Risk Analysis of Android Applications: A Multi-Criteria and Usable Approach

F. Del Bene, G. Dini, F. Martinelli, I. Matteucci, M. Petrocchi, A. Saracino, D. Sgandurra

IIT TR-04/2015

Nota Interna

Maggio 2015

Istituto di Informatica e Telematica
Risk Analysis of Android Applications: A Multi-Criteria and Usable Approach

Fabio Del Bene, Gianluca Dini, Fabio Martinelli, Ilaria Matteucci, Marinella Petrocchi, Andrea Saracino, Daniele Sgandurra

May 6, 2015

Abstract

The number of Android applications (apps) is constantly increasing, by bringing new functionalities on users’ smartphones and tablets everyday. Unfortunately, several apps pose many risks to the users, e.g., by including code that threaten user privacy or system integrity. Currently, most of the current security countermeasures for detecting dangerous apps show some weaknesses, mainly related to users’ understanding and acceptance. For these reasons, users would benefit from an effective but simple technique that indicates whether an app is safe or risky to be installed. In this paper, we present MAETROID (Multi-criteria App Evaluator of TRust in AndrOID), a framework to evaluate the trustworthiness of Android apps, i.e. the amount of risk they pose to the users, e.g. in terms of confidentiality and integrity. The framework performs a multi-criteria analysis of an app at deploy-time and returns a single easy-to-understand evaluation on the app’s risk level, aimed at driving the user decision on whether installing or not a new app. The used criteria include the set of requested permissions and a further set of metadata retrieved from the marketplace, which denote the app quality and popularity. We have classified 11,000 Android apps coming from Google Play and from a database of known malware. In particular, MAETROID has recognized as dangerous all the apps belonging to the database of malicious apps, while about 20% of apps from Google Play have been classified as medium risk. To evaluate the users’ experience in the usage of MAETROID, we have collected and analyzed data from a survey that was completed by 189 subjects. The survey has measured the users’ response to MAETROID, and has shown that MAETROID is more effective than the standard Android permission system in informing the user about an app risk level. Over the whole set of interviewees, MAETROID has been able to drive correctly the user in the decision of installing an app in more than 90% of cases.

1 Introduction

In the last years, smartphones and tablets have replaced legacy GMS/GPRS (2G) mobile phones. According to a recent market analysis [1], in several
countries more than 90% of registered mobile devices have a 3G or 4G subscription, leading to more than 2 billions active subscribers. Noticeably, more than 80% of these devices are based on the Android operating system, which is the most popular operating system (OS) for smartphones and tablets [2]. Such a dominance of use makes Android also the almost exclusive target for mobile threats identified in the last years [3]. Currently, 99% of the Android security attacks are brought through infected mobile applications (apps) [3].

Currently, apps for mobile devices are distributed through online marketplaces, such as Google Play or App Store. These marketplaces act as a hub where app developers publish their own products, which can be bought or downloaded for free by users. Usually, official markets charge users for these apps, while several unofficial marketplaces distribute their own apps free of charge. In this last case, trust is at risk, since there is no centralized control, as it happens with official markets, and it may happen that untrusted developers distribute malicious apps. This issue is particularly serious for Android – which represents the reference of our study – as it is both the most popular OS for mobile devices and the system with the greatest share of malware in the last years [3]. In particular, in 2013 Android accounted for 97% of all mobile malware. Moreover, the number of new malware is alarming: on average, more than 160,000 new specimens are reported everyday [4]. Recently, work in [5] pointed out that a quarter of all Google Play free apps are clones, i.e., repackaged apps of popular ones, such as WhatsApp and Angry Birds.

Android security policy it is not based on a strict control of the app distribution method. In fact, apps are published on the official market without strong security checks and Google is not condemning the presence of unofficial markets. This approach favors a greater flexibility and freedom of choice for both developers and users. Differently from Apple, which enforces application security directly on the market, publishing only apps that have passed a long and complex verification process (vetting), Android enforces security directly on the device. Android, in fact, implements the Permission System to force app developers to declare the security critical resources that the app can access and the security critical operations that the app can perform. Then, at run-time, the Android OS blocks any undeclared access or illegal operation attempt. The Android permission system has been proved to show flaws and weaknesses (see [6]), mainly for some assumptions on users' expertise in understanding the permission semantics. In fact, before the app is installed, requested permissions are shown to the user as a list. Then, the user has to decide if the app is trustworthy or risky by only visualizing the permissions list. Hence, if the user decides that some permissions are not justified, the installation procedure is aborted. Unfortunately, several users may not have enough expertise to understand whether an app is malicious or not by reading the permissions list only. Moreover, according to the analysis in [6], a large number of users does not even read permissions and simply installs the app. In this case, the permission system does not help such users in protecting them from malicious apps. The main criticism that has been raised against the permission system is that they are too coarse-grained and difficult to understand, both for users and developers.
We argue that an evaluation of the Android apps trustworthiness deserves more representative methods and techniques.

In this paper, we present MAETROID (Multi-criteria App Evaluator of TRust in AndrOID), a mechanism to evaluate the trustworthiness of Android apps, i.e., their level of risk in terms of potential security and privacy risks. To this end, MAETROID performs a static analysis of the app by using five different parameters, found in the app itself or retrieved from the marketplace. Then, MAETROID builds a single easy-to-understand trustworthiness decision for the app, which is shown to the user when installing a new app, i.e., at deploy-time. MAETROID is based on a customized instantiation of a well-known multi-criteria decision process, the Analytic Hierarchy Process (AHP) [7]. AHP conveys in a single index an evaluation performed through criteria which are both objective and subjective. MAETROID exploits AHP to analyze apps through both objective criteria, i.e., declared permissions, number of downloads, and market, and subjective ones, i.e., the app rating and the developer reputation. The outcome of MAETROID is an advise suggesting the user whether the considered app should be installed or not. The approach of MAETROID is scalable, since apps are evaluated at deploy-time, directly on the user device. Thus, despite the huge amount of available Android apps and their possible different provenances, the proposed approach is viable, since (i) it simply requires the users to install the MAETROID app on their devices (ii) apps are evaluated at installation time only.

We have tested the framework by classifying more than 11,000 apps. Furthermore, we have analyzed the user response and acceptance of MAETROID, designing and proposing a survey to a set of about 200 subjects. The survey results have been analyzed to synthesize outcomes on the users perception of mobile security and related threats. In particular, in our user-set, subjects are aware of mobile security threats, since only 10% of the interviewees state that they are not concerned about possible threats at all. However, more than 25% of the subjects gives no importance to the Android security warnings and only 27% of the subjects considers the Android permissions as useful and meaningful. The results of the survey have also confirmed that the MAETROID evaluation is effective in driving the users decision, by avoiding the installation of malicious apps. In particular, 90% of the interviewees changed their mind about installing a malicious app after that MAETROID evaluates the app as dangerous.

Contributions of the Paper The contributions of this paper are the following:

- we propose an evaluation of the app threat level based on the analysis of the threats represented by each declared permission. We have rated each of the 145 Android permissions with three threat values, which correspond to threats to user’s privacy, device, and financial; these values are used to compute the app global threat score, based on the required permissions. The threat score is the first of five criteria exploited by our framework;
- we describe the design of MAETROID, a system for the analysis of Android apps at deploy-time. Whenever a new app is being in-
stalled on the user mobile device, MAETROID exploits five criteria to evaluate the app trustworthiness, then returns to the user a simple decision to help the user in deciding whether to install or not the app:

- we have tested MAETROID against a set of more than 11,000 apps, coming either from Google Play, unofficial markets and two important mobile malware database, namely Genome [8] and Contagio1;
- we present the implementation of MAETROID for Android devices (which can be downloaded from: http://icaremobile.iit.cnr.it/);
- we discuss the results about users’ expectations and MAETROID acceptance, by detailing the results of a survey presented to almost 200 mobile device users. The survey was aimed at evaluating (i) the users understanding and knowledge of security mechanisms already existing on their mobile devices, (ii) the criteria that average users consider representative of the app trustworthiness, and (iii) whether the MAETROID outcome is more effective than the Android permission system in driving the user towards the right decision (whether to install or not install a new app).

In a nutshell, the current work largely extends and improves the description of previous work in [9]. The completely novel contributions are the implementation of MAETROID as an Android app, the classification of 11,000 apps and the study on the user’s acceptance and understanding of MAETROID through the survey.

Structure of the Paper In the next section, we discuss related work on the security of mobile devices. Section 3 recalls some notions about Android security mechanisms and provides a brief description of the Analytic Hierarchy Process. Section 4 describes the MAETROID approach by discussing in detail the criteria used for assessing the trustworthiness of an app. The current implementation of MAETROID for Android devices is presented in Section 5, which also discusses the results of the analysis on the testbed apps. Section 6 presents a survey, which was submitted to 189 subjects, to test user’s perception on security on mobile devices and users’ acceptance of MAETROID approach. In Section 7, we discuss the MAETROID framework, by discussing its advantages and limitations, by comparing it with alternative solutions, and finally we analyze in details the results and findings of the survey. Finally, Section 8 draws some conclusions and proposes some further future research directions.

2 Related Work

Several extensions and improvements to the Android permission system have been recently proposed. The work presented in [10] proposes a security framework that regulates the actions of Android apps defining security rules concerning permissions and sequence of operations. New rules can

1http://contagiominiidump.blogspot.it/
be added using a specification language. The app code is analyzed at deploy-time to verify whether it is compliant to the rule, otherwise it is considered as malicious code. With respect to this work, MAETROID does not require the code to be decompiled and analyzed. Indeed, it only requires the permissions list that can be retrieved from the manifest file and other pieces of information that can be retrieved from the website where the app can be downloaded.

Authors of [11] present a finer grained model of the Android permission system. They propose a framework, named TISSA, that modifies the Android system to allow the user to choose the permissions she wants to grant to an app and those that have to be denied. Using mocking data, they ensure that an app works correctly even if it is not allowed to access the required information. However, their system focuses on the analysis of privacy threatening permissions and it relies on the user expertise and knowledge. A work similar to TISSA is presented in [12], where the authors design an improved app installer that allows users to define three different policies for each permission: allow, deny, or conditional allow. Conditional allow is used to define a customized policy for a specific permission by means of a policy definition language. However, the responsibility of choosing the right permissions still falls on the user, whilst MAETROID directly shows to the user the risk classification of the app, performing automatically the permissions analysis.

In [13], the authors present a multi-level behavior-based intrusion detection system called MADAM. The proposed system learns the correct devices’ behavior and then detects significant deviations signaling an intrusion. The MADAM approach is orthogonal to that of MAETROID because MADAM analyzes the app behavior at run-time, while MAETROID performs a risk analysis before installing the app. In [14], apps have been classified based on their required permissions. Apps have been divided in functional clusters by means of Self Organizing Maps, proving that apps with the same set of permission have similar functionalities. However this work does not differentiate between good and bad (trojanized) apps. Another analysis of Android permissions is presented in [15], where the authors discuss a tool named Stowaway, which discovers permission overdeclaration errors in apps. Using this tool, it is possible to analyze the 85% of Android available functions, including the private ones, to obtain a mapping between functions and permissions. This work mainly concerns the analysis of permissions without proposing a direct link between declared permissions and apps security, as with MAETROID. A system to implement security policies on Android devices is presented in [16]. This system is based on the introduction of a monitor of security critical functionalities, which matches the performed actions with security policies defined by the mobile device user. However, the presence of the monitor imposes a consistent overhead. Another security framework based on user defined policies, preventing app to perform non compliant operations, is presented in [17]. The framework attempts to reduce the overhead and to improve the effectiveness through a probabilistic contract based approach. This leads to a probabilistic satisfaction of security requirements.

TrustGo [18] is a framework aimed at classifying mobile apps exploiting a multi-criteria analysis. TrustGo gives users a full description of the secu-
rity threats brought by an app and also works as an antivirus. TrustGo is catalogue-based: available Android apps are analyzed by security experts and are inserted in a catalogue, checked when a TrustGo user is installing a new app. TrustGo is effective and the catalogue-based approach ensures a good accuracy. However, it is not possible to collect all the existing apps, since only the apps distributed through official channels can be analyzed. Moreover, if an app is updated, it is possible that some security features may change in the new version, i.e., new permissions are added in the manifest and this requires a catalogue update, which may not be triggered in time. On the other hand, MAETROID is independent from the app version, i.e., the app is analyzed “as is”. App update will trigger a new classification process. Moreover, MAETROID classifies the app at deploy-time, without requiring any centralized catalogue. Thus, any app can be classified even if coming from unknown marketplaces. Another app classification system is presented in [19], where apps are classified in comparison with formerly analyzed apps. The methodology exploits probabilistic generative models to analyze apps on different criteria including permissions. However, the performed analysis is more effective in creating an awareness on developers in trying to avoid issues like permission overdeclaration, instead of providing an index effective in driving the user decision on the app installation.

Analysis of the Android permission understanding have been performed in [6], where subjects from an university campus have been asked to fill a survey on Android security and on their current approach to the permission security mechanism. The results of this survey matches with the ones discussed in the first part of our survey used to validate the effectiveness of the MAETROID evaluation. In particular, the percentage of users considering permissions when installing a new app is mostly equivalent. Recently, Android has introduced a service of remote monitoring of installed apps, called VerifyApps [20], which acts as a remote antivirus. The visual approach is similar to MAETROID: when an app is considered dangerous, the user is advised about the potential threat and asked if she desires to install the app considered dangerous. However, VerifyApps behaves like an antivirus, by looking directly for known malware signature. On the contrary, the risk analysis of MAETROID is based on different parameters that do not depend from the app code. A dangerous app can be considered malicious by MAETROID even if it is a brand new app with an unknown signature.

A similar approach to MAETROID is Androlyzer [21], which is a web-based service that gives the user a lot of information about the used API, used libraries, privacy leaks, requested permission, which might be too overwhelming for an ordinary user. For these reasons, in MAETROID we have decided to keep the output of the results as simple as possible as a first step towards a better understanding of the risk of an app from the point of average users. Furthermore, these reputation services, again with similar ones [22] [23], are usually centralized, hence they are not very scalable. In fact, these services need, first of all, to download all the apps (or the most important ones), and are usually limited to unofficial markets. Furthermore, their databases need to be constantly updated and the centralized service need to cope with several concurrent requests of
different users. On the contrary, MAETROID is run locally on the user device and only for the newly downloaded app so there are not scalability issues due to checking a large number of apps concurrently.

3 Background

This section describes the Android permission system and discusses both its strengths and weaknesses. Then, it recalls the Analytic Hierarchy Process (AHP), the multi-criteria decision system used by MAETROID.

3.1 Android Permission System

To reduce the likelihood that a user installs a dangerous apps, Android implements an access control mechanism called Permission System. The Permission System forces app developers to declare the security critical resources that the app can access and the security critical operations that the app can perform. At run-time, an Android component called permission checker monitors the access requests to security critical resources and operations. If an access request is issued by an app without authorization declared in the Permission System, the permission checker denies the access.

Figure 1: Time-line of Major Changes in the Android Permission System.

Figure 1 reports a time-line of the evolution of Android during 2009-2014, by highlighting the major changes concerning permissions and security. In particular, until 2011 existing malware for Android systems were only confined to research proof-of-concepts. In 2011, Android-specific malware started to spread, by exploiting a vulnerability of the Gingerbread system, named Gingerbreak, which allowed malicious apps to get root privileges. This issue has been fixed with the introduction of Android Ice Cream Sandwich in 2012. However, in 2013, another important permission vulnerability has been discovered (see [24]) that allowed the modification of the app permissions without modifying the app signature. This issue has been solved in the latest release of Jelly Bean (2013), which also includes for the first time the possibility to dynamically revoke, or grant, single permissions to apps.
During the period of observation (2009-2014), starting with the original set of 90 permissions, a further set of almost 50 permissions has been added, mainly due to new device resources and apps functionalities. Currently, Android defines 145 permissions\(^2\), where each permission is related to either a specific device resource or a critical operation. Permissions required by an app are declared by the developer in the `AndroidManifest.xml` file (manifest for short), which is included in the app package (apk), bound to the app code by means of digital signature. Android classifies permissions in four classes: normal, dangerous, signature, and signature-or-system. For the scope of this paper, we only focus on normal and dangerous permissions. In fact, signature and signature-or-system permissions cannot be required by any app. Only apps signed with the Google private key, thus developed by Google, can declare those permissions. The rationale behind signature and signature-or-system permission is that Google is directly interested in providing only genuine apps.

The Android permission classification is used to choose which permissions are shown to the user at deploy-time. All dangerous permissions are automatically shown to the user, whereas the normal ones are listed in a separate sub-list, the “Other Permissions” list. Once a permission has been granted, the app can access the corresponding protected resource (or perform some corresponding critical operations) without asking for further authorizations.

Several criticisms have been raised against the Android permission system. Firstly, the system is considered too coarse-grained [11], since the user can only choose whether to accept all of the permissions declared by an app or to refuse to install the app. Even if the latest versions of Android includes the `AppOps` feature, which allows users to revoke selected permissions to an already installed app, issues related to the coarse granularity of the system still exist. In fact, anytime the app tries to perform an operation for which the permission has been revoked, the operation is denied by the permission checker. Thus, since the error coming from the denied operation is not handled, the app is likely to terminate with error (it crashes).

One of the problem with such an approach is that a user is generally unable to determine if an app can be trusted. In fact, by only looking at the list of the required permissions it can be noticed that the list is not very user-friendly. An example of this is depicted in Figure 2, where we can see that it is difficult to fully understand the risk posed by such permissions. Since the number of requested permissions is rather large, and since some of them are quite difficult to understand, even for expert users, several users simply ignore them when installing a new app, leading to malicious apps being installed [6].

Another issue is that often developers declare (by mistake or for convenience) more permissions than those actually necessary, leading to the so called Permission Overdeclaration [15]. This happens because some permissions have similar names and their description is not self-explicative for some developers. It is quite intuitive that users, seeing a very long

\(^2\)http://developer.android.com/reference/android/Manifest.permission.html
permissions list, are less encouraged to read and understand them.

3.2 Analytic Hierarchy Process

The Analytic Hierarchy Process (AHP) [25] is a multi-criteria decision making technique, which has been largely used in several fields of study [26], including, e.g., work in [27], in which AHP has been applied to multi-factor reputation systems, work in [28], [29], and [30], all of them presenting apps related to security policies and access control.

The AHP approach is the following: given a decision problem, where several different alternatives can be chosen to reach a goal, AHP returns the most relevant alternative with respect to a set of previously established criteria. The decision problem is structured as a hierarchy, by linking goals and alternatives through the chosen criteria (Figure 3). AHP subdivides a complex problem into a set of sub-problems, equal in number to the chosen criteria. For each criteria, a local solution is computed. Then, the most relevant alternative, i.e., the best solution for the decision problem, is computed by properly merging the local solutions.

Differently from classical computational intelligence or statistic techniques used for classification, the AHP decision process does not require a training phase, because decision parameters are assessed directly by experts of the decision problem. The value of these parameters are based on both objective and subjective interpretation of the problem elements. For example, let us consider a decision problem in which one has to decide
Table 1: Fundamental Scale for AHP

<table>
<thead>
<tr>
<th>Intensity</th>
<th>Definition</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Equal</td>
<td>Two elements contribute equally to the objective</td>
</tr>
<tr>
<td>3</td>
<td>Moderate</td>
<td>One element is slightly more relevant than another</td>
</tr>
<tr>
<td>5</td>
<td>Strong</td>
<td>One element is strongly more relevant over another</td>
</tr>
<tr>
<td>7</td>
<td>Very strong</td>
<td>One element is very strongly more relevant over another</td>
</tr>
<tr>
<td>9</td>
<td>Extreme</td>
<td>One element is extremely more relevant over another</td>
</tr>
</tbody>
</table>

which car is the best to buy. An example of an objective criterion is the “price”. Instead, if we consider the “aesthetic” criterion, its value is the output of a subjective evaluation. In general, in AHP problems it is up to some experts to assess the relevance of each criterion and the resulting assessment is generally subjective.

![Generic AHP Hierarchy](image)

Figure 3: Generic AHP Hierarchy

Pairwise Comparison Matrices Local solutions for each criterion are computed by means of comparison matrices, which describe how much an alternative is more or less relevant with respect to another one in a pairwise fashion. For each criterion, the relevance of each alternative with respect to others is expressed in a matricial form. A pairwise comparisons matrix $M$ is a square matrix $n \times n$ (where $n$ is the number of alternatives), which has positive entries and it is reciprocal, i.e., for each element $a_{ij}$, $a_{ij} = \frac{1}{a_{ji}}$, where $a_{ij} \in \{1, ..., 9\}$ (see Table 1).

The concept of consistency is defined for comparison matrices. A comparison matrix of size $n \times n$ is consistent if $a_{i,j} \cdot a_{j,k} = a_{i,k}$, $\forall (i, j, k)$. If a comparison matrix is consistent, the pairwise comparisons are well related between them. However, it is difficult to obtain perfectly consistent matrices using empirically defined comparisons. AHP requires that comparison matrices are, at least, semi-consistent. To measure the consistency of a comparison matrix, the consistency index $CI = \frac{\lambda_{\text{max}} - n}{n-1}$ has been defined [31], with $\lambda_{\text{max}}$ being the largest matrix eigenvalue. For a consistent matrix, we have that $CI = 0$, whilst a matrix is considered semi-consistent if $CI < 0.1$. If this condition does not hold, the comparison matrix should be re-evaluated.

Computing Local Priorities Local priorities express the relevance of the alternatives for a specific criterion. Given a comparison matrix,
local priorities are computed as the normalized eigenvector associated with the maximum eigenvalue of the matrix \([31]\). Thus, for each criterion \(c_j\), AHP extracts from the comparison matrix a vector \(p_{c_j}\) of size \(n\) expressing the relevance in percentage of each alternative for that criterion.

**Computing Global Priorities** The relevance of a criterion with respect to the goal is described by means of an additional pairwise comparisons matrix \(P_g\) of size \(k \times k\), where \(k\) is the number of criteria.

Global priorities are computed through a weighted sum of all the local priorities computed over the whole hierarchy (from alternatives to goal):

\[
P_{a_i}^{\text{g}} = \sum_{j=1}^{k} p_{g}^{c_j} \cdot p_{a_i}^{c_j}
\]

(1)

where \(P_{a_i}^{\text{g}}\) is the global priority of the alternative \(a_i\), \(p_{g}^{c_j}\) is the local priority of criterion \(c_j\) extracted from \(P_g\) with respect to the goal and \(p_{a_i}^{c_j}\) is the local priority of alternative \(a_i\) with respect to criterion \(c_j\).

\section{MAETROID}

In this section, we describe in detail the MAETROID framework. First, we present a permission classification method, in terms of the amount and the type of the declared permissions to evaluate the potential threat represented by an app. Then, we describe the specific and customized instantiation of AHP to assess the apps’ trustworthiness.

\subsection{Threat Score}

Each Android permission regulates the access to a specific resource or operation on an Android device. Generally, the more permissions are declared by an app, the greater is its potential threat, since more security critical operations are granted to the app. However, the number of permissions should not be the only parameter to assess the global app threat level. In fact, some permissions are related to operations or resources much more critical than others. For example, the permission to send text messages represents a different threat from the permission to control the smartphone vibration. For this reason, we propose a novel way to compute the threat score, i.e. the level of risk, of an app based upon the number and typologies of the declared permissions. MAETROID performs a static analysis of declared permission on new apps to compute a threat score representing the dangerousness of the app. This score is represented through a number ranging over the interval \([0, 15]\), where 0 represents an app that requires no permissions, or only harmless permissions, whilst 15 represents the worst case of an app that requires all the Android permissions, or only dangerous one. The value is proportional to the dangerousness of the requested permissions, as will be detailed later.

To properly compute the threat score, we have manually analyzed all the permissions defined by Android. This operation has been done to assess what resources and operations can be accessed through each permis-
sion and what is the effect of the misuse of such resource. In particular, each permission has been scored according to the level of threat represented against three security parameters, namely: privacy threat (threat to the confidentiality of the user), system threat (threat for the system integrity), and financial threat (threat for user’s mobile credit). Finally, we have created a table which assigns to each permission a score for each of the three threats. The score is in the range $[0, 1]$ according to the six levels of severity defined in Table 2.

<table>
<thead>
<tr>
<th>Table 2: Threat Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0.2</td>
</tr>
<tr>
<td>0.4</td>
</tr>
<tr>
<td>0.6</td>
</tr>
<tr>
<td>0.8</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

To explain the rationale behind this permissions analysis, let us consider the permission `android.permission.CALL_PHONE` as an example. We have assigned a score of 0.6 to the corresponding privacy threat, since this permission allows an app to perform phone call without the user being aware of it. For example, this permission enables an app to listen and remotely record the user activities and nearby sounds to infer her behavior. We have further assigned a score of 0.2 to the system threat, since the phone drains the battery faster during a call but it is unlikely that an attacker can exploit phone calls to attack the device integrity by reducing the battery lifetime. Finally, we have assigned a score of 1 to the financial threat, since an app with this permission can call any phone number included premium-rate numbers. Hence, a malicious app can cause a consistent financial damage to the user, e.g. by issuing calls to premium numbers which are likely to pass unnoticed until the user credit ends or the user receives the phone bill.

Table 15 in Appendix B gives an excerpt of the threat score we have assigned to all the Android permissions (for the full list please refer to [32]), where acronyms PT, ST, and FT are an abbreviation of privacy, system, and financial threat, respectively. The threat values have been given according to the documentation associated with each permission, which gives details on how permissions can be exploited by an app. The rationale behind the assignment of threat values to permissions is detailed in the following.

**Privacy Threat.** According to [33], personal data, i.e., any information referring to an identified, or identifiable, individual (i.e., the data subject), are subject to principles aiming to guarantee and preserve their privacy. As an example, the Use Limitation Principle states that “Personal data should not be disclosed, made available or otherwise used for purposes other than those specified at the time of data
collection except: i) with the consent of the data subject, or ii) by law.”

Permissions that refer to actions that may compromise or shatter these principles, such as permissions about access to users’ contacts, files, Internet bookmarks and chronology, or SIM and device information, such as the IMEI and IMSI codes, have received a high value for this index. In this cases, such permissions are considered risky from the privacy point of view. In particular, based on the principles highlighted in [33], we have given the highest privacy threat value to the following permissions:

- **android.permission.READ_CONTACTS**: access the contact list on the device. This may create a serious privacy leakage since an app can send the read data to an attacker through Internet or text message.
- **android.permission.READ_PROFILE**: access information of the user account. User’s account may contain private data like birth-date, location and occupation and apps with this permission can read, store and eventually send it outside of the user device.
- **android.permission.READ_SMS**: access the text message Inbox. This may constitute a serious privacy leakage, since the app is able to read both the text and the sender number of all private text messages stored in the device.
- **android.permission.RECEIVE_SMS**: allows the app to control and handle the event of incoming text message. Similar privacy risk of the READ_SMS permission, but the app can only intercept incoming messages and is not able to access the history. However, the app with such a permission may even decide not to show any notification of the received message to the user.
- **android.permission.RECEIVE_MMS**: allows the app to control and handle the event of incoming multimedia message. Same considerations for the former permission but applied to multimedia messages.

Moreover, a medium-to-low value has been assigned to those permissions that access sensors, such as camera or microphone, since they could be potentially exploited to spy the user behavior. All the remaining permissions, which neither access sensitive resources nor use a resource at all, have been given a low or zero value.

**System Threat.** A high value of system threat is assigned to permissions accessing system components and that if misused can cause integrity issues to the OS, to personal files, or even the physical device. A list of permissions which are critical on the system threat is the following:

- **android.permission.INSTALL_PACKAGES**: allows the app to install new packages. This functionality has been used by several malware to install later other dangerous apps with additional permissions (e.g. the ZFT malware) or advertisement apps (Adware).
• **android.permission.WRITE_EXTERNAL_STORAGE**: allows the app to modify the external memory content. An app with this permission can fill the device memory and remove or permanently modify files of the user of other apps. As an example, the malware *Moghava* permanently damages all the pictures in the user gallery overlapping on them a propaganda image.

• **android.permission.CHANGE_WIFI_STATE**: gives to an app the control on the WiFi device status. The WiFi interface has a consistent impact on the device battery lifetime and also may cause Internet disconnection, since WiFi overrides the mobile data connection even if the access point is not connected to the Internet. Thus, a malicious control on the WiFi represents an integrity violation.

Other permissions with a consistent value of system threat are all those permissions that give access to device interface and peripherals (e.g. camera, vibration, etc.) whose (mis)use cause both a performance or battery overhead.

**Financial threat.** High values of this index are assigned to permissions related to the usage of services that imply a financial cost, such as phone calls or outgoing SMSs. Conversely, if the cost is indirectly related to a specific permission, it receives a medium financial threat value. Some permissions that we consider critical for financial threat are the following:

• **android.permission.SEND_SMS**: allows an app to send text messages. With this permission an app can virtually send as many messages desires to whatever number. Sending text message is an operation which has a monetary cost established by the provider and that may vary with the recipient. Moreover text messages can be used for subscription to premium services which impose a weekly or monthly cost. Such a strategy has been exploited by several malware known as SMS Trojan [34].

• **android.permission.CALL_PHONE**: gives to an app the authorization to initiate phone calls. Phone calls have the same financial implications of text messages, with usually a higher cost which can be imposed on the user [35].

• **android.permission.INTERNET**: gives to an app the authorization to open sockets for external connections. Bytes of data received and transmitted is another element that telephony providers charge to users. Opening a connection and streaming data on it always generates a cost and an app with this permission can virtually send any amount of data. This permission becomes particularly dangerous if coupled with the **CHANGE_WIFI_STATE** permission discussed formerly, whose financial threat is in fact considered moderate (0.6).

An additional detailed example of threat score assignment, which consider the privacy, system and financial threat is given in Table 3 for the **SEND_SMS** permission and discussed in the following.
**SEND_SMS Permission** The **SEND_SMS** permission enables an app to send SMS messages, also without requiring user confirmation. Hence, an app that declares this permission can send SMS messages, with any text, at the current rate, and at any phone number, without the user noting it (unless the user checks periodically the available credit). This permission has been exploited by several malware to leak the user credit by sending messages to premium-rate numbers, or to threaten her privacy by sending information, such as the IMEI and IMSI codes, to a phone number controlled by the attacker [34]. Therefore, we set its financial threat to “High”. The financial threat is considered high since sending SMS text messages has a cost and they can be used to perform subscriptions to premium services. The privacy threat is considered medium-high since SMS messages can be used as a vector to steal sensitive information. However, this information has to be accessed before it can be sent and this requires other specific permissions. Finally, sending text messages does not represent a threat to the system itself. Hence, the system threat is set to zero. It is worth noting that the threat levels have been assigned only to the permissions defined by Google.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Privacy Threat</th>
<th>System Threat</th>
<th>Financial Threat</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEND_SMS</td>
<td>0.8</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

### 4.1.1 Global Threat Score

For each app \( \alpha \), we define the global threat score \( \sigma \), which summarizes in a single index the threats of all the requested permissions declared. This is done by analyzing the manifest file, and by calculating a weighted sum as follows:

\[
\sigma = \frac{\sum_{i=1}^{n} w_p \cdot pt_i + w_s \cdot st_i + w_f \cdot ft_i}{1 + \lceil \log(1 + n) \rceil}
\]  

(2)

where \( n \) is the number of permissions declared by a specific app, \( pt_i, st_i, ft_i \) are, respectively, the privacy, system, and financial threat of the \( i \)-th permission required by the app, and \( w_p, w_s, w_f \) are the corresponding weights. In the current implementation, we consider \( w_f \) being three times greater than \( w_s \) and \( w_p \): this means that we consider the financial threat more relevant than the system and privacy threats, since it can harm the user with more impact. The denominator of (2) has been added so that the dangerousness of the permission is considered more relevant than the number of permission. We consider apps with \( \sigma \) lower than 4 as low-threat apps, while ones with \( \sigma \) in the interval \([1, 4] \) are moderate threat to high-threat. Higher values of \( \sigma \) mean extremely critical apps.

The value \( \sigma \) estimates how much an app is critical from the security point of view, by considering the declared permissions only. Hence, the
more permissions are required by an app, and the more dangerous these permissions are, the more critical the app becomes. The idea is that if an app receives a low-threat score, this should increase the likelihood that this app is downloaded and, as a consequence, this should encourage developers to accurately choose the permissions required by their apps. However, several apps actually require a large number of permissions to perform all their functions, especially communication and social apps, and they should not be considered as suspicious. This leads us to rely on a multi-criteria decision system (presented in Section 3.2) in order to classify an app with respect to a set of criteria, among which the threat score $\sigma$.

4.2 Classification Problem Instantiation

Since the global threat score $\sigma$, computed from the requested permissions, should not be the only parameter used to assess the trustworthiness of the app, we instantiate the AHP decision methodology to include further criteria. In detail, an Android app is described by the following parameters: a threat score $\sigma$, a developer $\delta$, a number of download $\eta$, a market $\mu$, and a user-rating $\rho$. The goal consists in assigning the app one of the following alternative labels:

**Trusted.** The app works correctly and does not hide malicious functionalities. A trusted app is characterized by a low threat score, i.e. it is considered not to be able to harm the system due to the low threat of the required permissions. Moreover, a trusted app generally comes from the official market, downloaded by thousands of users, having very good reviews and/or developed by a developer with outstanding reputation (i.e. top developer). All the aforementioned features shape an app which is both secure and appreciated by the users. Therefore, the user can safely install such an app. MAETROID will show a green “happy” smiley when installing a trusted app.

**Medium-Risk.** The app does not work correctly and includes unwanted functionalities. An app is considered to represent a Medium-Risk to the device security when, even if it shows an acceptable (low) threat score, it has received poor reviews, or has been downloaded by too few users (less than 100) to infer that the app does not hide threats. Generally this decision is given to low quality apps published on official or unofficial market by non-skilled developers. Another reason is that the app is likely to be unwanted from the user, such as an Adware, who should rethink about installing it. A yellow smiley with a neutral expression (“poker face”) is shown to the user when installing a Medium-Risk app.

**High-Risk.** The app likely includes malicious code. This decision is given to apps that require several dangerous permissions, representing a potential threat to the device and its user. In fact, the greatest majority of malware (95% [3]) asks for several dangerous permissions related to text messages, which reasonably should not be asked by any app which is not related to instant messaging. As discussed in the following, apps that genuinely ask for several dangerous permissions are recognized by MAETROID thanks to the positive user
reviews combined with a conspicuous number of downloads (more than 100,000).

Figure 4 gives a graphic description of the AHP instantiation of the MAETROID classification problem. The variables $\sigma$, $\delta$, $\eta$, $\mu$, $\rho$ represent the problem criteria, but, differently from classic AHP instantiation, for each criterion more comparison matrices are defined. A specific comparison matrix is chosen for each criterion, based on its variable value. Hence, for two different apps, the AHP classification problem is instantiated with different comparison matrices\(^3\), representing however the same criteria.

![Figure 4: AHP Instantiation of the MAETROID Classification Problem.](image)

In the following we detail the building and choice process for AHP comparison matrices used by MAETROID and their relation to the possible values of each of the criteria.

### 4.2.1 Criteria

We have defined five criteria, namely $\mu$ for the market, $\delta$ for the developer, $\rho$ for the user rating, $\eta$ for the number of downloads, and $\sigma$ for the threat score. They are detailed in the following.

**Market ($\mu$).** Apps are normally distributed through online marketplaces. The most popular market is Google Play, also referred as the official market. This market is considered as a more protected environment since app developers build their reputation on the base of the apps they publish. More specifically, a developer who wants to publish apps on Google Play has to buy a developer account, at the price of 25$. In exchange, the developer receives a private key which can use to digitally sign her apps [36]. If users report an app as malicious, then this app is removed both from the market and remotely from all the devices that have installed it. Moreover, the developer can be tracked and blacklisted. In addition, Google Play includes some reputation indexes that should help users to understand the app quality. These features make the official market a trustworthy place where to download apps. Nevertheless, several malware have also been found in the official market [3] [37] [38].

There also exists a plethora of unofficial marketplaces, among which:

- [http://www.appbrain.com](http://www.appbrain.com)\(^3\)

---

\(^3\)The whole set of comparison matrices is presented in [32].
These marketplaces do not require developer registration, however still attracts several users since these markets usually give access to apps that are not available on the market or distribute free versions of apps that on the Play store. However, unofficial markets often miss reputation indexes and sometimes there is no control on the quality of the apps, so it is easier to download malicious apps. Related to the market source, in the problem instantiation we also consider another parameter, namely app that are manually installed, which happens when the user manually installs the app, e.g. when downloading the app apk and installing it through a file manager.

With reference to Table 1, we show the relevance of each alternative, for the three possible values of $\mu$:

- $\mu =\text{official}$: we consider that Trusted is moderately more relevant than Medium-Risk and strongly more relevant than High-Risk;
- $\mu =\text{unofficial}$: we consider that High-risk is moderately more relevant than Trusted and slightly more relevant than Medium-Risk;
- $\mu =\text{manually installed}$: we consider that High-Risk is slightly more relevant than Trusted and Medium-Risk (that are equally relevant).

According to this information, comparison matrices are directly computed.

Developer ($\delta$). We consider three types of developers: Standard, Top, and Google. Google rewards the best app developers with a Top Developer badge, which is reported on each app they publish. Hence, these developers should be considered strongly trusted since they produce high-quality apps and should not be interested in lowering their reputation. On Google Play, Google Inc. itself is considered a Top Developer; however, we consider Google more trusted than other developers, given the interest that Google has in the well being of Android users.

All the other developers are considered standard and since the Top Developer badge is only used on Google Play, all developers of apps coming from unofficial markets have been labelled standard as well. We make this assumption because the bound between app and developer is not ensured on unofficial markets.

The comparison matrix for the developer parameter is defined according to the following analysis:
• $\delta = \text{Google}$: we consider that Trusted is extremely more relevant than Medium-Risk and High-Risk (that are equally relevant);
• $\delta = \text{Top Developer}$: we consider that Trusted is very strongly more relevant than High-Risk and Medium-Risk (that are equally relevant);
• $\delta = \text{Standard}$: we consider that the three alternatives are equally relevant.

User Rating ($\rho$). On several markets, users can rate apps and leave a comment, which can be shown to other users. Rating is generally expressed as a number that ranges from 1 to 5 (or it is normalized in this range). We consider apps with a rate less than 2 as low-quality, for which the Medium-Risk alternative is extremely more relevant than the Trusted one. A score higher than 4 means a high-to-very-high quality apps for which the Trusted alternative is very strongly more relevant than the other two. Intermediate values mean a neutral comparisons matrix.

Number of Downloads ($\eta$). Several markets report the number of downloads for each app. As an example, the so-called “killer apps”, i.e., extremely popular apps, have been downloaded from Google Play more than 100 millions of times. These apps should be considered differently from those downloaded a lower number of times, e.g., less than 100 times. In fact, apps with a very high amount of downloads are popular apps already tested by several users and, usually, more trustworthy. Notice that the number of downloads, though independent, is needed to contextualize the User Rating criterion. In fact, a rating of five stars (out of five), given to an app downloaded by a single user, is practically meaningless. Hence, we define 7 intervals in which the value $\eta$ may fall. For very high values of $\eta$, Trusted is extremely relevant. As the value of $\eta$ decreases, the relevance gradually turns from Trusted to High-Risk.

Threat Score ($\sigma$). For each app, the threat score is computed as explained in Section 4.1. We define the following intervals:
• $\sigma < 4$: trusted is very strongly more relevant than High-Risk and moderately more relevant than Medium-Risk;
• $4 \leq \sigma \leq 7$: High-risk is very strongly more relevant than the other alternatives (that are equally relevant);
• $\sigma > 7$: High-Risk is extremely more relevant than Trusted, and Medium-Risk is strongly more relevant than Trusted.

For marketplaces without download counters and/or rating systems, we define two additional comparison matrices whose elements are all equal to 1. When using these matrices to describe a criterion, all the alternatives have the same relevance for that criterion. Hence, this criterion will not influence the decision. We have defined 20 comparison matrices, but it is possible to increase their number to have finer, or customized, granularity for each criterion. Finally, it is worth noting that the list of proposed criteria is not exhaustive, and the methodology enable the insertion of other rules to evaluate the alternatives. In the current implementation, we consider all the criteria as equally relevant.
The classification process of MAETROID is depicted in Figure 5, which shows that apps belong to one of the three classes of the left-hand side of the picture (safe, unwanted behavior, malware) and are classified using one of the three indexes of the right-hand side (Trusted, Medium-Risk, High-Risk). The parameters used to perform the classification are the market reputation, developer reputation, threat score (computed from the permission list), download number and user rating.

![Figure 5: MAETROID Classification Process.](image)

## 5 A Prototype Implementation of MAETROID

The MAETROID framework has been implemented as an app for Android devices. The MAETROID app is composed of an activity\(^4\) and several services\(^5\) running in background. Whenever a new app is being installed, MAETROID intercepts the event broadcasted by Android through an intent filter\(^6\). Hence, the MAETROID app comes in foreground with its activity, showing to the user that the app is being analyzed. In the meanwhile, analysis services retrieve the values of the five criteria (market, developer reputation, user rating, number of downloads, threat score) for the app being analyzed. The threat score is computed by parsing the app manifest file, to retrieve the set of requested permissions, and by computing the global threat score as shown in Section 4.1.1. Note that the evaluation is performed locally on user-devices whenever a new app is going to be installed. Hence, there are not scalability issues on the market, since the market is not affected by the computation or by the threat values.

The market is inferred from the installer of the downloaded app. In fact, as discussed in Section 4, both official and unofficial markets provide an on-device custom app called “installer”, which is used to browse the marketplace, download, and install some selected apps. The name of the

\(^4\)http://developer.android.com/reference/android/app/Activity.html
\(^5\)http://developer.android.com/guide/components/services.html
\(^6\)http://developer.android.com/guide/components/intents-filters.html
installer is reported in the message which is broadcasted by the OS to communicate the event of a new app installed on the device. The other criteria, namely user rating, number of downloads, and developer reputation are extracted by parsing the HTML code of the market web page. Upon computing the values for the five criteria, MAETROID implements AHP through Jama, the Java matrix package for matrix calculi. 

Upon completing the analysis (left screenshot in Figure 6), MAETROID returns its decision in the form of a smiley (Figure 6, second, third and fourth screenshots). We have decided to use a simple output decision in the form of a smiley to make it more user-friendly and understandable also by ordinary users. If the app is considered High-Risk or Medium-Risk, 

![Some Screenshots of the MAETROID App.](image)

Figure 6: Some Screenshots of the MAETROID App.

Upon completing the analysis (left screenshot in Figure 6), MAETROID returns its decision in the form of a smiley (Figure 6, second, third and fourth screenshots). We have decided to use a simple output decision in the form of a smiley to make it more user-friendly and understandable also by ordinary users. If the app is considered High-Risk or Medium-Risk,
the user is advised of the potential threat (through a red 'sad' smiley or a yellow 'poker face' smiley) and asked if she wants to uninstall such an app. If the user decides to uninstall the app, MAETROID handles the uninstallation process. Otherwise, if MAETROID considers the app as trusted, the user is invited (through a green 'happy' smiley) to run the installed app. It is worth noting that MAETROID does not block the installation process, but prevents the app from being started until the analysis outcome is shown and the user has taken her decision. The fact that the installation process is not blocked a priori by MAETROID is not harmful. In fact, in Android an app does not perform any operation (including deploying assets and saving files on the device) until it is run. Given that users follow the MAETROID advise, a dangerous app can neither harm the user nor the device, since apps can be opened only after that users trigger the app start from the launcher app.

It is worth noting that the app classification is performed directly on the device by the MAETROID app. This approach has the following advantages:

- MAETROID is not affected by scalability issues, since it is not necessary to classify “a priori” all the existing Android apps, building a centralized database which would need continuous update and maintenance;
- app updates automatically trigger a reclassification process as soon as the updated version is downloaded. Thus, if the classification result changes because the updated version asks for more permissions, the new version will not be run on the device, given that the user follows the MAETROID decision.

The sequence of steps of the MAETROID analysis process are depicted in Figure 7.

**Step 1:** a new app is downloaded locally from the marketplace;

**Step 2:** the user decides to perform the app installation;

**Step 3:** the installation process is hijacked (and paused) by the MAETROID app (which is installed on the user device);

**Step 4:** MAETROID retrieves the metadata used to perform the classification, locally from the app manifest file and remotely from the marketplace;

**Step 5:** MAETROID exploits the retrieved metadata to apply the AHP classification locally on the device;

**Step 6:** the decision is shown to the user, in form of a smiley;

**Step 7:** the user decides whether to continue the installation or remove the app, based on the output of the classification.

### 5.1 Classification Results

To test the ability of MAETROID in detecting potentially malicious Android apps and classifying the risk of Android apps, we have conducted two set of experiments, one on a large set of applications coming from
known databases and one on a smaller set of apps manually analyzed. In the first set we have used the classification algorithm of MAETROID to classify a dataset of 11,046 apps. This dataset is composed of 9,804 apps selected from the official market Google Play, while the remaining 1,242 apps come from the database of known malware Genome[8]. To extract the meta-data of the Google Play apps, we have built a crawler to retrieve the rating, download number and permissions set starting from an existing database.

The classification results are shown in Fig. 8 and also reported in Table 4 for the sake of clarity. As shown, malicious apps of the Genome database have all been classified as risky from MAETROID. More precisely 85% of the malicious apps of Genome have been classified as High-Risk and the remaining 15% as Medium-Risk. None of the apps from Genome have been classified as Trusted. We have used the apps from Google Play as a control set. We can see that the greatest share of Play apps (77.37%) have been classified as Trusted, while 22.4% have been classified as Medium-Risk and only 0.23% with High-Risk. It is worth noting that in these tests there is no knowledge beforehand whether the apps coming from Google Play are really secure or infected by malware, but the classification on the control set is plausible. In fact, the classification results show that more than 75% of the apps from Google Play do not represent a threat to security, while about 22% of the apps show some criticalities, usually due to a low number or download, and only a very small number of apps represent a potential threat to security, mainly due to the set of dangerous permission they ask. Additional details on this set of experiments, i.e. links to the tested apps and their metadata, can be retrieved at http://www.doc.ic.ac.uk/~dsgandur/maetroid/app_list_final.xlsx.

In the second set of experiments we have verified the classification

---

8https://github.com/MarcelloLins/GooglePlayAppsCrawler
accuracy of MAETROID, by measuring both its precision and recall (i.e., the overall classification error). The testbed dataset does not overlap with the previous one and is composed of 180 Android apps, which are known in advance to be:

- **Safe Apps**: those apps that behave correctly both from the security and functional point of view. This class is further divided in two subclasses: **Official** and **Unofficial**, stating, respectively, whether the app has been downloaded from Google Play or not. Good Apps are correctly classified by MAETROID if its output is “trusted” (green “happy” smiley);

- **Apps with Unwanted Behavior**: the app permissions given to these apps may be used to cause potentially unwanted behavior, such as with Adware. These apps are correctly classified by MAETROID if its output is “Medium-Risk” (yellow “poker face” smiley);

- **Malicious Apps**: those apps infected by a malware. Malicious Apps are correctly classified by MAETROID if its output is “High-Risk” (red “sad” smiley).

In more details, the test-set consists of 180 manually vetted apps, of which 90 come from Google Play, 50 from unofficial markets, and 40 are downloaded from Websites that are different from marketplaces (these apps are denoted hereafter with “manually installed”). Among all these 180 apps, 40 are infected by well-known malware. Apps belong to different categories: augmented reality, books and news, communication, desktop manager, entertainment, file managers, game, social and utility, and antivirus. The app user rating ranges over $[1, 5]$, the number of downloads ranges over $[0, 10M+]$, and the apps were produced either by standard developers, or by Top Developers, or by Google.

The MAETROID outcome over the test-set is reported in Figure 9,
where the x-axis shows the three possible outcomes of MAETROID (Trusted, High-Risk, Medium-Risk), while the y-axis shows the number of apps classified per outcome. The light-green color represents safe apps (in dark green the ones coming from unofficial markets), the red color represents apps infected with malware, whilst violet (vertical lines pattern) represents apps with unwanted behavior. All the infected apps have been correctly recognized by AHP as High-Risk. It is worth noting that some good apps also fall in this class. These apps come from unofficial markets (labelled as “Good Apps (Unoff.)” in Figure 9). Since no user rating is available for these apps, MAETROID applies a safe approach by considering them as High-Risk, at least initially. However, as soon as new information become available for these apps [39], they will eventually be classified as trusted ones. All the apps coming from Google Play have been classified either as trusted or Medium-Risk based upon the user rating, threat score, and number of downloads. All the apps with unwanted behavior coming from Google Play have been correctly considered as Medium-Risk. These apps do not work as expected or they crash upon starting. An example of this class of apps is the game Avoid the Ghosts, which is a reproduction of the classic Pac Man game. The app does not work correctly: in fact, when the app starts, it is impossible to control the Pac Man movements. The app has been found on the official market, but it has been downloaded few times and it received bad ratings. However, Avoid the Ghosts does not require dangerous permissions and, hence, it is considered Medium-Risk by MAETROID rather than High-Risk.

Figure 9: Classification Results on Validation Set

To better understand the functionalities of MAETROID, in the following we show the classification process for two popular apps, namely Angry Birds Space and Skype.

Classification Example 1: Angry Birds Space The values of the five criteria in input to MAETROID are shown in Table 5. The app developer is a Top Developer. The app has been downloaded by more

---

9The app was available on Google Play at time of experimentation, while at time of writing it was not available anymore.
than 10 millions of users, receiving a global rating of 4.7. Furthermore, it comes from the official market Google Play and it has a low threat score (2.7).

Table 5: Parameters of Angry Birds Space

<table>
<thead>
<tr>
<th>σ</th>
<th>ρ</th>
<th>µ</th>
<th>δ</th>
<th>η</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7</td>
<td>4.7</td>
<td>Google Play</td>
<td>Top Developer</td>
<td>10M+</td>
</tr>
</tbody>
</table>

Table 6 shows the matrix used to compare the three alternatives with respect to the “App Developer” criterion for this app. Top Developers generally produce high quality apps and they are not likely to publish malicious apps. Following this intuition, we assigned the following pairwise relevances to the alternatives: trusted is very strongly favorite with respect to Medium-Risk and strongly favorite with respect to High-Risk. Trusted (green “happy” smiley) obtains the highest priority (0.7) compared to the other two alternatives.

Table 6: Comparison Matrix for Top Developer for Angry Birds Space

<table>
<thead>
<tr>
<th>Top Developer</th>
<th>Trusted</th>
<th>High-Risk</th>
<th>Medium-Risk</th>
<th>Local Priorities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trusted</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>0.7</td>
</tr>
<tr>
<td>High-Risk</td>
<td>1/4</td>
<td>1</td>
<td>4</td>
<td>0.23</td>
</tr>
<tr>
<td>Medium-Risk</td>
<td>1/7</td>
<td>1/4</td>
<td>1</td>
<td>0.07</td>
</tr>
</tbody>
</table>

Using Equation (1), MAETROID merges the local priorities for criterion “developer” with the ones coming from the comparison matrices of the other four criteria, to finally obtain the global priorities [0.7, 0.16, 0.14]. The three values represent the priorities for the three alternatives, respectively Trusted, High-Risk, and Medium-Risk. Trusted is the alternative with the highest value and, thus, is also the result of the MAETROID classification for that app.

On the contrary, when MAETROID analyzes a version of Angry Birds Space found on a database of infected apps it outputs High-Risk as the highest priority. This app has been found in the past to be infected by the malware Geinimi [34]. The malware steals information concerning both the user and the device, which are sent via SMS to a number controlled by the attacker. To perform these further operations, the malware asks for several other permissions (Figure 10), leading to a threat score equal to 7.3. This high value for the threat score correctly drives AHP towards the High-Risk outcome.

Classification Example 2: Skype Skype is a popular software used for VoIP and free chat and its mobile version has earned a positive outcome, since it enables phone calls with smartphones, using the data
connection instead of traditional (and more expensive) landline and cellular calls connections. To work properly, the Android version of Skype requires a large number of permissions. Computing the global threat score by means of Equation (2), Skype gets a score of 6.8. Skype is an example of a high-threat app. In our analysis, we have considered two Skype versions, one from the official market the other one from an unofficial market, as reported in Table 7.

Table 7: Two Skype Versions

<table>
<thead>
<tr>
<th>Name</th>
<th>$\sigma$</th>
<th>$\rho$</th>
<th>$\mu$</th>
<th>$\delta$</th>
<th>$\eta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skype 1</td>
<td>6.8</td>
<td>3.8</td>
<td>Google Play</td>
<td>Standard Developer</td>
<td>10M+</td>
</tr>
<tr>
<td>Skype 2</td>
<td>6.8</td>
<td>4</td>
<td>Unofficial</td>
<td>Standard Developer</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

MAETROID computes the global priorities vector $[0.47, 0.4, 0.13]$ on the Skype version coming from the official market. These results slightly favor the trusted alternative (green “happy” smiley). Both the marketplace and the large number of downloads increase the trustworthiness of this app, even if it has a high threat score. For the Skype version downloaded from the unofficial market, which does not even provide a download counter, the global priorities are very different from the previous ones: $[0.29, 0.52, 0.19]$ and the app is labeled as High-Risk (red “sad” smiley). Even if the two versions require the same set of permissions, it is possible that their source codes are different (possibly malicious). Since
more than 10 millions of users have downloaded the version from the official market, it is strongly unlikely that malicious behaviors have not been noticed and reported, forcing the app removal from the market.

6 Evaluation of MAETROID Effectiveness: User Expectation and Acceptance

In this section, we discuss the effectiveness of the MAETROID approach in conveying its decision, by analyzing the users’ response to the trustworthiness index. In particular, we have created a survey with a list of questions related to the security of Android apps. The survey also contains questions to evaluate the usefulness of MAETROID, i.e., how the MAETROID outcome influences the user’s decision (whether to install an app or not). The structure of the survey is available both in A and online at http://icaremobile.iit.cnr.it/survey/mobilesecuritysurvey.htm.

The survey consists of twelve multiple-choice questions which are understandable by average mobile device users. The survey has been made available both online and physically, at a public event about Internet technology\(^\text{10}\). During the period of observation, in October 2013, we have collected 189 responses, by subjects with different age, background, and technical expertise. The subjects have not been formerly instructed about the survey content.

In the following, we present the survey structure and analyze the answers.

6.1 Subjects Set

The first four questions of the survey aim at assessing the variety of the sample. In Table 8, 9, 10, we report, respectively, the distribution of age, background, of the subjects. Most of the subjects, as shown in the Table 8, range over 18 to 45 years. One third of the subjects are students, 55% are workers, and 17% answer “other” or nothing.

<table>
<thead>
<tr>
<th>Age</th>
<th>% of Respondants</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 18</td>
<td>3.17%</td>
</tr>
<tr>
<td>18 – 25</td>
<td>23.28%</td>
</tr>
<tr>
<td>26 – 35</td>
<td>38.90%</td>
</tr>
<tr>
<td>36 – 45</td>
<td>19.04%</td>
</tr>
<tr>
<td>&gt; 45</td>
<td>15.87%</td>
</tr>
<tr>
<td>No answer</td>
<td>0.52%</td>
</tr>
</tbody>
</table>

Table 8: Age of Respondants.

We have also asked the users to specify their mobile OS. Figure. 11 shows the percentage of market share of mobile OSes among the subjects.

\(^\text{10}\)http://www.internetfestival.it/
Table 9: Gender of Respondants.

<table>
<thead>
<tr>
<th>Gender</th>
<th>% of Respondants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Female</td>
<td>32.2%</td>
</tr>
<tr>
<td>Male</td>
<td>66.6%</td>
</tr>
<tr>
<td>No answer</td>
<td>1.05%</td>
</tr>
</tbody>
</table>

Table 10: Occupation of Respondants.

<table>
<thead>
<tr>
<th>Occupation</th>
<th>% of Respondants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>29.10%</td>
</tr>
<tr>
<td>Worker</td>
<td>54.50%</td>
</tr>
<tr>
<td>Other</td>
<td>15.34%</td>
</tr>
<tr>
<td>No answer</td>
<td>0.52%</td>
</tr>
</tbody>
</table>

As shown, 60% of the subjects use an Android-based mobile device, while the 30% of them use iOS (other OSs are used by the remaining subjects). These shares are in line with statistical results by a recent market analysis in [1].

![Platform Shares]

Figure 11: Platform Shares

6.2 Security Understanding

Questions 5, 6, and 7 are related to subjects’ concerns on security threats on mobile devices. In particular, these questions aim at verifying how much the average user is aware of the existence of security threats against their mobile devices, e.g., coming from the installation of non secure apps. Also, the subjects are asked about existing security countermeasures (e.g., anti-virus) and their trust towards them.

Question 5 presents a list of security threats and asks the subject to select those threats that are perceived as the most dangerous. Subjects are allowed to choose from none to all of the presented options. The questions and percentage of subjects that selected the corresponding option is shown.
<table>
<thead>
<tr>
<th>Security Concerns</th>
<th>% of Respondants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smartphone theft</td>
<td>50.80%</td>
</tr>
<tr>
<td>Identity theft (e.g., Facebook credentials)</td>
<td>33.33%</td>
</tr>
<tr>
<td>Credit card data theft</td>
<td>27.51%</td>
</tr>
<tr>
<td>Installing Malware (e.g., virus, etc)</td>
<td>35.98%</td>
</tr>
<tr>
<td>Attacks to privacy (location tracking)</td>
<td>44.44%</td>
</tr>
<tr>
<td>Other</td>
<td>9.52%</td>
</tr>
<tr>
<td>None</td>
<td>3.70%</td>
</tr>
</tbody>
</table>

Table 11: Security Concerns of Respondants.

in Table 11. As shown in the Table, about 50% of the subjects is concerned about the theft of their physical device and attacks to their private data, whilst 40% of the subjects are aware about the possibility of installing malicious software on their mobile devices. These results show that most of the subjects have basic understandings about security problems. It is worth noting that there is a relevant percentage of subjects (about 10%) that are not concerned about security at all. However, according to results, average users are worried about mobile security issues.

Question 6 asks the subjects what security mechanisms and practices they use to protect their mobile devices. Subjects are allowed to select any of the presented options. The questions and percentage of subjects that selected the corresponding option is shown in Table 12. As shown in the Table, most of the subjects protect their SIM card through a PIN. A large percentage of users also protect their phone and data through backups and OS update. Several subjects deactivate GPS when not used, but it is possible that most of them are driven by the high battery consumption, rather than by a privacy concern. About one third of users check the app permission list when installing the app. This shows that most of the users (i.e. 2/3) is not concerned at all about the possible risks due to apps asking too many permissions. As we have already discussed, this is because permission list is too difficult to understand for users. Hence, a simple and understandable index as the one proposed by MAETROID is needed. Finally, only a low percentage of subjects (15.87%) has an anti-virus installed on their devices.

Question 7 presents to subjects a list of security features and asks to select those that they would consider useful for their devices. The questions and percentage of subjects that selected the corresponding option is shown Table. 13. As shown in the Table, the most required feature is the anti-theft functionality, which is able to lock and/or locate a stolen device. This result is sound with the one of Question 5, stating that users are afraid of having their device physically stolen. Interestingly, the subjects have shown a keen interested in other two security features, namely a classifier of apps (39.15% of preferences), in terms of their hazardousness, and the availability of feedback on the app behavior (40.21% of preferences). We want to underline that MAETROID provides both these features.
Table 12: Adopted Security Features and Practices of Respondants.

<table>
<thead>
<tr>
<th>Adopted Security Feature</th>
<th>% of Respondants</th>
</tr>
</thead>
<tbody>
<tr>
<td>USIM PIN protection</td>
<td>56.08%</td>
</tr>
<tr>
<td>Lock screen with password/sequence/PIN</td>
<td>33.33%</td>
</tr>
<tr>
<td>Disable GPS when not used</td>
<td>44.33%</td>
</tr>
<tr>
<td>Constantly update OS and apps</td>
<td>41.27%</td>
</tr>
<tr>
<td>Check permissions required by apps at install time</td>
<td>34.39%</td>
</tr>
<tr>
<td>Disable WiFi connection in public places</td>
<td>10.05%</td>
</tr>
<tr>
<td>Use anti-virus or secure apps (e.g., to encrypt private data)</td>
<td>15.87%</td>
</tr>
<tr>
<td>Backup personal data</td>
<td>31.75%</td>
</tr>
<tr>
<td>Other</td>
<td>4.76%</td>
</tr>
<tr>
<td>None</td>
<td>10.58%</td>
</tr>
</tbody>
</table>

Table 13: Requested Security Features by Respondants.

<table>
<thead>
<tr>
<th>Requested Security Feature</th>
<th>% of Respondants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable applications</td>
<td>24.33%</td>
</tr>
<tr>
<td>SMS blocker</td>
<td>22.22%</td>
</tr>
<tr>
<td>Call blocker</td>
<td>20.11%</td>
</tr>
<tr>
<td>Anti-theft features (e.g., remotely block a stolen smartphone)</td>
<td>60.85%</td>
</tr>
<tr>
<td>Classification of an application hazardousness before installing it</td>
<td>39.15%</td>
</tr>
<tr>
<td>Do not install applications from unofficial markets</td>
<td>19.05%</td>
</tr>
<tr>
<td>Anti-virus</td>
<td>35.98%</td>
</tr>
<tr>
<td>Personal data encryption with a password</td>
<td>32.30%</td>
</tr>
<tr>
<td>Send of SMS to secure number in case of anomalous event</td>
<td>40.21%</td>
</tr>
<tr>
<td>Block adult content</td>
<td>8.47%</td>
</tr>
<tr>
<td>Hide GPS positions to some apps</td>
<td>35%</td>
</tr>
<tr>
<td>Secure/anonymous browser</td>
<td>31.75%</td>
</tr>
<tr>
<td>Data Backup</td>
<td>33.33%</td>
</tr>
<tr>
<td>Disable permissions given to apps</td>
<td>27.51%</td>
</tr>
<tr>
<td>Feedback on bad application behavior (leaking user money, battery depletion, etc)</td>
<td>40.21%</td>
</tr>
<tr>
<td>Other</td>
<td>0.53%</td>
</tr>
<tr>
<td>None</td>
<td>5.30%</td>
</tr>
</tbody>
</table>

With question 8, subjects are asked to specify the level of importance that they give to some parameters when evaluating the risk of installing an Android app. In this question, the subject is asked to choose a value of importance, ranging from 1 (very low importance) to 5 (very high importance), for each index. Namely, these indexes are:

- the number of permissions;
- the number of downloads;
- the app rating;
- the app popularity;
- and marketplace.
It is worth noting that these indexes reflect the criteria used in MAETROID to evaluate the trustworthiness of apps. This question has been inserted in the survey to evaluate how much people consider such criteria relevant for defining the app risk.

Figure 12 reports the perceived levels of importance for each index. The bars concerning the market index show that this criterion is perceived as highly important for a large number of subjects (30%). However, almost the same number of subjects (28%) consider the market as a criterion with a very low importance. Quite obviously, the latter are exposed to a high risk of downloading malicious apps. The user rating criterion is considered important by most of the subjects. In fact, only 20% of subjects give this criterion a limited importance (i.e., option “very low” or “low”). On the other hand, only 24% of subjects consider permissions as an important criterion. Instead, 25% of the subjects give no importance to permissions when deciding whether or not to install an app. Popularity and number of downloads are considered from moderately important (medium) to highly important by most of the subjects. Summarizing these results, on average, the importance given to the five criteria is comparable with the relevance we have assigned to the same criteria in the AHP instantiation presented in Section 4. The only exception is the user rating criterion that is considered slightly more important by the users, and the permissions index that is considered less important than the other criteria. Although the user rating criterion is an index perceived as very important by interviewed people, it has two major problems when adopted as the main evaluation criterion. Firstly, its meaning need to be coupled with the total number of ratings and with the aggregator metric used to compute the global rating [40]. For example, a single, very positive rating, which can also be assigned by the developer himself, does not imply a good quality of the app. A second shortcoming is that, as discussed in Section 4, several markets do not include user ratings. These results suggest that users have not a clear understanding of which parameters are more relevant when deciding whether to install or not an app based on the perceived risk. As such, we believe that MAETROID, which incorporates a more careful selection of weights for the indexes, help users in taking a more informed decision.

6.3 Selection of apps

In the last two questions, the survey shows a list of apps with different features, asking which apps the subjects would like to install on their (brand-new) mobile devices. For each app, the survey includes a screenshot of the market that shows the set of requested permissions and available criteria (the market name, the developer, and the user rating, when applicable). The apps included in the survey provide basic features to a mobile device. Each app has three further parameters:

- very popular or unpopular,
- free or premium versions,
- from official or unofficial markets.
The rationale of this question is to understand whether users prefer a free, even if unknown and, hence, potentially harmful app, versus a popular but not free one. The two apps Angry Birds Space Premium Unofficial Free and Ruzzle Unofficial Free are the trojanized versions of two popular games. These are infected by the trojan GEINIMI, which exploits text messages to register to premium services.

The question has two variants: the first one (question 11), without an index helping the users to make an informed decision, while the second one (question 12) also shows the MAETROID classification result represented by a coloured smiley (see Sect. 5). As shown in the results of Table 14, a consistent percentage of subjects choose the free (and Trojanized) version of the games, instead of the versions coming from the official market. This happens even if the permission list of such trojanized versions declares some anomalous permissions, like SEND_SMS, and a large subset of the interviewed users would install such bad apps (probably, this happens just because they are free). Comparing the second column of Table 14 with the first one, we notice that now less subjects are willing to install the trojanized apps, after knowing the MAETROID decision (10%). In fact, 90% of the subjects prefer not to install the trojanized apps after having seen the MAETROID classification result. It is worth also noting that users prefer not to install that app even from the official markets.

7 Discussion

MAETROID is an app that helps users understand the risk level of downloaded apps, i.e. if they have potential security and privacy risks. Several factors contribute to make an app likely dangerous. The most relevant
Table 14: Subjects Willing to Install Apps (Questions 11 and 12).

<table>
<thead>
<tr>
<th>App</th>
<th>% of Respondants Before MAETROID Decision</th>
<th>% of Respondants After MAETROID Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whatsapp (Google Play) Free</td>
<td>77.25%</td>
<td>80.42%</td>
</tr>
<tr>
<td>Skype Whatsapp (Google Play) Free</td>
<td>77.7%</td>
<td>57.67%</td>
</tr>
<tr>
<td>Angry Birds Space Premium (Google Play) 0.89 €</td>
<td>11.14%</td>
<td>11.14%</td>
</tr>
<tr>
<td>Angry Birds Space Premium (Unofficial) Free</td>
<td>14.29%</td>
<td>4.23%</td>
</tr>
<tr>
<td>Monkey Jump 2 (Unofficial) Free</td>
<td>1.06%</td>
<td>1.06%</td>
</tr>
<tr>
<td>Viber (Google Play) Free</td>
<td>99.66%</td>
<td>99.74%</td>
</tr>
<tr>
<td>WeChat (Google Play) Free</td>
<td>11.11%</td>
<td>6.35%</td>
</tr>
<tr>
<td>Candy Zuma (Unofficial) Free</td>
<td>4.76%</td>
<td>0.53%</td>
</tr>
<tr>
<td>Ruzzle (Google Play) 2.50 €</td>
<td>13.17%</td>
<td>13.22%</td>
</tr>
<tr>
<td>Ruzzle (Unofficial) Free</td>
<td>17.56%</td>
<td>5.82%</td>
</tr>
</tbody>
</table>

is the requested set of permissions, which effectively gives the app the “power” to invoke critical functions and access critical resources. This, per se, cannot be the only parameter used to decide whether an app is risky or not. In fact, genuine apps may request these permissions to legally access, for example, the contact detail (as a contact manager), or to send SMSs (as a customized text manager), and so on. Obviously, if these permissions were dangerous regardless of the apps, then Android would remove the possibility of using them. Hence, other criteria are relevant, such as the marketplace (an official market has usually a pre-filtering step to remove some malicious or repackaged apps), the developers (the most important ones do not want to risk their reputation), the user ratings and the number of downloads (which are an indicator of the app “goodness”).

MAETROID is shipped as a custom app to run on the user-device. Whenever a new app is downloaded, the installation process is paused to run the classification process locally. Then, MAETROID gives the user a user-friendly decision about the app risk level. Then, it is up to the user to decide whether the installation needs to be resumed. We have to point out that MAETROID is not an anti-virus nor an intrusion detection system. MAETROID focuses on elements which are visible to every users, i.e. permissions, market and app popularity, by evaluating them and then generating a single decision, easy to understand. Furthermore, it does not analyze the app’s code. On the contrary, anti-virus solutions base their decision by looking statically for known bad signatures inside the app’s code (black-list approach). Other approaches, such as anomaly intrusion detection systems, look for anomaly patterns at run-time. For these reasons, we see MAETROID as an orthogonal approach to these solutions. In particular, MAETROID can be the first line of defense in deciding whether an app may be potentially dangerous. As an example, if an app is classified as Medium-Risk (yellow face), one can decide to run it in a sand-boxed environment under control of an IDS.

The analysis of MAETROID, as already discussed in the former sections, is performed at deploy time and no further checks are performed after the user accepts the decision. In particular, MAETROID does not enforce security at run-time, to not impose any overhead on the apps execution. Thus, MAETROID strongly differs, by design choice, from security frameworks like MOCANA [41] or Samsung Knox [42] which are
designed for “high-security government or military deployment”, enforcing security from the hardware to app level through trusted storage for remote attestation and a dedicated market where only vetted apps are published. Furthermore, the configuration of these systems are usually centralized and implemented by expert administrators. The MAETROID solution is, on the other hand, designed for a wider set of users, not requiring dedicated hardware nor customized OS, and with little (or none) knowledge of security.

As discussed in Section 4, in our analysis we first assigned to all the five criteria the same priority (0.2). After elaborating the answers to question 8 of the survey, we have re-classified all the 180 apps discussed in Section 4, assigning different priorities to each of the criteria, namely by ranking them based on the user’s perception of importance. More precisely, we considered the “user rating” criterion two times more important than the other criteria and we reduced the importance of the “permission” criterion by half of its value. The new classification of the apps yielded slightly worse results than the one obtained in our first analysis, i.e. with apps more app being misclassified. This is due to the fact that by reducing the importance of permissions there are more chances of installing malicious apps. MAETROID coalesces in a single and easy-to-understand decision index five criteria that describes the security and quality of an Android app. This trustworthiness index could be significant for yielding better advices to the users than the permission system currently adopted by Android devices, as the first results of our survey-based investigation indicate. In fact, as shown by the comparison of the answers to questions 11 and 12, several subjects changed their mind when installing malicious apps after using MAETROID. Those apps that have been classified by MAETROID as highly-dangerous have been selected by less than the 0.5% of the subjects. The number of users willing to install dangerous apps has noticeably decreased after the MAETROID classification results, in particular by 66% for Angry Birds Space Premium Unofficial and Ruzzle Unofficial, and by 90% for Candy Zuma Unofficial, respectively. For apps classified as Medium-Risk, such as Skype, a significant percentage subjects are not willing to install the app anymore (25% less than before). These results are encouraging, since they indicate that MAETROID is effective in driving the correct decision when installing a new app.

One limitation of the MAETROID approach is that it uses a single index for evaluation, with three possible values only, and this might somehow be too coarse-grained in some situations. As an example, the vast majority of apps found on official market, such as Google Play, are classified as trusted. This is because it rarely happens that malicious apps are hosted on official markets. However, one could argue that even if safe, these apps sometime requires a too large set of permissions and should be penalized. Further, advanced users would like to know more about the threats for each of the MAETROID categories (financial, privacy and system), which is not represented by the final output. The first issue can be easily taken into account by penalizing apps that requires a large number of permissions regardless of the market, developer or the number of download. As we have already detailed, the parameters, such as the weight given to each criteria, can be customized by the users. Regarding
the second issue, one future development concerns the addition of some further explanations to the final decision, e.g. in the form of an optional tab that the user can open to understand in detail the final decision.

Concerning the results of the classification, we have to say that weights (i.e., the importance) of the parameters can be customized as to have more granular results with respect to the user’s expectation. As an example, one can decide to give more importance to the developer rather than to the market, if the user always downloads from official markets. Further, if a particular user is really concerned about the privacy, the matrix weights can be biased towards giving a higher threat score whenever an app includes more privacy-risky permissions. To summarize the results of the survey, we can conclude that subjects are aware of the security threats brought by malicious mobile apps. However, the current Android alerting system, which consists of showing to the user the list of permissions requested by an app before installing it, seems to fail to be effective. As highlighted by the answers to the survey, several subjects gave to permissions a limited importance and they base their decision (either to install an app or not) on non security-related criteria, such as the number of downloads and the user rating, which we have shown no to be relevant.

8 Conclusions

Protecting users from dangerous apps is a compelling issue. Though the main mobile OSes have already introduced some security mechanisms for device and user protection, they still present several usability-related flaws. The results of the survey we have conducted show that users have a realistic view of mobile security threats and are willing to protect their devices. However, users are often tricked in installing malicious apps looking as genuine, since users seem to consider the app popularity and user rating more important than, e.g., the declared permissions. To this end we have developed MAETROID, a multi-criteria decision framework for the analysis of Android apps. MAETROID has been exploited to classify more than 11,000 Android apps, coming from Google Play and from a database of known malware. In our experiments, the trustworthiness index of MAETROID has proved to be able to drive correct decisions on whether to install dangerous apps.

We believe that the introduction of a simple index, as the one produced by MAETROID, may improve the overall mobile device security and user awareness. In fact, suspicious apps could be identified and further analyzed, before being executed by users. Moreover, the presence of the threat score could be an incentive for developers to accurately choose the permissions needed by their apps, effectively tackling also the permission overdeclaration issue. MAETROID comes as an Android app which enforces security without imposing overhead to the user, since it becomes active only when installing a new app.
Acknowledgement

The research leading to these results has received funding from the EU Seventh Framework Programme (FP7/2007-2013) under grant no 610853 (COCO Cloud), Regione Toscana (POR CReO 2007-2013) line of activity 1.5.a - 1.6 (SECURE!) and My Information Bubble (Funded by Registro.it).

References


URL http://dx.doi.org/10.1007/978-3-642-35371-0_7


A Survey Structure

Survey on security for mobile devices. Welcome to the survey on security for mobile devices!

This survey helps users understanding their awareness of how to securely use their smartphones.

There are 12 questions in this survey.

1. Age of user.

Please choose only one of the following:

- < 18
- 18 – 25
- 26 – 35
- 36 – 45
- > 45

2. Gender.

Please choose only one of the following:

- Female
- Male
3. Occupation status.

Please choose only one of the following:
- Student
- Worker
- Other

4. What kind of smartphone do you use?

Please choose all that apply:
- Android
- iPhone
- 8 Windows 8
- No one
- Other

5. What are the threats for smartphones you fear most?

Please choose all that apply:
- Theft of Smartphone
- Identity Theft (e.g., Facebook credentials saved on the Smartphone)
- Theft of credit card data
- Anonymous calls / stalking
- Installing malware (e.g., virus, etc)
- Attacks to user privacy (access to personal images from unauthorized people, position tracking)
- None
- Other

6. Which of the following basic security functionalities do you use to protect your Smartphone?

Please choose all that apply:
- PIN to protect calling card
- Lock screen with password/sequence/PIN
- Disable GPS when not used
- Constant update of operating system and apps
- Check permissions required by apps at install time
- Not use wireless in public places
- Use of Anti-virus or secure apps (e.g., to encrypt private data)
- Backup of personal data
- None
- Other

7. Which of the following features would you like to see in a security app?

Please choose all that apply:
• Disable applications
• SMS blocker
• Call blocker
• Anti-theft features (e.g., remotely block a stolen smartphone)
• Classification of an application hazardousness before installing it
• Do not install applications from unofficial markets
• Anti-virus
• Personal data encryption with a password
• Send of SMS to secure number in case of anomalous events (e.g., change of SIM, settings)
• Block adult contents
• Hide GPS positions to some apps
• Secure/anonymous browser
• Data Backup
• Disable permissions given to apps
• Feedback on bad application behavior (leaking user money, battery depletion, download of apps in background, etc)
• None
• Other

8. How much each of the following item is critical when deciding whether to install or not an application (1 little, 5 very much) ?

Please choose the appropriate response for each item:

• Market: 1 2 3 4 5
• User ratings: 1 2 3 4 5
• Few permissions: 1 2 3 4 5
• Number of download: 1 2 3 4 5
• Popular application: 1 2 3 4 5

9. Would you use an application that, automatically, rates the degree of hazardousness of an application you are going to install?

Please choose only one of the following:

• No
• Yes, with a score between 1 (harmless) and 10 (dangerous)
• Yes, with a traffic light: red (harmless) / yellow (suspicious) / red (dangerous)
• Yes, with a smiley: smile (harmless) / “poker face” (suspicious) / sad (dangerous)
• Other

10. The score on WhatsApp degree of hazardousness downloaded from an UNOFFICIAL market shows a red light (sad smiley): what are you going to do?

Please choose all that apply:
• I’m not going to install it and I quit
• I’m going to search it in another market
• I’m going to install it because it’s very popular
• Other

11. Which of the following applications would you install on a brand new Smartphone?

Please choose all that apply:
• Whatsapp (Google Play) Free
• Skype (Google Play) Free
• Angry Birds Space Premium (Google Play) 0,89€
• Angry Birds Space Premium (Unofficial) Free
• Monkey Jump 2 (Unofficial) Free
• Viber (Google Play) Free
• WeChat (Google Play) Free
• Candy Zuma (Unofficial) Free
• Ruzzle (Google Play) 2,50€
• Ruzzle (Unofficial) Free

12. If you also had a score on the hazardousness of an application using a smiley, which of the following applications would you install on a brand new Smartphone?

Please choose all that apply:
• Whatsapp (Google Play) Free 😊
• Skype (Google Play) Free 😊
• Angry Birds Space Premium (Google Play) 0,89€😊
• Angry Birds Space Premium (Unofficial) Free 😊
• Monkey Jump 2 (Unofficial) Free 😊
• Viber (Google Play) Free 😊
• WeChat (Google Play) Free 😊
• Candy Zuma (Unofficial) Free 😠
• Ruzzle (Google Play) 2,50€🙁
• Ruzzle (Unofficial) Free 😠

B Excerpt of Analyzed Android Permissions
Table 15: Partial list of Android Permissions and Associated Threat Levels, per Index

<table>
<thead>
<tr>
<th>Permission</th>
<th>Class</th>
<th>PT</th>
<th>ST</th>
<th>FT</th>
</tr>
</thead>
<tbody>
<tr>
<td>android.permission.ACCESS_COARSE_LOCATION</td>
<td>Dangerous</td>
<td>0.4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.ACCESS_FINE_LOCATION</td>
<td>Dangerous</td>
<td>0.8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.ACCESS_LOCATION_EXTRA_COMMANDS</td>
<td>Normal</td>
<td>0.2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.ACCESS_LOCK_LOCATION</td>
<td>Normal</td>
<td>0</td>
<td>0.4</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.ACCESS_NETWORK_STATE</td>
<td>Normal</td>
<td>0.2</td>
<td>0</td>
<td>0.4</td>
</tr>
<tr>
<td>android.permission.ACCESS_WIFI_STATE</td>
<td>Normal</td>
<td>0</td>
<td>0</td>
<td>0.4</td>
</tr>
<tr>
<td>android.permission.AUTHENTICATE_ACCOUNTS</td>
<td>Dangerous</td>
<td>0.6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.BATTERY_STATS</td>
<td>Normal</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.BLUETOOTH</td>
<td>Dangerous</td>
<td>0.6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.BLUETOOTH_ADMIN</td>
<td>Dangerous</td>
<td>0.8</td>
<td>0.6</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.BROADCAST_STICKY</td>
<td>Normal</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.CALL_PHONE</td>
<td>Dangerous</td>
<td>0.6</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.CAMERA</td>
<td>Dangerous</td>
<td>0.8</td>
<td>0.6</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.CHANGE_CONFIGURATION</td>
<td>Dangerous</td>
<td>0</td>
<td>0.4</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.CHANGE_NETWORK_STATE</td>
<td>Dangerous</td>
<td>0.2</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>android.permission.CHANGE_WIFI_MULTICAST_STATE</td>
<td>Dangerous</td>
<td>0</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>android.permission.CHANGE_WIFI_STATE</td>
<td>Dangerous</td>
<td>0</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>android.permission.CLEAR_APP_CACHE</td>
<td>Dangerous</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.PROCESS_OUTGOING_CALLS</td>
<td>Dangerous</td>
<td>0.8</td>
<td>0.6</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.READ_CALENDAR</td>
<td>Dangerous</td>
<td>0.8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.READ_CONTACTS</td>
<td>Dangerous</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.READ_SMS</td>
<td>Dangerous</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.RECEIVE_BOOT_COMPLETED</td>
<td>Normal</td>
<td>0.2</td>
<td>0.4</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.RECEIVE_SMS</td>
<td>Dangerous</td>
<td>1</td>
<td>0.8</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.RECEIVE_SMS_COMPLETED</td>
<td>Dangerous</td>
<td>1</td>
<td>0.8</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.RECEIVE_WAP_PUSH</td>
<td>Dangerous</td>
<td>0.4</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>android.permission.RECORD_AUDIO</td>
<td>Dangerous</td>
<td>0.8</td>
<td>0.6</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.REORDER_TASKS</td>
<td>Dangerous</td>
<td>0.4</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>android.permission.RESTART_PACKAGES</td>
<td>Normal</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.SEND_SMS</td>
<td>Dangerous</td>
<td>0.8</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.WRITE_CALENDAR</td>
<td>Dangerous</td>
<td>0.8</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.WRITE_CONTACTS</td>
<td>Dangerous</td>
<td>0.6</td>
<td>0.6</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.WRITE_EXTERNAL_STORAGE</td>
<td>Dangerous</td>
<td>0.2</td>
<td>0.6</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.WRITE_SMS</td>
<td>Dangerous</td>
<td>0.4</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.WRITE_SOCIAL_STREAM</td>
<td>Dangerous</td>
<td>0.6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>android.permission.WRITE_SYNC_SETTINGS</td>
<td>Dangerous</td>
<td>0</td>
<td>0.4</td>
<td>0</td>
</tr>
</tbody>
</table>