several interesting observations can be derived from this figure. First, we can see that our dataset is relatively new: the majority of applications have been first spotted on VirusTotal in 2018. Second, the figure shows that before the last quarter of 2017 there were a few submissions of new miners, while in the beginning of 2018 we observe huge spike. This phenomenon can be explained by cryptocurrencies popularity in general. Indeed, before 2017 the interest to the cryptocurrencies was mostly driven by niche experts and geeks. However, in 2017 the price of Bitcoin and other cryptocurrencies started to grow exponentially. This attracted the attention of malware developers, who started to explore this market. Clearly, a miner is a very attractive type of malicious application because it directly earns money for the developer, while almost no efforts need to be spent on its preparation and distribution through repackaging. Not surprisingly, in the end of 2017 antivirus companies started to consider Android miners as harmful applications [37, 8].

We have also ranked the VirusToal scanners according to their ability to detect mining applications on our the dataset of manually confirmed miners and their detection results. 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 9 reports the top 10 VirusTotal scanners based on this score.

Note that Table 9 shows the rating based only on our dataset consisting on the samples from one class (miners). This could introduce a deviation in our ranking because a scanner that marks all submitted files as malicious would take the first place in our ranking. To make a more fair list, we would need to get also the list of benign applications and test the scanners on them. However, a comprehensive evaluation of the VirusTotal scanners is out of our scope for this work.

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 9: Top 10 VirusTotal scanners evaluated on our dataset

5 Detecting Android Miners

5.1 Static Indicators

In this Section we describe the heuristics that we identified and used when performing manual analysis of potential miner apps (Section 3). These heuristics can be used as static indicators for pinpointing potential Android miners across large amounts of apps.

Static heuristics. As we describe 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, finding the presence of the code of these libraries will indicate a potential miner with high degree of certainty. For libraries written in JavaScript we take note of the distinctive code patterns and strings. For libraries written in Java (e.g., *CoinHive Android SDK* shown in Table 4) we take note of distinctive components of the library such as package names, classes, and smali code patterns. For native libraries and executables we take note of their filename and SHA256 hash code, as well as specific string patterns that can be present inside them. For instance, most of the native mining libraries listed in Table 4 had 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 string patterns retrieved from known mining libraries. While such 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 find that this simple heuristic is quite powerful, allowing us to find many miners that were difficult to spot otherwise.

We illustrate this heuristic 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 6. 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 relatively easily detected by searching for known code patterns specific for the mining library (e.g., the smali code that corresponds to the miner initialization code shown in Listing 6), and/or for the code patterns present in 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 [20] (the Android-specific code looks similar to the example we made on Listing 1). 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., "`minerd`" ELF executable) under the "`res/raw/`" or the "`assets/`" folder of an app archive. These libraries are invoked either via the Android "`System.loadLibrary(...)`" interface, or by spawning separate application processes for executables (the Android-specific code looks similar to the examples we made in Listing 2 and 3).

```
public class App extends Application {
    @Override
    public void onCreate() {
        super.onCreate();

        CoinHive.getInstance()
                .init("YOUR-SITE-KEY") // mining credentials
                .setNumberOfThreads(4) // CPU threads
                .setThrottle(0.2)      // CPU throttle
    }
}
```

Listing 6: CoinHive Android SDK initialization example

We also looked for the mining credentials (e.g., cryptocurrency *wallet* and *site key* identifiers) passed into the mining initialization code – the presence of known mining credentials in apps immediately indicates that they are most likely miners. In general, we observed that *illicit* miners contain the mining credentials somewhere in the app code (**577** *illicit* miners from our sample have hardcoded mining credentials). Therefore, to significantly reduce the effort of quickly pinpointing 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 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 6), or the parameters passed directly into the "`engine.html`" file[23]. We also used regular expressions based on the patterns of mining credentials for various cryptocurrencies. However, these regular expressions yielded too many irrelevant strings: for instance, we often found strings like "*provideSHealthSyncedWorkoutsDAO*", while we have been searching for strings like "*NDMtBC8iLiUkEjUzKC8mYSQzMy4zYXxh*". Therefore, we calculated the Shannon Entropy metric [22] for such 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, and found many more mining apps using this heuristic.

Finally, to search through the apps for which we could not easily find known mining credentials or code patterns, we used potential mining domains[24] and simple keywords such as `miner`, `bitcoin`, `stratum`, `monero`, `hashrate`, etc. While the keyword search allows to find new previously unseen miners (which helped us a lot at the initial stages of our work), 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 our miner domain list).

Evasion techniques. We found cases when *illicit* miners apply various evasion techniques and their combination to avoid detection: 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 can be encrypted within an app and only be decrypted at runtime.

For example, we found cases when an *illicit* miner loads the mining credentials from a remote server upon the application 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. Upon further inspection of the file[25], it became clear that it contained the mining credentials. We provide a screenshot of this file in Figure 5. In this figure, we can see the settings for the mining script, including the 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**[26] where the link is hosted, we found several other files that had similar structure but different mining wallets. We added these wallets

[23] E.g., "`file:///android_asset/engine.html?site_key=...`"

[24] 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`.

[25] The link is still available at the time of writing: `https://raw.githubusercontent.com/cryptominesetting/setting/master/setting.txt`

[26] `https://github.com/cryptominesetting/setting`

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 (e.g., Google Chrome extensions).


```
{ "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":"44V8ww9soyFfrivJDfcgmT2gXCFPQDyLFXyS7mEo2xTSaf7NFXAL9usGxrko3aKauBGcwZaF1duCWc2p9eDNt9H7Q8iB7gy",
  "nativeMinerThread":1,

  "versionConfig":{
    "7":{
      "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":"",

      "wallet":"44V8ww9soyFfrivJDfcgmT2gXCFPQDyLFXyS7mEo2xTSaf7NFXAL9usGxrko3aKauBGcwZaF1duCWc2p9eDNt9H7Q8iB7gy",
      "nativeMinerPool":"pool.supportxmr.com:3333",
      "nativeMinerThread":1
    },
    "8":{
      "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":"",

      "wallet":"44V8ww9soyFfrivJDfcgmT2gXCFPQDyLFXyS7mEo2xTSaf7NFXAL9usGxrko3aKauBGcwZaF1duCWc2p9eDNt9H7Q8iB7gy",
      "nativeMinerPool":"pool.supportxmr.com:3333",
      "nativeMinerThread":1
    }
  }
}
```


Figure 5: Illicit miner configuration served online

Another approach for hiding miner credentials that we observed is as follows. The application resources contain an `.html` resource file with the link to the CoinHive mining script, yet, there seem to be no code that initializes the mining process. Initially we thought that such apps had no mining capabilities. Yet, upon closer inspection of other links embedded into the html pages, we identified a set of links to some JavaScript code located at suspicious websites. Several of these links[27] contained the code we have been looking for (shown in Listing 7). Therefore, for improving the results of static Android miner detection, it is important to download and inspect the remote resources, such as external links.

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

Listing 7: Remote CoinHive initialization script example

[27] The link is still available at the time of writing: `https://api.kanke365.com/ads/app-tongyong-wk-7.js`

We also found an interesting case when an *illicit* miner consists of heavily obfuscated code (thus we initially flagged it only as suspicious after a keyword match). The app actively tries to obtain the administrative permissions from its users, and contains an encrypted file "`assets/5a240bed02ae6`". Upon thorough inspection of the app, we we found the decryption key and were able to decrypt the file: we realized that the file contains a miner initialization code which is being decrypted at runtime and dynamically called at via the Dalvik classloader (the main reason why the app needs the admin privileges).

We have also found that 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. To achieve dynamic detection, we have developed an approach described in the next section.

5.2 Dynamic Detection

One of the most effective approaches to detect miners is to observe their dynamic behavior. Indeed, in order to gain maximum profit for the developers, a miner should use all available resources [31]. At the same time, in order to persist on the device, an illicit miner should conceal its mining activity, e.g., by applying throttling or doing this when the user does not use the phone.

In this section, we propose an approach and a prototype called BRENNTDROID that leverage machine learning for detecting the Android miners using dynamic features. In order to build this prototype we selected a dataset consisting of 200 Android applications: 100 miners and 100 benign apps.

For the miners dataset, we selected 100 apks from our sample that start the mining process immediately after they have launched. This is a valid assumption because our tool is supposed to constantly monitor applications on a device and, thus, can detect the moment when an app starts mining. Moreover, dormant miners do not cause damage for the user.

As the benign dataset, we randomly selected 100 apps from the local Google Play store among the "Trending", "Top Apps", and "Top Grossing" application groups. These apps include various categories such as "Gaming", "Education", "Sports" and "Shopping", and their number of downloads ranged from 100 to more than 500M.

For each of these apps, we collected a set of traces that contain different dynamic parameter values generated by the corresponding application. We used the *Snapdragon Profiler* [28] to collect these traces. This is a tool developed by Qualcomm Technologies to profile execution of an Android app by collecting CPU, GPU, DSP, memory, power, thermal, and network data in order to find and fix performance issues. We ran each application from our dataset on LG Nexus 5 powered by the Snapdragon 800 system-on-chip running the Lineage OS 14.1 operating system (based on Android 7.1), and collected data about the low-level system events. Each application was exercised for 300 seconds.

In a nutshell, each low-level system event is represented by 4 values: Process name, Timestamp, Metric name, and Metric Value. We consider as a *metric timeseries* a sequence of the values with the corresponding timestamps that share the same application and profiled metric. 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 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). Thus, for every application we obtained a feature vector consisting of 150 values.

This amount of features is large, considering the size of our dataset. Therefore, we applied two feature selection techniques to eliminate excessive variables. It should be mentioned that we used the filtering techniques that perform cleaning only based on the internal properties of the dataset, without considering its connection to our application classes. First, we removed the features that have low variance in our dataset (threshold=0.1) using the scikit-learn library [27]. This operation removed 20 features from our dataset. Second, we eliminated highly correlated features (Pearson correlation coefficient is more than 0.9). After this procedure, only 67 features were left to be used further (see Figure 6 for the list).

To detect the strongest features in our dataset[28], 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 [27] on our dataset. Figure 6 lists the extracted features and shows their importance. The first two positions in this figure occupy

[28]This information can be further used to collect only a subset of strong features on a device.

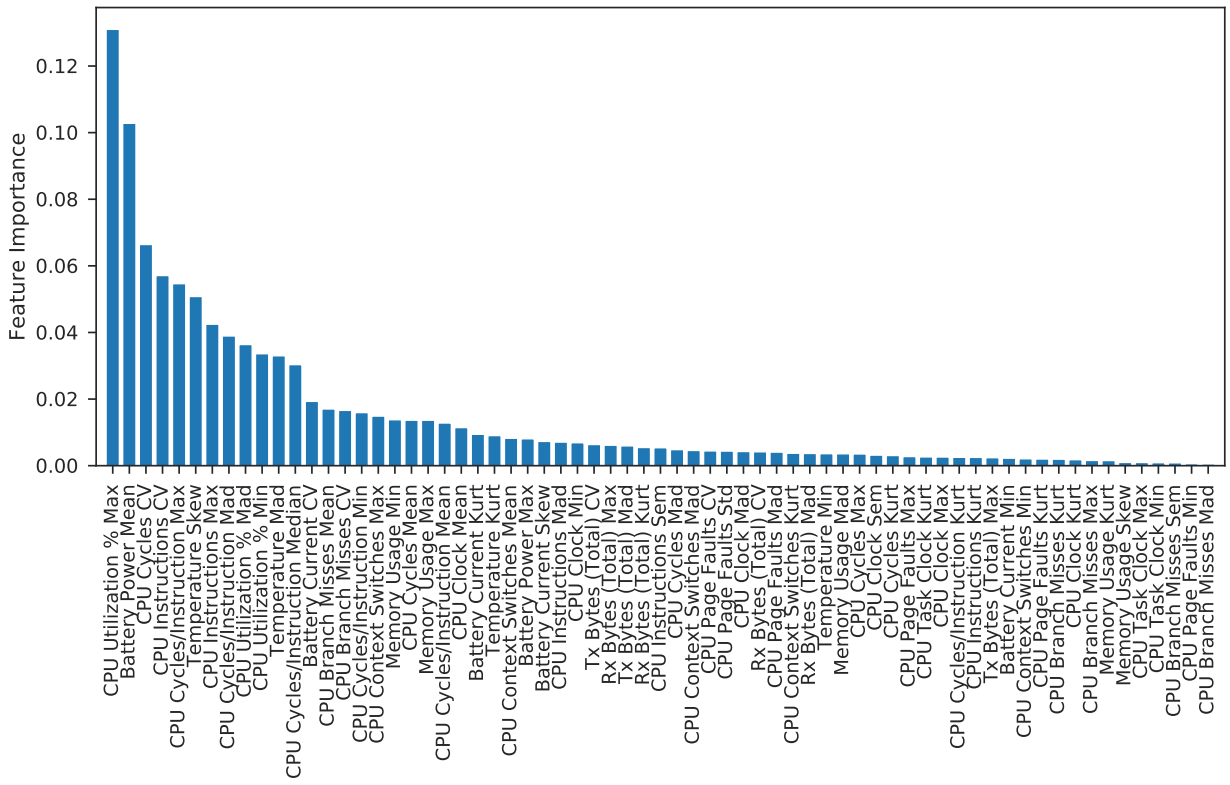


Figure 6: Feature importance

the Maximum CPU Utilization % and Average Battery Power features. The corresponding Kernel Density Estimation (KDE) plots[29] are shown in Figures 7a and 7b.

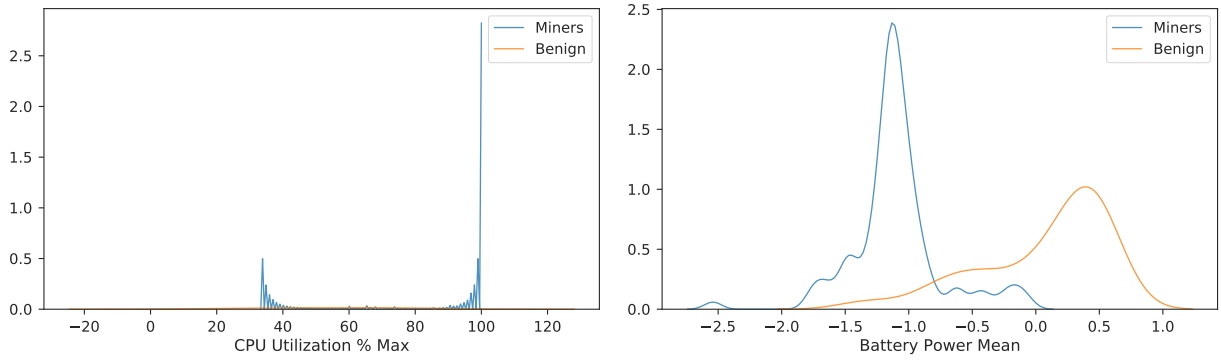
Several observations can be taken from these graphs. First, in Figure 7a, a huge spike around 100% could be observed for miners. This confirms that miners try to utilize all CPU resources on the device. At the same time, we also see some spikes around 30% tick. This proves that some miners in our dataset throttled their mining capability, or used a subset of all available CPU cores.

Second, the Maximum CPU Utilization % KDE for benign applications is almost uniformly distributed along the X axis. That means that there is no specific pattern of CPU utilization by benign apps. I.e., different applications consume on average different amount of CPU resources. The more intensive tasks a processor executes, the more power it requires. During our experiment, the phone collecting the dataset was attached to the computer through a USB cable. As a side-effect, during the dataset collection the phone was also charging. It can be seen on Figure 7b that when a benign application is executed the phone was actually charging (the extremum value is on the positive side). At the same time, the miners were consuming so much energy that the battery was even draining, even though the phone was attached to a source of energy.

We evaluated our model using the 10-fold stratified cross validation applying the Random Forest classifier [27], using all our selected features. Figure 8 shows the Receiver Operating Characteristic (ROC) curve – the dependency between False Positive (FPR) and True Positive (TPR) Rates. The graph confirms that even with simple statistical dynamic features, it is possible to detect mining activity with high confidence. Indeed, in our experiment we managed to achieve 95% of accuracy with the Area Under Curve (AUC) score equal to 0.988 ± 0.009. Our prototype model proves that it is possible to build a very accurate detection tool working at runtime that is able to detect and block mining activities on a device.

We admit that collection of dynamic features is connected with large power consumption overheads. 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 be not transferable to any device. However, the developed approach

[29] We omit the rest of the KDE plots due to the space limitations.



(a) Maximum CPU Utilization % KDE

(b) Battery Power Average KDE

Figure 7: An example of dynamic features

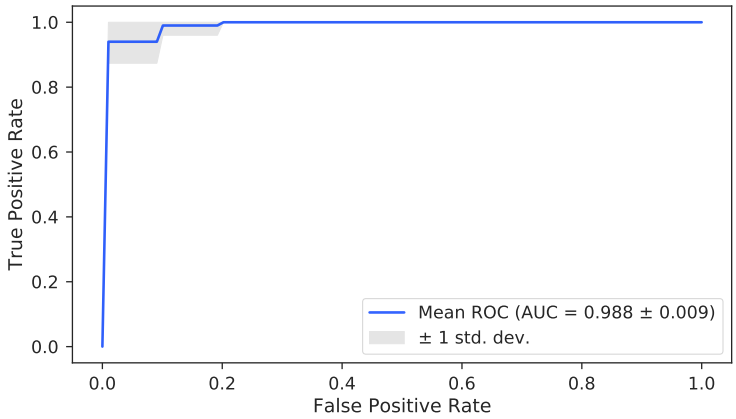


Figure 8: Receiver operating characteristic curve

could be applied during application vetting process [44]. In this case, app stores could inspect an application and use BRENNTDROID as one of the tests. Indeed, in controlled environment an app could be tested on a particular device, which a priori has a trained model.

6 Threats to Validity

In this paper we have presented our findings from a large sample of Android cryptominers and proposed an approach to dynamically detect mining (or verify its absence). These results could be subject to several threats to validity.

The *internal validity* of the presented results depends upon our interpretation of the collected data and the analysis we performed. The main source of this threat in our case may be posed by the heuristics we employed to detect miners among the general population of apps that we have downloaded. To mitigate this threat, we manually checked every app from the resulting miner sample looking for the evidence of the mining code. We have also ran each app to collect more evidence that would support the fact that it is indeed mining.

The *external validity* of our results may be affected by how well they generalize to the entire population of Android cryptocurrency miners in the wild. We acknowledge that our current sample is skewed towards one big campaign, and that we have not processed all available Android applications (which is infeasible) looking for the mining code. It would be interesting to validate our findings working with a large security company that has already privately collected many mining samples in its networks.

We are aware that there exist three main approaches to hide the presence of mining code and hinder detection: CPU throttling, payload hiding and mining script obfuscation [16, 10]. We could have failed to identify certain cryptocurrency miners among the apps that we have been downloading from *Koodous* and *VirusTotal* due to complex obfuscation and evasion techniques employed by the authors of miners, bugs in the *apktool*, or the lack of a major popularity of

miners present at these services. Yet, our main goal was not to capture every miner that is available in the wild, but to obtain and to study a significantly large sample of real-world Android cryptocurrency miners and to pave possible directions for detecting them.

7 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 [26, 18], and there have been several papers focusing on browser-based cryptojacking [20, 24, 30, 11, 31, 29].

7.1 Cryptojacking in Other Contexts.

Browser-based cryptojacking. The ease of integration of Coinhive-like services into websites has led to the proliferation of drive-by cryptomining attacks. Eskandari et al. [11] have applied keyword-based search to the website code on the PublicWWW database, and have reported finding more than 30K occurrences of the Coinhive library and some occurrences of its alternatives, such as Crypto-Loot and JSECoin. Konoth et al. [20] 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. Like in our approach, their keywords included mining services' names and specific strings pertinent to these services (in the miner initialization code and in the Wasm/asm.js, i.e., web assembly, mining payload), Stratum protocol keywords, WebSocket communication. They have also used a high number of WebWorker threads in a web site as a feature pertinent to mining. Similarly, Musch et al. [24] report that 0.25% websites from Alexa Top 1 Million are serving crypto-mining code. They have applied CPU usage profiling and presence of web assembly code and several WebWorkers as indicators. Hong et al. [16] have proposed a run-time mining detection tool CMTracker that integrates hash computation-based and stack structure-based profilers.

Ruth et al. [30] have seeded their mining website dataset from the NoCoin list [17] and have proposed a fingerprinting technique for Wasm code. Rauchberger et al. [29] have proposed the MiningHunter technique to detect browser-based miners by analysing Web logs.

Saad, Khormali and Mohaisen [31] have used public services (Pixalate and Netlab 360) to acquire a list of websites with mining code embedded. Using these sites as 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. [4], where opcode traces have been used as features, and in Draghicescu et al. [9], where the CPU allocation features and the threads and socket connections have been captured. Inlined reference monitoring for Web cryptojackers have been proposed by Wang et al. [38].

Binary-based cryptojacking. Malicious Bitcoin cryptominers have been investigated by Huang et al. [18] already in 2014. Division into campaigns and profits generated by the recent binary-based cryptominers have been analysed by Pastrana and Suarez-Tangil [26]. The data collection approach used in [26] 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 related to browser-based and binary-based cryptojacking, ours focuses on the mining applications in the Android ecosystem. Our results show that the Android platform is affected by both Web-based and binary cryptojackers. Yet, we collected and analyzed not only malicious cryptominers, as in [26], but also bona fide miners that some users might want to explore. 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.

To the best of our knowledge, the SophosLabs report on mining Android apps [34] is the only paper analyzing Android mining malware and providing some samples. We have used this report to seed our miner dataset, as it only mentions a few samples hashes (not all samples were obtainable through our main app sources, VirusTotal and Koodous, and available application markets or the app database AndroZoo [1]).

7.1.1 Energy and CPU Consumption Evaluation.

Recently, Clay et al. [6] have evaluated the CPU consumption required for mining on Android devices. As mentioned, Saad, Khormali and Mohaisen [31] 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). The authors reported that mining scripts have had significant impact on the CPU and battery (at least 40% of CPU usage on Android with low throttle and higher rate of battery charge consumption.

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 [31, 4], our solution for dynamic miner detection is based on comparison of mining apps with benign but fully functional ones.

Several approaches for detecting Android malware based on energy consumption fingerprints have been proposed and evaluated, e.g., [23, 15, 3, 5, 13]. Yet, these works focused on detection of malware behaviors other than mining.

7.1.2 Android malware detection.

As our analysis of VirusTotal results and security industry reports show [10], 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., [19, 2, 14, 42, 39, 41, 45, 36], to name just a few. Particularly, the cross-language Dual-Force technique [35] has a big potential for Web-based miner detection.

8 Conclusions

Cryptojacking poses a serious threat to mobile devices. At best, illicit cryptocurrency miners deplete the battery of mobile devices fast. However, they may cause more serious damage: from monetary loss to physical harm to the device's owner due to overheating. 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 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.

In addition, we performed the analysis of the miners from our sample with *VirusTotal*. Our findings are very interesting: **16** miners from our dataset are not detected by any antivirus engine, and a single miner has been detected by at most *39 out of the total of 74* scanners available at *VirusTotal*, meaning that there is no consistency among the scanner engines.

With the clean dataset available, we performed a dynamic analysis of the miners and compared the results with benign applications. We identified a set of dynamic metrics that contribute the most to the accurate classification results. Indeed, based on our dataset, we managed to achieve 95% of accuracy with the AUC score of whoping 0.988 ± 0.009, according to the 10-fold cross validation. Based on our findings, we proposed a tool called BRENNTDROID that can be used to detect miners at runtime.

For the future work, we plan to extend our dataset with more illicit cryptocurrency miners that use heavy code obfuscation and to study them. We also plan to extend BRENNTDROID with reliable techniques that account for various static and dynamic evasion methods such as network traffic obfuscation, CPU throttling, and evading user interaction (e.g., mining only when a user does not interact with a device, or at night).

References

[1] ALLIX, K., BISSYANDÉ, T. F., KLEIN, J., AND LE TRAON, Y. Androzoo: Collecting millions of android apps for the research community. In *Mining Software Repositories (MSR), 2016 IEEE/ACM 13th Working Conference on* (2016), IEEE, pp. 468–471.

[2] ARP, D., SPREITZENBARTH, M., HUBNER, M., GASCON, H., AND RIECK, K. Drebin: Effective and explainable detection of android malware in your pocket. In *Proc. of NDSS* (2014), pp. 23–26.

[3] CANFORA, G., MEDVET, E., MERCALDO, F., AND VISAGGIO, C. A. Acquiring and analyzing app metrics for effective mobile malware detection. In *Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics* (2016), ACM, pp. 50–57.

[4] CARLIN, D., O'KANE, P., SEZER, S., AND BURGESS, J. Detecting cryptomining using dynamic analysis. In *2018 16th Annual Conference on Privacy, Security and Trust (PST)* (2018), IEEE, pp. 1–6.

[5] CAVIGLIONE, L., GAGGERO, M., LALANDE, J.-F., MAZURCZYK, W., AND URBAŃSKI, M. 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.

[6] CLAY, J., HARGRAVE, A., AND SRIDHAR, R. A power analysis of cryptocurrency mining: A mobile device perspective. In *2018 16th Annual Conference on Privacy, Security and Trust (PST)* (2018), IEEE, pp. 1–5.

[7] COINHIVE. Discontinuation of coinhive, February 2019.

[8] CYBER THREAT ALLIANCE. The illicit cryptocurrency mining threat, https://www.cyberthreatalliance.org/wp-content/uploads/2018/09/CTA-Illicit-CryptoMining-Whitepaper.pdf, 2018.

[9] DRAGHICESCU, D., CARANICA, A., VULPE, A., AND FRATU, O. Crypto-mining application fingerprinting method. In *2018 International Conference on Communications (COMM)* (2018), IEEE, pp. 543–546.

[10] EITZMAN, R., GOODY, K., WOLCOTT, B., AND KENNELLY, J. 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, 2018.

[11] ESKANDARI, S., LEOUTSARAKOS, A., MURSCH, T., AND CLARK, J. A first look at browser-based cryptojacking. *arXiv* (2018).

[12] FELDMAN, S., STADTHER, D., AND WANG, B. Manilyzer: automated android malware detection through manifest analysis. In *Mobile Ad Hoc and Sensor Systems (MASS), 2014 IEEE 11th International Conference on* (2014), IEEE, pp. 767–772.

[13] GAO, X., LIU, D., LIU, D., AND WANG, H. On energy security of smartphones. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy* (2016), ACM, pp. 148–150.

[14] GRACE, M., ZHOU, Y., ZHANG, Q., ZOU, S., AND JIANG, X. Riskranker: scalable and accurate zero-day android malware detection. In *Proceedings of the 10th international conference on Mobile systems, applications, and services* (2012), ACM, pp. 281–294.

[15] HOFFMANN, J., NEUMANN, S., AND HOLZ, T. Mobile malware detection based on energy fingerprints—a dead end? In *International Workshop on Recent Advances in Intrusion Detection* (2013), Springer, pp. 348–368.

[16] HONG, G., YANG, Z., YANG, S., ZHANG, L., NAN, Y., ZHANG, Z., YANG, M., ZHANG, Y., QIAN, Z., AND DUAN, H. How you get shot in the back: A systematical study about cryptojacking in the real world. In *Proc. of CCS* (2018).

[17] HOSHSADIQ. Nocoin adblock list https://github.com/hoshsadiq/adblock-nocoin-list, 2019.

[18] HUANG, D. Y., DHARMDASANI, H., MEIKLEJOHN, S., DAVE, V., GRIER, C., MCCOY, D., SAVAGE, S., WEAVER, N., SNOEREN, A. C., AND LEVCHENKO, K. Botcoin: Monetizing stolen cycles. In *NDSS* (2014), Citeseer.

[19] JIANG, X., AND ZHOU, Y. Dissecting android malware: Characterization and evolution. In *Proc. of S&P* (2012).

[20] KONOTH, R. K., VINETI, E., MOONSAMY, V., LINDORFER, M., KRUEGEL, C., BOS, H., AND VIGNA, G. Minesweeper: An in-depth look into drive-by cryptocurrency mining and its defense. In *Proc. of CCS* (2018).

[21] LUO, T., HAO, H., DU, W., WANG, Y., AND YIN, H. Attacks on webview in the android system. In *Proc. of ACSAC* (2011).

[22] MA, W., CAMPBELL, J., TRAN, D., AND KLEEMAN, D. Password entropy and password quality. In *Proc. of NSS* (2010).

[23] MERLO, A., MIGLIARDI, M., AND FONTANELLI, P. Measuring and estimating power consumption in android to support energy-based intrusion detection. *Journal of Computer Security 23*, 5 (2015), 611–637.

[24] MUSCH, M., WRESSNEGGER, C., JOHNS, M., AND RIECK, K. Web-based cryptojacking in the wild. *arXiv preprint arXiv:1808.09474* (2018).

[25] PAPADOPOULOS, P., ILIA, P., AND MARKATOS, E. P. Truth in web mining: Measuring the profitability and cost of cryptominers as a web monetization model. *arXiv preprint arXiv:1806.01994* (2018).

[26] PASTRANA, S., AND SUAREZ-TANGIL, G. A first look at the crypto-mining malware ecosystem: A decade of unrestricted wealth. *arXiv preprint arXiv:1901.00846* (2019).

[27] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M., AND DUCHESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[28] QUALCOMM TECHNOLOGIES, INC. Snapdragon profiler https://developer.qualcomm.com/software/snapdragon-profiler, 2019.

[29] RAUCHBERGER, J., SCHRITTWIESER, S., DAM, T., LUH, R., BUHOV, D., PÖTZELSBERGER, G., AND KIM, H. The other side of the coin: A framework for detecting and analyzing web-based cryptocurrency mining campaigns. In *Proceedings of the 13th International Conference on Availability, Reliability and Security* (2018), ACM, p. 18.

[30] RÜTH, J., ZIMMERMANN, T., WOLSING, K., AND HOHLFELD, O. Digging into browser-based crypto mining. In *Proc. of IMC* (2018).

[31] SAAD, M., KHORMALI, A., AND MOHAISEN, A. End-to-end analysis of in-browser cryptojacking. *arXiv preprint arXiv:1809.02152* (2018).

[32] SALEM, A., PAULUS, F. F., AND PRETSCHNER, A. Repackman: a tool for automatic repackaging of android apps. In *Proceedings of the 1st International Workshop on Advances in Mobile App Analysis* (2018), ACM, pp. 25–28.

[33] SMALI/BACKSMALI. `https://github.com/JesusFreke/smali`, 2018.

[34] SOPHOS LABS. Coinminer and other malicious cryptominers targeting android, 2018.

[35] SUFATRIO, D. J. T., CHUA, T.-W., AND THING, V. Securing android: a survey, taxonomy, and challenges. *ACM Computing Surveys (CSUR) 47*, 4 (2015), 58.

[36] TAM, K., FEIZOLLAH, A., ANUAR, N. B., SALLEH, R., AND CAVALLARO, L. The evolution of android malware and android analysis techniques. *ACM Computing Surveys (CSUR) 49*, 4 (2017), 76.

[37] TUNG, L. Android security: Coin miners show up in apps and sites to wear out your cpu, October 2017.

[38] WANG, W., FERRELL, B., XU, X., HAMLEN, K. W., AND HAO, S. Seismic: Secure in-lined script monitors for interrupting cryptojacks. In *European Symposium on Research in Computer Security* (2018), Springer, pp. 122–142.

[39] WANG, W., WANG, X., FENG, D., LIU, J., HAN, Z., AND ZHANG, X. Exploring permission-induced risk in android applications for malicious application detection. *IEEE Transactions on Information Forensics and Security 9*, 11 (2014), 1869–1882.

[40] WIŚNIEWSKI, R., AND TUMBLESON, C. Apktool - A tool for reverse engineering 3rd party, closed, binary Android apps.`https://ibotpeaches.github.io/Apktool`, 2017.

[41] XU, L., ZHANG, D., JAYASENA, N., AND CAVAZOS, J. Hadm: Hybrid analysis for detection of malware. In *Proc. of IntelliSys* (2016).

[42] YANG, W., XIAO, X., ANDOW, B., LI, S., XIE, T., AND ENCK, W. Appcontext: Differentiating malicious and benign mobile app behaviors using context. In *Proceedings of the 37th International Conference on Software Engineering* (2015), pp. 303–313.

[43] ZHAUNIAROVICH, Y., AND GADYATSKAYA, O. Small changes, big changes: an updated view on the android permission system. In *Proc. of RAID* (2016), Springer, pp. 346–367.

[44] ZHAUNIAROVICH, Y., GADYATSKAYA, O., AND CRISPO, B. Demo: Enabling trusted stores for android. In *Proc. of CCS* (2013), pp. 1345–1348.

[45] ZHU, Z., AND DUMITRAS, T. Featuresmith: Automatically engineering features for malware detection by mining the security literature. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), ACM, pp. 767–778.