A Multi-Criteria-based Evaluation of Android Applications

Gianluca Dini[†], Fabio Martinelli[‡], Ilaria Matteucci[‡], Marinella Petrocchi[‡], Andrea Saracino[†‡], and Daniele Sgandurra[‡]

[†] Dipartimento di Ingegneria dell' Informazione, Università di Pisa, Italy
name.surname@iet.unipi.it
[‡] Istituto di Informatica e Telematica, CNR, Pisa, Italy
name.surname@iit.cnr.it

Abstract. Android users can face the risk of downloading and installing bad applications on their devices. In fact, many applications may either hide malware, or their expected behavior do not fully follow the user's expectation. This happens because, at install-time, even if the user is warned with the potential security threat of the application, she often skips this alert message. On Android this is due to the complexity of the permission system, which may be tricky to fully understand.
We propose a multi-criteria evaluation of Android applications, to help the user to easily understand the trustworthiness degree of an application, both from a security and a functional side. We validate our approach by testing it on more than 180 real applications found either on official and unofficial markets.

1 Introduction

Android is an open source Operative System (OS) designed for mobile devices, such as smartphones and tablets, that currently has the largest share of the mobile device market. Part of its success is due to the large number of applications (or *apps*) that are available for Android devices, which can be developed using the Standard Development Kit (SDK). Android SDK is free to download and to use: hence, virtually anyone can develop applications, from expert and professional developers to programmers with limited experience. Applications are distributed through the application market, in which any user (even malicious ones) can share their own applications. The greatest channel for application distribution is *Google Play* (formerly known as *Android Market*), which tries to ensure the quality of distributed apps with some simple control mechanisms. However, it is always possible to get into low-quality applications, or malicious ones, when surfing Android markets or the several unofficial markets found on the web, especially if we consider that in these unofficial markets applications can be distributed without any kind of control.

Malicious applications are the greatest security threat for Android systems and, hence, to prevent such applications to damage smartphones, Android implements two security-control mechanisms: *sandboxing* and *permissions* [1]. *Sandboxing* is achieved by means of application isolation: each application runs in its own instance of the Dalvik Virtual Machine (DVM), an optimization of the Java Virtual Machine, and each DVM

is treated as a different UNIX user, by the Android's underlying Linux kernel. The isolation ensures that malicious applications do not interfere with the activity of the good ones. The *permission* system is a mechanism of access control to protect resources and critical operations. At install-time, permissions required by an application are shown to the user, which can decide whether to grant or to deny them. However, several criticisms have been raised against this system, which results too coarse-grained [2] and too much reliant on user knowledge and expertise [3]. The main problem of this approach is that the acceptance policy for an application's requested permission is "all or nothing", that is, the user cannot accept only a subset of the required permissions. Then, if the user does not agree even with a single permission, the installation is not performed. Furthermore, due to the large number of existing permissions, even an expert user may not fully understand all of them and, as a consequence, several users install applications without caring about the required permissions and without questioning about the potential security threats [3]. Hence, a simpler mechanism to guide average users in the job of deciding whether to install or not an application is necessary, without the burden of reading (and understanding) all the declared permissions.

In this paper we present a multi-criteria approach that combines information retrieved from permissions with the reputation indexes provided by markets, to compute the trustworthiness of an application and the security threat that it may represent. In more detail, the contribution of the paper are the following:

- we propose a novel classification of Android permissions, in which we assign to each permission a *threat score* according to the criticality of both resources and critical operations they control;
- we compute a *global threat score* for each application, which is a function of the threat score of all the required permissions;
- we propose the application of the Analytical Hierarchy Process (AHP), a well-known methodology for multi-criteria decision, to classify applications according to the global threat score and to reputation indexes retrievable from markets. Each application can be considered *trusted*, or *untrusted*, or *deceptive*. By following the suggested value of the classification, users can avoid the installation of potentially infected or not-properly behaving applications;
- we validate our approach applying the methodology to 180 real applications with different features, where 40 applications were infected by common malware. The tested applications have been correctly classified. Hence, the user can consider the trustworthiness level of the application by only observing the result of this classification process, without the need of understanding all of the requested permissions.

The paper is organized as follows: in Section 2, we explain how we classify permissions and how we measure the threat of applications. Section 3 recalls the AHP methodology and how it is applied in our scenario. Section 4 reports the results of our approach and some practical examples. Section 5 points to some related work concerning the Android permission system. Finally, Section 6 briefly concludes, proposing some future extensions.

2 Classification of Android permissions

In this section we give some notions on the Android permission system and we explain how we assign a threat score to each permission and a global threat score to an application.

Currently, Android defines 120 permissions[1], where each permission is related to a specific device resource or to a critical operation that can possibly be exploited to harm the user privacy, her money, or the device itself. Permissions required by an application are declared in the `AndroidManifest.xml` file that is part of the application itself and that is bound to it 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 focus on the first two classes. In fact, *signature* and *signature-or-system* Android permissions cannot be required by custom applications, since only applications signed with the Google private key can use those permissions. The Android permission classification is used to choose which permissions have to be shown to the user at install-time. The *dangerous* permissions are automatically shown to the user, whereas the *normal* ones are listed in a separate sublist addressed as "Other Permissions". If the user accepts all the permissions required by an application, then this application is installed and, at run-time, it is allowed to use the critical resources and operations granted to the permissions without asking for further authorizations.

Several criticisms have been raised against the Android permission system. Firstly, the system is too coarse-grained [2], since the user can only choose whether to accept all of the permissions declared by an application or to refuse to install the application. Furthermore, the user is usually unable to determine if an application can be trusted, based upon this list of required permissions. In fact, there are several permissions and some of them are really difficult to understand even to expert users. It is often the case that average users do not care about permissions and their security hazards, thus installing potentially malicious applications [3]. Furthermore, some developers are used to declare more permissions in the manifest file than those effectively needed by the application (the so called *Permission Overdeclaration* [4]). This happens because some permissions have similar names and their description is not self-explicative for some developers. Therefore, Android users, seeing a very long permission list when installing a new application, are less encouraged to read and understand them.

To overcome the problem of permission understanding, we discuss a novel way to compute the threat score of an application, based upon the requested permissions. The proposed system shows to the user, in a simple way, the dangerousness of the application. This score is a number ranging over the interval $[0, 15]$, where 0 represents an application that only requires unharmful permissions, whilst 15 is a strongly critical application that requires all the Android permissions.

2.1 Threat Indexes

The goal of the proposed method is to compute a threat score of applications according to the permissions that they declare. For this reason, we have analysed all the 120 default

[1] http://developer.android.com/reference/android/Manifest.permission.html

Android permissions, and scored according to their threat. For each permission we have defined three threat indexes, to represent the type of threat and the severity of damage that can be achieved if these permissions are exploited by a malicious application. These indexes are: *privacy* threat, *system* threat, and *money* threat, and they are defined in the interval $[0, 1]$, where 0 means no threat and 1 means the highest threat (see Table 1). We have manually assigned to each permission these three threat values, according to the actions, or resources, controlled by that specific permission and their relation with well-known malware attacks. In more details, the rationale of which value to assign to

0	No Threat
0.2	Low Threat
0.4	Low-to-Moderate Threat
0.6	Moderate Threat
0.8	Moderate-to-High Threat
1	High Threat

Table 1: Threat Levels

each threat index, according to the considered permission, is discussed in the following.

Privacy Threat Permissions with a high value of this threat are those that control the access to sensitive data, e.g. the user's contact list, stored files, Internet bookmarks and chronology, or SIM and device information such as the IMEI and IMSI codes. On the other hand, a medium-low value of privacy threat is assigned to those permissions that access sensors such as camera or microphone, since they can be maliciously used to spy the user behavior.

System Threat A high value of system threat is assigned to applications accessing system data, e.g. permissions that allows the application to write to the device memory, install and uninstall other applications, or access sensors whose improper use can leak the battery energy.

Money threat High values of this index are assigned to permissions that control services whose use directly imply a money cost, such a phone calls or outgoing SMS. Conversely, if the cost is indirectly related to a specific permission, it receives a medium money threat value, e.g. the `CHANGE_NETWORK_STATE` permission that allows an application to enable or disable the data connection whose available traffic amount is generally limited to a few Gigabytes per month and, afterward, the user has to pay all the outgoing/incoming traffic byte per byte.

Example. The permission `SEND_SMS` enables an application to send SMS messages without requiring user confirmation. Thus, an application that declares this permission can send SMS messages, with any text, at any rate, and at any phone number, without the user noticing it (unless she checks her available credit). This permission has been exploited by several malware to leak the user credit by sending messages to premium-rate number, or to threaten her privacy by sending information, such as the IMEI and

IMSI codes, to a phone number controlled by the attacker [5]. Table 2 shows the threat indexes assigned to the permission `SEND_SMS`.

Permission	Privacy Threat	System Threat	Money Threat
`SEND_SMS`	0.8	0	1

Table 2: Threat Level of `SEND_SMS` permission

The privacy threat is considered medium-high since SMS messages can be used as a vector to steal sensitive information (some malware use them to do so). However, this information has to be accessed before it can be sent and this requires other specific permissions. Our complete classification of Android permissions can be found in [6]. It is worth noticing that we do not consider the default "signature" and "signature-or-system" permissions, since they can only be requested by device manufactures or by Google applications that we assume trustworthy. Moreover, those permissions are not shown to the user at install-time, thus they should not be part of the scoring process.

2.2 Global Threat Score

For each application α, we define the *global threat score* σ, which is a function of the threat score of all the permissions declared by the application α, as follows:

$$\sigma = \frac{\sum_{i=1}^{n} w_p pt_i + w_s st_i + w_m mt_i}{max\left\{1, \lceil log(n) \rceil\right\}} \tag{1}$$

where n is the number of permissions declared by the application α, pt_i, st_i, mt_i are, respectively, the privacy, system, and money threat of the i-th permission required by α, and w_p, w_t, w_m are used to weight the importance of a specific threat factor. In the current implementation, we consider w_m being three times greater than w_t and w_p: we consider the money threat the more relevant, since it can harm the user more directly. The number of permissions that concern privacy threat and system threat are three time larger than the number of permissions concerning money. The denominator of (1) should render the idea that an application with a lot of medium threat permissions should not be considered as dangerous as an application that comes with few extremely dangerous permissions. However, since the increase of the denominator is logarithmic, it is not feasible for an attacker to hide the threat of an application declaring a large number of low threat permissions. We consider applications with σ lower than 4 as *low-threat* applications, while ones with σ in the interval $[1, 4]$ are *moderate threat* to *high-threat*. Higher values of σ mean *extremely* critical applications.

The value σ estimates how much an application is critical from the security point of view. Hence, the more permissions are required by an application, and the more dangerous these permissions are, the more critical the application becomes. If an application receives a low-threat score, this should increase the likelihood that this application

is downloaded and, as a consequence, this should encourage developers to accurately choose the permissions required by their applications. However, several applications actually require a large number of permissions to perform all their functions, especially *communication* and *social* applications, and they should not be considered as suspicious. This leads us to rely on a multi-criteria decision system (Section 3) in order to classify an application with respect to a set of criteria, among which the threat score σ.

3 Multi-criteria assessment of Android applications

In this section, we show how to apply the Analytical Hierarchy Process (AHP) to assess the security level of an Android application. Before instantiating the methodology, we briefly recall the basic steps of AHP.

3.1 The Analytical Hierarchy Process

The Analytic Hierarchy Process (AHP) [7,8] is a multi-criteria decision making technique, which has been largely used in several fields of study. 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*. This approach requires to subdivide a complex problem into a set of sub-problems, equal in number to the chosen criteria, and then compute the solution (alternative) by properly merging the various local solutions for each sub-problem.

The process can be described using an example: let the reader suppose to have as *goal* "choosing a restaurant for dinner". The possible alternatives are a Japanese sushi bar, a French *brasserie*, and an Italian *trattoria*. The problem must be structured as a hierarchy, as shown in Figure 1, linking goal and alternatives through a set of criteria. In the proposed example, appropriate *criteria* could be: cost, food, and staff.

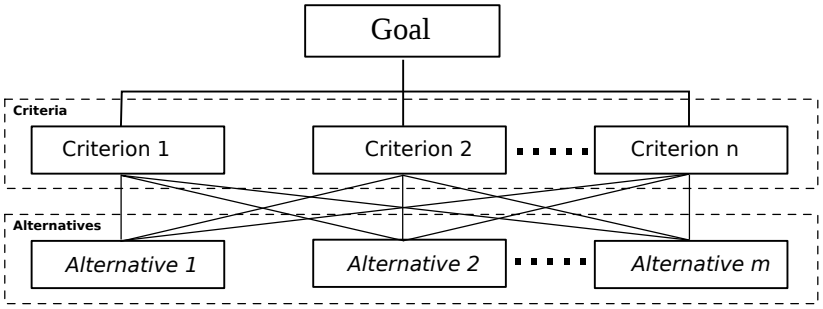

Fig. 1: Generic AHP Hierarchy

Once the hierarchy is built, the relevance of each alternative with respect to each criterion is established, comparing them in a pairwise fashion. Comparisons are done

Intensity	Definition	Explanation
1	Equal	Two elements contribute equally to the objective
3	Moderate	One element is slightly more relevant than another
5	Strong	One element is strongly more relevant over another
7	Very strong	One element is very strongly more relevant over another
9	Extreme	One element is extremely more relevant over another

Table 3: Fundamental Scale for AHP

through a scale of numbers typical to AHP (see Table 3). The scale indicates how many times an alternative is *more relevant* than another one, with respect to a specific criterion. The relevance is established according either to subjective or objective statements; for example, if the Italian trattoria is much more cheaper than the Japanese sushi bar, we can state that the alternative trattoria is strongly more relevant (and then more advised) than the sushi bar according to the *cost* criterion.

Pairwise Comparison Matrices. Pairwise comparisons for each criterion are expressed in a matricial form, called *pairwise comparison matrices*. 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}, $a_{ij} = \frac{1}{a_{ji}}$. For comparisons matrices the concept of *consistence* is defined. 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 comparisons matrices are, at least, semi-consistent. To measure the consistency of a comparison matrix, the consistency index $CI = \frac{\lambda_{max}-n}{n-1}$ has been defined [9]. For a consistent matrix $CI = 0$, whilst a matrix is considered semi-consistent if $CI < 0.1$. If this condition does not hold, the comparisons matrix should be re-evaluated. Table 4 shows the comparison matrix for the *cost* criterion of the three restaurants example. The same procedure is repeated to compare the restaurants with respect to the other criteria, namely food and staff, obtaining two additional comparisons matrices that we do not report for the sake of brevity.

COST	Japanese	Italian	French	Loc. Prio.
Japanese	1	$\frac{1}{5}$	$\frac{1}{3}$	0.11
Italian	5	1	2	0.58
French	3	$\frac{1}{2}$	1	0.31

Table 4: Example Comparisons Matrix: Restaurants vs Cost (CI=0.0018473)

How to Compute 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 largest eigenvalue [10]. In Table 4, the vector of local priorities for the cost criterion is reported on the right side of the matrix, and it expresses that, for the cost criterion, the Italian restaurant is the most advised among the three alternatives.

Furthermore, it is possible to express the relevance of a criterion with respect to the goal. For example, if the dinner cost has more relevance than the kindness of the staff, that is, we may accept that the staff kindness is not satisfactory, but is very important that the dinner is cheap, this can be specified using the scale of Table 3. Hence, we have an additional pairwise comparisons matrix for the criteria, which size is $k \times k$ where k is the number of criteria.

How to Compute Global Priorities. Global priorities are computed through a weighted sum of the local priorities computed in the previous step:

$$P_g^{a_i} = \sum_{j=1}^{k} p_g^{c_j} \cdot p_{c_j}^{a_i} \tag{2}$$

where $P_g^{a_i}$ is the global priority of the alternative a_i, $p_g^{c_j}$ is the local priority of criterion c_j with respect to goal and $p_{c_j}^{a_i}$ is the local priority of alternative a_i with respect to criterion c_j. The vector of global priorities for the restaurant problem is computed as:

$$P_g = 0.47 \begin{bmatrix} 0.11 \\ 0.58 \\ 0.31 \end{bmatrix} + 0.15 \begin{bmatrix} 0.1 \\ 0.6 \\ 0.3 \end{bmatrix} + 0.38 \begin{bmatrix} 0.31 \\ 0.24 \\ 0.45 \end{bmatrix} = \begin{bmatrix} 0.19 \\ 0.45 \\ 0.36 \end{bmatrix}$$

Where $\{0.47, 0.15, 0.38\}$ are an example of local priorities of the criteria, and the column vectors are the local priorities of each criterion w.r.t. the three alternatives. This result means that the Italian trattoria is the best choice[2].

3.2 An AHP Instance for Evaluating Android Applications

We instantiate the AHP decision methodology to assess the quality of an Android application as follows: given an Android application with the following parameters: a threat score σ, a developer δ, a number of download η , a market μ , a user-rating ρ, then the *goal* consists in assigning to the application one the following *alternative* labels:

Trusted. This alternative means that the application correctly works and should not hide malicious functionalities.

Untrusted. This alternative means that, even if apparently working as the user expects from a functional perspective, the application could violate the security of the mobile device.

Deceptive. This alternative means that the application is neither functional nor secure.

[2] The numerical values have been assigned as an example to show a well-formed matrix. The authors do not aim at ranking Japanese, Italian, and French restaurants.

The problem is parametric w.r.t. the application, thus for two different applications the same fixed alternatives have a different relevance w.r.t. the same criterion. Hence, the five parameters (σ, δ, η, μ, ρ), wich are the *criteria* of the problem, assume different values for different applications.

In the following we explain how to build the related comparison matrices and the possible values that each criterion may assume:

Market (μ). Applications are generally distributed through application markets. The most popular market is Google Play, also referred as the *official* market. A developer that wants to publish applications on Google Play has to buy a developer account at the price of 25$, receiving in exchange a private key that she will use to digitally sign her application before publishing them [1]. If users report an application as malicious, then this application 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 the user to understand the application quality. These features make the official market a trustworthy place where to download apps. Nevertheless, several malware have been found in the Android Market [11][12] starting from the second half of year 2011.

There exists also a plethora of *unofficial* marketplaces that do not require developer registration and that give access to some applications that are not available on the market. However, unofficial markets often miss reputation indexes and sometimes there is no control on the quality of the applications, so that it is easier to accidentally download malicious applications. We also consider a third value for the market parameter: *manually installed*. This is the case in which the user manually installs the application, without the need of an installer, like those that are usually necessary to download apps from markets, both official and unofficial ones.

With reference to Table 3, we show the *relevance* of each alternative, for the three possible values of μ:

- μ = official: we consider that *trusted* is moderately more relevant than *deceptive* and strongly more relevant than *untrusted*.
- μ = unofficial: we consider that *untrusted* is moderately more relevant than *trusted* and slightly more relevant than*deceptive*.
- μ = manually installed: we consider that *untrusted* is slightly more relevant than *trusted* and *deceptive* (that are equally relevant).

According to this information, comparison matrices (full list in [6]) are directly computed.

Developer (δ). We consider three types of developers: *standard*, *Top*, and *Google*. Google rewards the best developers with a *Top Developer* badge, reported on each application they publish. Hence, we are considering these developers as strongly trusted and known to produce high-quality applications. On Google Play, *Google Inc.* itself is considered a Top Developer; however, we consider Google more trusted than other developers, since it distributes apps that often are vital to the normal activity of Android smartphones.

All the other developers are considered standard and since the Top Developer badge is only used on Google Play, all the developers of applications from unofficial markets have been labeled standard. We make this assumption because applications

coming from unofficial markets can be modified versions of well-known applications on Google Play, but these modifications also change the identity of the developer, together with her trustworthiness.

Evaluating pairs of alternatives with respect to the developer means that:

- δ = Google: we consider that *trusted* is extremely more relevant than *deceptive* and *untrusted* (that are equally relevant).
- δ = Top Developer: we consider that *trusted* is very strongly more relevant than *untrusted* and *deceptive* (that are equally relevant).
- δ = Standard: we consider that the three alternatives are equally relevant.

Number of downloads (η). Several markets report the number of downloads for each application. As an example, the so-called "killer applications", i.e. extremely popular apps, have been downloaded from Google Play more than 100 millions of times. In our opinion, these applications should be considered differently from those downloaded much fewer time, e.g. less than 100 times. Hence, we define 7 intervals in which the value η may fall. For very high values of η, *trusted* is extremely relevant. As the value of η decreases, the relevance slowly passes from *trusted* to *untrusted*.

User Rating (ρ). Users can rate applications and leave a comment, which can be shown to other users. Rating is generally expressed as a number that ranges from 1 to 5. We consider applications with a rate less than 2 as *low-quality*, for which the *deceptive* 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.

Threat Score (σ). For each application the threat score is computed as explained in Section. 2. We define the following intervals:

- $\sigma < 4$: *trusted* is very strongly more relevant than *untrusted* and moderately more relevant than *deceptive*.
- $4 \leq \sigma \leq 7$: *untrusted* is very strongly more relevant than the other alternatives (that are equally relevant).
- $\sigma > 7$: *untrusted* is extremely more relevant than *trusted*, and *deceptive* 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. Globally, we define 20 comparison matrices, but it is possible to increase their number to compute a finer granularity for each criterion. Finally, it is worth noticing that the list of proposed criteria is not exhaustive, and the methodology allows the insertion of other rules helpful to evaluate the alternatives. In the current implementation we consider all the criteria as equally relevant.

4 Implementation and Results

We have developed a framework to analyze and classify Android applications, which fully implements the previous strategy. Since several markets do not allow the direct

download of `apk` files on a personal computer, and the applications are installed directly on the device through an installer, we have firstly used the ADB tool to extract the `apk` files from the device and, secondly, `apktool` decoder [3] to extract the manifest files.

To compute the application threat score σ, we have extracted the permissions from the manifest file using an XML parser, and we have computed its σ value according to formula (1). We have used Matlab[4] to implement AHP. Our test-set is composed of 180 Android applications of different categories, which we know in advance to be:

- *Good App*: Application that behaves correctly both from the security and functional point of view.
- *Infected*: Application infected by a malware.
- *Bad App*: Application whose behavior is not coherent with the declared one.

For each application, the five parameters (score σ, market μ, developer δ, rating ρ, and download number η) have been given as input to our implementation of AHP.

In more detail, the test-set consists of:

- 90 applications that come from Google Play, and 50 that come from unofficial markets, whilst 40 are manually installed. Two of the applications coming from unofficial markets and 38 of those manually installed are infected by malware.
- Application categories: Augmented Reality, Books and News, Communication, Desktop Manager, Entertainment, File Managing, Game, Social and Utility, and Antivirus.
- The applications user rating ranges over $\{1, \ldots, 5\}$.
- The number of downloads ranges over [0, 10M+].
- Applications produced by standard developers, Top Developers, or Google.

The results of the tests are shown in Figure 2. All the infected applications have been recognized by AHP as *untrusted*. It is worth noticing that some good apps also fall in this class. These applications come from unofficial markets (labelled as Unoff. in the graph of Figure 2), for which no user rating was available and, hence, they cannot be considered trusted. However, if new information became available for these applications, they will eventually be considered as trusted ones. All the applications coming from Google Play have been classified either as *trusted* or *deceptive* on the base of the user rating, threat score, and number of downloads. All the bad apps applications have been considered *deceptive*.

In the following subsection we describe the application of AHP to two of these applications and the corresponding results.

4.1 Baseball Superstars 2010

Let us suppose that we are uncertain about the nature of an application called Baseball Superstars 2010. We can apply AHP using the values of the five parameters in Table 5. As an example, Table 6 shows the matrix used to compare the three alternatives with respect to the application developer criterion. Baseball Superstars 2010 has been developed by a *Top Developer*.

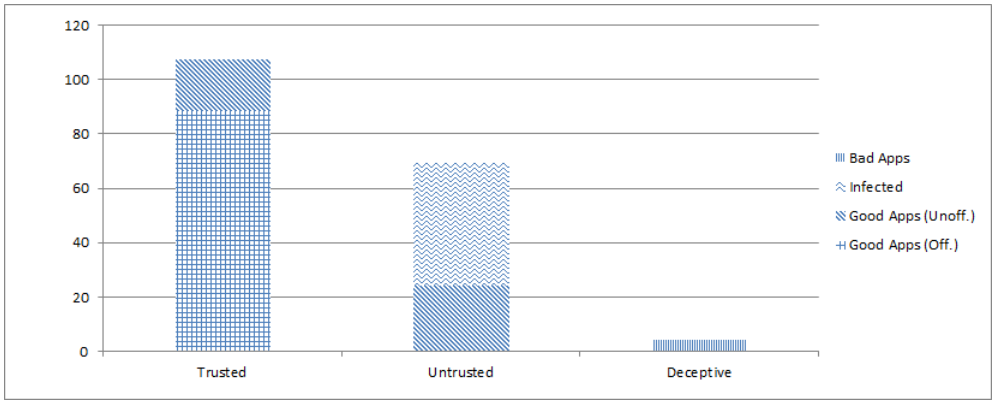

Fig. 2: Evaluation Results

Name	σ	ρ	μ	δ	η
BBSS2010	1	3.8	Google Play	Top Developer	10M+

Table 5: Parameters of Baseball Superstars 2010

top developer	Trusted	Untrusted	Deceptive	Loc.Prio.
Trusted	1	4	7	0.7
Untrusted	$\frac{1}{4}$	1	4	0.23
Deceptive	$\frac{1}{7}$	$\frac{1}{4}$	1	0.07

Table 6: Alternatives vs Criterion: Developer for BBSS2010

Top Developers generally produce high quality apps and they are not likely to publish malicious apps. This is supported by the priorities shown in Table 6. We notice that *trusted* is very strongly favored with respect to *deceptive* and strongly favored with respect to *untrusted*. *Trusted* obtains a higher priority with respect to the other alternatives. Using (2), we merge these local priorities with the ones coming from the comparison matrices for the other four parameters. Thus, we have the vector of the global priorities: $\{0.7, 0.16, 0.14\}$, meaning that the application is considered *trusted*

On a repository of well-known malicious apps[5], we have found an infected version of this game trojanized by the Geinimi malware. This malware leaks information concerning both the user and the device, which is sent via SMS to a number controlled by

[3] http://code.google.com/p/android-apktool
[4] http://www.mathworks.com/products/matlab/
[5] http://contagiominidump.blogspot.com

the attacker. Moreover, it opens a backdoor[6]. Table 7 shows the parameters' values for the two versions of the app. The trojanized version of Baseball Superstars asks several permissions; hence, its threat score σ is much higher than the genuine version. Applying AHP, we have the following global priorities for the malware: $\{0.23, 0.49, 0.28\}$. The potential danger of the application has been recognized; in fact, AHP classifies this application as *untrusted*.

Name	σ	ρ	μ	δ	η
`BBSS2010`	1	3.8	Google Play	Top Developer	10M+
`BBSS2010 (Trojan)`	7.3	-	-	Standard Developer	-

Table 7: Two Versions of Baseball Superstars 2010

4.2 Skype

`Skype` is a popular software used for VoIP and free chat and its mobile version obtained a large success: anyone can make phone calls via Skype with their smartphones, using the data connection instead of the classical and expensive phone call. To work properly, the Android version of `Skype` requires a large number of permissions with a high threat. In fact, computing the threat score with the expression presented in Section 2, `Skype` gets a score of 6.8. Therefore, Skype is an example of a high threat application. In our analysis, we have considered two Skype versions, as reported in Table 8.

Name	σ	ρ	μ	δ	η
`Skype`	6.8	3.8	Google Play	Standard Developer	10M+
`Skype`	6.8	4	Unofficial	Standard Developer	-

Table 8: Two Skype Versions

Applying AHP on the version of `Skype` coming from the official market, we have produced the following global priorities vector $\{0.47, 0.4, 0.13\}$, which means that the application looks *trusted*; in fact, the reputation of the market, and the large number of downloads, strongly raise the trustworthiness of this application, even if it has received 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: $\{0.29, 0.52, 0.19\}$, and the application is considered *untrusted*. Even if the two versions of this app 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 application removal from the market.

[6] http://www.symantec.com/security_response/writeup.jsp?docid=2011-010111-5403-99&tabid=2

5 Related Work

Several extensions and improvements to the Android permissions system have been recently proposed. [13] proposes a security framework that regulates the actions of Android applications defining security rules concerning permissions and sequence of operations. New rules can be added using a specification language. The application code is analyzed at deployment-time to verify whether it is compliant to the rule, otherwise it is considered as malicious code. Our proposal does not require the code to be decompiled and analyzed, since it only requires the permissions list that can be retrieved from the manifest file and other generic information that can be retrieved from the website where the application can be downloaded.

Authors of [2] 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 application and those that have to be denied. Using data mocking, they ensure that an application 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 [14]. The authors designed an improved application installer that allows 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.

In [15], applications have been classified on the base of their required permissions. Applications 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. Finally, another analysis of Android permissions is presented in [4], where the authors discuss a tool, *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.

6 Conclusions and Future Work

We have presented a multi-criteria evaluation of Android applications, on the basis of their meta-data, such as reputation indexes and declared permissions. We have proposed a novel definition for the threat score of an application, according to the application declared permissions. The proposed decision making procedure combines this threat score with information regarding the developer, the rating, and the number of downloads of the application. We have validated this procedure by testing it on well-known trusted and infected applications. The results confirm the good nature, or the malicious nature, of such applications. The proposed solution is based upon static analysis of XML files and, hence, it does not require the code to be decompiled and then analyzed.

We are currently investigating the possible combination of the decision framework with a *per-application monitoring intrusion detection systems* (IDS). Per-application IDSes generally monitor only a subset of all the applications running on a device, e.g.

those that are considered suspicious. Since monitoring is a costly operation, from the point of view of resources, we aim to combine the proposed approach with an IDS, to monitor only the applications that received an *untrusted* classification.

References

1. S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.R. Sadeghi, B. Shastry: Practical and Lightweight Domain Isolation on Android. In ACM, ed.: 1st ACM workshop on Security and privacy in smartphones and mobile devices (SPSM11). (2011) 51 – 61
2. Y. Zhou, X. Zhang, X. Jiang, V. W. Freeh: Taming information-stealing smartphone applications (on android). In: 4th International Conference on Trust and Trustworthy Computing (TRUST 2011). (2011)
3. A.P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, D. Wagner: Android permissions: User attention, comprehension, and behavior. Technical report, Electrical Engineering and Computer SciencesUniversity of California at Berkeley (2012) http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-26.html.
4. A.P. Felt, E. Chin, S. Hanna, D. Song, D. Wagner: Android Permissions Demystified. In ACM, ed.: 8th ACM conference on Computer and Communications Security (CCS'11). (2011) 627 – 638
5. Xuxian Jiang: Multiple Security Alerts: New Android Malware Found in Official and Alternative Android Markets (2011) http://www.csc.ncsu.edu/faculty/jiang/pubs/index.html.
6. G. Dini, F. Martinelli, I. Matteucci, M. Petrocchi, A. Saracino, D. Sgandurra: A Multi-Criteria-Based Evaluation of Android Applications. Technical report, Istituto di Inforatica e Telematica, CNR, Pisa (2012) url: http://www.iit.cnr.it/node/17019.
7. Saaty, T.L.: Decision-making with the ahp: Why is the principal eigenvector necessary. European Journal of Operational Research **145**(1) (2003) 85–91
8. Saaty, T.L.: Decision making with the analytic hierarchy process. International Journal of Services Sciences **1**(1) (2008)
9. Saaty, T.L.: How to make a decision: The analytic hierarchy process. European Journal of Operational Research **48**(1) (1990) 9–26
10. Saaty, T.L.: A scaling method for priorities in hierarchical structures. Journal of Mathematical Psychology **15**(3) (1977) 234–281
11. A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner: A survey of mobile malware in the wild. In ACM, ed.: 1st ACM workshop on Security and privacy in smartphones and mobile devices (SPSM11). (2011) 3 – 14
12. R. Cannings: An update on Android Market security (2011) http://googlemobile.blogspot.com/2011/03/update-on-android-market-security.html.
13. W. Enck, M. Ongtang, P. McDaniel: On Lightweight Mobile Phone Application Certification. In ACM, ed.: 16th ACM conference on Computer and Communications Security (CCS'09). (2009) 235 – 254
14. M. Nauman, S. Khan, X. Zhang: Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints. In ACM, ed.: 5th ACM Symposium on Information Computer and Communication Security (ASIACCS'10). (2010)
15. D. Barrera, H.G. Kayacik, P.C. van Oorschot, A. Somayaji: A Methodology for Empirical Analysis of Permission-Based Security Models and its Application to Android. In ACM, ed.: 17th ACM Conference on Computer and Communications Security (CCS'10. (2010)