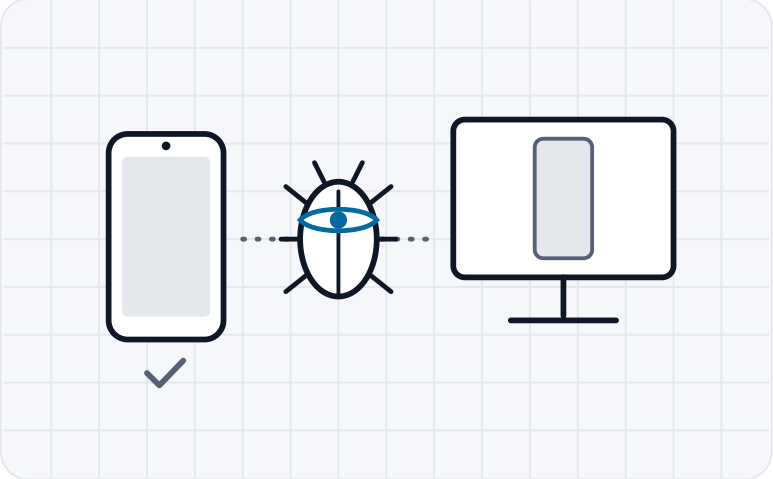
A-SIT Technology
E-Government Innovation Center

23 October 2023 Analysis · Mobile Security · 15 min read

# Detection of emulators in malware analysis on Android

Gerald Palfinger



**Authors** Gerald Palfinger
**Version** 2023-10-06
**Source** https://technology2.a-sit.at/en/publications/detection-of-emulators-in-malware-analysis-on-android/
**Note** Machine translation.

## Key findings

- Emulator and Pixel 4 (both Android 13) differ in 178 methods, 123 fields and 49 content providers of the API.
- The differences concern software, hardware, preconfigured and user-influenced settings, and other artefacts.
- The emulator always reports T-Mobile USA as network operator; Google Mobile Services are missing despite the Google Play configuration.
- The emulator does not support attestation because it lacks cryptographic hardware.

- Malicious apps can exploit these differences to detect analysis environments and hide their behaviour there.

To prevent malware from spreading, applications are examined before they are published in app stores. Dynamic analysis, in which the application is executed, plays an important role here. For practical reasons, Android emulators are often used instead of real devices for this purpose. However, small differences between emulators and real devices potentially allow malicious applications to detect the analysis and thus hide their malicious behaviour throughout the analysis. This report addresses what differences have been found in the literature to distinguish emulators from real devices. Furthermore, an experiment is conducted to see what differences can be found using the programming interface between real devices and the emulator.

## 1. Introduction

Smartphones are an attractive target for malware because of the large amount of information they store, the functions they provide and security-relevant uses such as serving as a second factor. Unlike desktop applications, which are often downloaded manually from the internet via the browser, smartphone applications are usually obtained from one or a few central application sources. As a result, the operators play an important role as "gatekeepers" between the (potentially malicious) developers and the end users. However, given the large number of submitted applications, it is difficult to examine each application manually. Likewise, malicious behaviour is often not immediately apparent, and the source code of the application is not available to the store operators. For this reason, applications are examined by automated static and dynamic analysis tools before publication. Dynamic analysis often relies on emulators, as these have certain advantages in handling. For example, an emulator can quickly be reset to its initial state after the analysis. Emulators running on standard server hardware are also more scalable than specialised device farms. While Android emulators try to emulate devices, they do not always succeed in every detail. Certain static information, such as model names or network details, can point to an emulator. In addition, the emulation of certain hardware may be missing or may behave differently from a real device. Likewise, the presence of certain artefacts or the absence of signs of use, such as a small number of files, applications or contacts, can indicate an emulator. This report shows which ways of distinguishing an emulator from a real device have been found in the literature. Furthermore, an experiment examines which differences between an Android emulator and a real device can be found using the programming interface.

## 2. Related work

In [1], an attempt was made to detect malware analysis environments by means of various properties. Malware could use such properties to detect these environments. The researchers divided the detection heuristics into three categories: static properties, readings from the device sensors, and intrinsic properties of the emulation environment used. Information about the connected network, build information and device IDs were identified as static detection features. Many of the emulators used could emulate device sensors such as the accelerometer, but the emulated values showed a distinct pattern that repeated over a short period with only slight deviation. Finally, differences in the processor architecture and optimisations in its emulation also indicate the use of an emulator. For example, the virtual program counter is not incremented

after every executed instruction, which can improve the performance of the emulator. The behaviour of the processors also differs when executing self-modifying code. This is due to differences in the design of instruction and data caches in the processor architectures. These two differences make it possible to detect whether code is running on a real device or in an emulator. In the analysis, various malware analysis environments were tested for their detectability using the established heuristics. Only one of the examined environments could not be detected using the defined static detection features. The sensor readings were emulated in many environments. Detecting the intrinsic properties of the emulation environment requires native code. This is not supported by every analysis environment. However, all examined environments that support native code allow the emulation to be identified unambiguously.

With Morpheus [2], heuristics that make it possible to detect emulated environments can be found automatically. To this end, the framework is run on both emulators and real devices. In a first step, data is collected from real devices and emulators with the help of a purpose-built program. The data was collected from three different sources: information about the structure and content of the file system, information from the programming interface, and information from the system properties. These were then compared between the devices and the emulators. More than 10,000 different information sources were found that can potentially be used as heuristics. A large part of the sources found come from the file system, with more than 9,800 potentially usable heuristics. In addition, slightly fewer than 400 potential heuristics each were discovered in the programming interface and the system properties. The heuristics found were evaluated by using them to distinguish malware analysis systems from a set of real devices. Only the 10 most important heuristics from each category were selected. This was because the execution time was severely limited on many of the emulators that are only available online, so not all heuristics could be retrieved. The analysis found that the heuristics from the file system performed best and reached an accuracy of 97%, followed by the system properties with 90%. The information from the programming interface performed considerably worse, however, with an accuracy of only 63%.

In [3], the efficiency of malware analysis on emulators and on real devices is compared. This makes it possible to determine whether applications behave differently on real devices than in the emulator. More applications could also be examined successfully, since real devices have more functions than the emulator. Various machine learning algorithms were used for the comparison. The analysis found that almost 99% of the 1222 malware samples examined could be executed on real devices, while only about 77% could be executed on the emulator. It was also found that some methods of the programming interface are executed only on real devices, or by more applications there. As a result, more malware could also be detected correctly on the real devices.

[4] presents methods for detecting emulators based on discrepancies in behaviour at the instruction level between software-based emulators and real ARM CPUs. To this end, traces of application execution were collected on a large scale, in particular the changes to registers and memory. An accurate software model of an ARM CPU was created for this purpose. In the work, various test applications were run both in this model and in the emulator. The framework that was created automatically identifies instructions that cause deviating behaviour between emulated CPUs and the model. The differences detected could be divided into three main classes. Some of the causes found could be fixed without performance losses by making changes to the emulator.

In [5], it was examined how the data collected during dynamic analysis differs between real devices and emulators. For this purpose, 110 benign and 110 malicious applications were examined. These were examined both on a real device and on an emulated device. During the examination, all system calls triggered by the examined applications were recorded. The evaluation of the data found that there are differences between the real devices and emulators. More system calls were executed on the emulator, but these correlated more strongly than on real devices.

In [6], current malware analysis environments were once again examined for their detectability compared with real devices. An application was created that retrieves both static information, such as device details, and so-called usage artefacts of the examined Android environment. In the study, these include information such as the number of contacts and calendar events, the number of stored media files and reminders, and certain device settings such as the number of stored wireless networks. However, retrieving this information often requires permission from the user. The information sources examined were selected manually by the authors from the documentation of the Android programming interface. In the experiments carried out, 12 different malware analysis environments were examined, as well as the intake processes of several application markets. Using machine learning, an attempt was made to distinguish the examined analysis environments from real devices on the basis of the information collected. For this purpose, information from more than 1,200 real and actively used Android devices was collected beforehand. The evaluation found that the model created has an accuracy of over 98% in distinguishing between a real device and a malware analysis environment. It was also shown that many analysis environments emulate device models inadequately, for example because manufacturer customisations such as preinstalled applications, or the hardware, differ from a real device.

CamoDroid [7] is a malware analysis environment that was created with the aim of reproducing real devices as realistically as possible in order to prevent detection. To this end, sensors, static information, network details and user inputs are emulated. Accesses to potentially dangerous methods of the programming interface are also monitored, and realistic values are returned. In the experiments, CamoDroid was tested both with Morpheus [2] and against real malware. 219 of the 227 malware samples tested in total could be executed successfully without concealing their malicious behaviour by detecting the malware analysis environment.

## 3. Methodology

To identify the differences between real devices and emulators, the data from the programming interface is collected systematically. To this end, the methods of the programming interface are called and their return values are collected. The fields of classes are also retrieved. Finally, the Android-specific content providers are read as well. The collected data is stored in a textual representation. To filter out temporary information, the collection of information is run twice on each device or emulator. The device or emulator is restarted in between. After the information has been collected, it is compared between the examined devices and emulators. Information from the same source that differs between devices and emulators is marked as a potential distinguishing feature. These were then inspected manually and divided into categories for clarity. The results are presented in the following chapter. While the related work often chose information sources that require a permission, this study concentrates on sources that require no separate permission from the user. This allows an emulator to be detected as inconspicuously as possible.

## 4. Results

In the experiments, the Android Emulator provided by Google was compared with a physical Pixel 4 device. Android 13 was installed on both. When configuring the emulator, a Pixel 4 with Google Play was selected as the emulation target. The information sources found are shown in Table 1. They include both sources that have already been found in the literature and new information sources. The already known sources include, for example, the build information, CPU architecture or network details. In principle, the information sources found can be divided into missing or differently configured software components, hardware functions, preconfigured settings, settings influenced by the user, and other artefacts. The differences found are examined in more detail below.

Various methods and content providers showed that certain Google software components are missing, even though the emulator was configured with Google Play. These include the absence of the Google Mobile Services (com.google.android.gms), the feedback component and various notification components from Google. The number of installed applications and the number of preinstalled codecs are also lower on the emulator. Preinstalled modules, content providers and applications, as well as their accounts, also differ between emulator and device. The available permission groups also differ between emulator and real device. In addition, certain constants differ, as do certain operating system characteristics such as concurrency settings for multithreading or the size of the socket buffer. Some shared libraries are also missing on the emulator, especially those that come from the SoC manufacturer Qualcomm and can only be found on the real device. The real device supports two additional accessibility features compared with the emulator. Certain functions such as SIP or VoIP are likewise available only on the real device. On the software side, certain labels, system properties, supported languages and the time zone version also differ.

As with the software components, there are also numerous differences between the emulator and the real device on the hardware side. These include the number of cameras, which is higher on the real device than on the emulator. The emulator also supports only one modem, whereas the real device can use two mobile connections at the same time. The real device offers more audio functions, such as noise suppression. The reported audio latency also differs and is lower on the emulator. Compared with the emulator, the device has more streams in the audio database. Certain constants for the support of Bluetooth devices also differ. The supported functions of the GPS hardware and the year of manufacture of the hardware differ as well. The size of the main memory is also smaller when the emulator's Pixel 4 profile is used than on the real device. With regard to the screen, a lower pixel density was found on the emulator, which matches the computer monitor used. Owing to the lack of cryptographic hardware, the emulator does not offer attestation. The available cryptographic algorithms also differ between the two. The MMS user agent and the MMS profile differ as well. On the emulator, the latter still refers to Nexus devices. When the default settings of the emulator are used, the properties of the persistent storage also differ, such as the size of the internal storage or the presence of a memory card on the emulator. With regard to hardware, the number of input devices, the SoC model and the functions of the WiFi hardware also differ.

Different notification sounds are preconfigured on the emulator than on the device. The preset device name also differs. Settings relating to device security also differ between the real device and the emulator, in particular the use of a PIN. On the real device, this is usually already set during setup, whereas the emulator does not suggest this. Numerous other user-customisable settings could also be found via a content

provider. Since these are usually not changed on the emulator, a change in a large number of these settings points to a real device. The network operator also usually differs between emulator and device. The emulator always reports T-Mobile USA.

Table 1 Differences found between emulator and real device

| Type | # Methods | # Fields | # Content Providers |
|---|---|---|---|
| (Missing) Google components | 2 | | 1 |
| Other constants | 7 | 2 | 3 |
| android.os.Build information | | 18 | |
| Number of installed apps | 8 | | |
| Number of installed codecs | 1 | | |
| Number of cameras | 2 | | |
| Number of modems | 2 | | |
| Audio functions | 9 | | |
| Audio latency | 1 | | |
| Audio streams | 1 | | |
| Notification sounds | 19 | | 6 |
| Permission groups | 1 | | |
| Operating system settings | 2 | | |
| Bluetooth functions | 5 | | |
| CPU architecture | | 5 | |
| Input devices | 3 | | |
| Feedback component | 4 | | |
| Accessibility features | 2 | | |
| Device name | | | 2 |
| Device security (PIN set) | 3 | | |
| Shared libraries | 2 | | |
| GPS hardware | 4 | | |
| Size of main memory | 2 | | |
| Insets type | 2 | 1 | |
| Java system properties | 1 | | |
| MMS profile & user agent | 2 | | |
| Mobile network operator | 8 | | |

| Type | # Methods | # Fields | # Content Providers |
|---|---|---|---|
| Network details | 7 | | |
| Persistent storage (settings, volume, size) | 27 | | |
| Pixel density | 1 | | |
| Resource labels | 13 | 96 | |
| SIP support | 1 | | |
| SoC model | | 1 | |
| Tethering | 1 | | |
| Supported cryptographic algorithms | 3 | | |
| Supported languages | 1 | | |
| Support for attestation | 3 | | |
| Available account types | 1 | | |
| Modem version | 2 | | |
| VoIP support | 1 | | |
| Preinstalled apps | 6 | | |
| Preinstalled content providers | 1 | | |
| Preinstalled modules | 1 | | |
| Other user-customisable settings | | | 36 |
| Other functions | 5 | | |
| WiFi functions | 10 | | 1 |
| Time zone version | 1 | | |
| Total | 178 | 123 | 49 |

## 5. Conclusion

This report has shown which types of differences exist between real devices and Android emulators that malicious applications can exploit to suppress their malicious behaviour during examinations and thus remain undetected. These include static information such as model number, device information or network details. Certain hardware that is standard on smartphones may also be missing or behave differently. Sensors in particular often deliver static or implausible values. In addition, there may be artefacts that are present only during emulation, such as certain drivers or applications. The emulation of the processor

architecture can cause noticeable differences in the execution of certain instructions. Furthermore, artefacts of use are often missing. This becomes apparent through a small number of stored files, contacts or calendar entries, or a small number of installed applications.

## References


[1] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis und S. Ioannidis, „Rage against the virtual machine: hindering dynamic analysis of Android malware," EuroSec '14: Proceedings of the Seventh European Workshop on System Security, p. 1–6, 2014.

[2] Y. Jing, Z. Zhao, G.-J. Ahn und H. Hu, „Morpheus: automatically generating heuristics to detect Android emulators," Proceedings of the 30th Annual Computer Security Applications Conference, p. 216–225, 2014.

[3] M. K. Alzaylaee, Y. Y. Suleiman und S. Sezer, „EMULATOR vs REAL PHONE: Android Malware Detection Using Machine Learning," IWSPA '17: Proceedings of the 3rd ACM on International Workshop on Security And Privacy Analytics, p. 65–72, 2017.

[4] O. Sahin, A. K. Coskun und M. Egele, „Proteus: Detecting Android Emulators from Instruction-Level Profiles," International Symposium on Research in Attacks, Intrusions, and Defenses, p. 3–24, 2018.

[5] A. Guerra-Manzanares, H. Bahsi und S. Nõmm, „Differences in Android Behavior Between Real Device and Emulator: A Malware Detection Perspective," Sixth International Conference on Internet of Things: Systems, Management and Security, 2019.

[6] B. Kondracki, B. A. Azad, N. Miramirkhani und N. Nikiforakis, „The Droid is in the Details: Environment-aware Evasion of Android Sandboxes," Annual Network and Distributed System Security Symposium, 2022.

[7] F. Faghihi, M. Zulkernine und S. Ding, „CamoDroid: An Android application analysis environment resilient against sandbox evasion," Journal of Systems Architecture, Bd. 125, 2022.

[8] J. Lin, C. Liu und B. Fang, „Out-of-Domain Characteristic Based Hierarchical Emulator Detection for Mobile," 2nd International Conference on Information Technologies and Electrical Engineering, p. 1–5, 2019.

[9] J.-d. Lim, I.-k. Kim, N. Kim, B. Kang und S.-j. Cho, „A Study on Android Emulator Detection Using Build Properties," The 8th International Conference on Next Generation Computing, pp. 321-324, 2022.