Android Run-time Permission Exploitation User Awareness by Means of Formal Methods

Fausto Fasano[1], Fabio Martinelli[2], Francesco Mercaldo[1,2] and Antonella Santone[1]
[1]*Department of Biosciences and Territory, University of Molise, Pesche (IS), Italy*
[2]*Institute for Informatics and Telematics, National Research Council of Italy, Pisa, Italy*

Keywords: Mobile, Malware, Permission, Security, Model Checking, Formal Methods, Android.

Abstract: Our mobile devices store a lot of sensitive and critical information. Moreover, considering the ability of smartphones and tables to detect the position and to record audio, it is not absolutely an exaggeration to admit that potentially our devices can easily spy on us. The ability to perform these crucial tasks must be granted by the user at the time of their first use by accepting the so-called permissions. The problem is that in Android, once these permissions are granted, the application can always use them without notifying the user. In this paper, we propose a methodology, based on formal methods, aimed to detect the exact point in the code (in term of package, class and method) of an Android application where a permission is invoked at run-time. Moreover we design a tool able to advise the user whether a permission is invoked, in this way the user can be informed about the application behaviour.

1 INTRODUCTION AND RELATED WORK

In last years, mobile malware increased its complexity from the point of view of malicious actions able to steal more and more sensitive and private information. In particular, the Android official market, due to the official market publication process, it is the perfect vector to diffuse malware (Mercaldo et al., 2016c; Mercaldo et al., 2016d). Another motivation behind the increasing amount of Android malware in the wild is represented by the easiness of requesting permissions to sensitive resources such as, for example, the camera and the microphone. As a matter of fact, the Android platform employs the permission system to restrict application privileges to secure the sensitive resources of the users. An application needs to get a users approval for the requested permissions to access the privacy-relevant resources. Thus, the permission system was designed to protect users from applications with invasive behaviours, but its effectiveness highly depends on the users comprehension of permission approval (Tchakounté, 2014; Martinelli et al., 2017a; Fasano et al., 2019b). The developer is responsible for determining appropriately which permissions an application requires. According to (Kelley et al., 2012; Felt et al., 2012; Tchakounté et al., 2014; Martinelli et al., 2017b), lot of users do not understand what each permission means and blindly grant them, allowing the application to access sensitive information of the user. Many users, although an app might request a suspicious permission among many seemingly legitimate permissions, still confirm the installation. Most of permissions defined by Google are coarse-grained. Especially, the INTERNET permission (Barrera et al., 2010), the READ PHONE STATE permission (Pearce et al., 2012), and the WRITE SETTINGS permission (Jeon et al., 2011) are coarse-grained as they give to an application arbitrary access to certain resources. The INTERNET permission allows an application to send HTTP(S) requests to all domains, and connect to arbitrary destinations and ports (Felt et al., 2010). As a result, the INTERNET permission provides insufficient expressiveness to enforce control over the Internet accesses of the application (Canfora et al., 2016). Because of the previous problems, researchers have been involved to determine mechanisms that employs individual permissions and the combination of permissions to detect and characterise malware (Enck et al., 2014; Zhou and Jiang, 2012). The Android framework, starting from Android 7, allows the user to approve a subset of the permissions requested by appli-

Fasano, F., Martinelli, F., Mercaldo, F. and Santone, A.
Android Run-time Permission Exploitation User Awareness by Means of Formal Methods.
DOI: 10.5220/0009372308040814
In *Proceedings of the 6th International Conference on Information Systems Security and Privacy (ICISSP 2020)*, pages 804-814
ISBN: 978-989-758-399-5; ISSN: 2184-4356

cations[1] (even though some applications do not work if not all permissions are provided) but this mechanism is not enough to guarantee the security of our information stored on mobile devices (Chakraborty and Chattopadhyay, 2020; Foster, 2020).

As such, Enck et al. (Enck et al., 2008) introduce a policy-based system called Kirin to detect malware at install time based on undesirable combination of permissions. Diverse works evaluate the detection of malware with permissions using machine learning on Android (Huang et al., 2013). They all realise that a permission-based mechanism can be used as a quick filter to identify malicious applications. Zhou and Jiang (Zhou and Jiang, 2012) characterise Android applications (both normal and malware applications) with individual permissions focusing on the number of occurrences of permissions in those groups. Authors in (Tramontana and Verga, 2019) propose a method to protect data and resources in Android. basically developing a method aimed to ask user when a permission in invoked. The main difference with respect to the proposed work is represented by the adoption of formal verification environment, that demonstrate their efficacy and robustness in the detection of malicious behaviour in software and in particular in Android environment (Canfora et al., 2018).

In this context, the permission to use the device camera (*android.permission.CAMERA*) and to record audio (*android.permission.RECORD_AUDIO*) exploiting the device microphone are really critical. In fact, the *android.permission.RECORD_AUDIO* permission can easily turn the smartphone into an environmental recorder, while the *android.permission.CAMERA* permission is able to capture pictures and videos. Clearly, these permissions can be used for purposes useful to the user (for instance, for a voip call) but can be easily exploited for malicious purposes from malware writers.

Basically, the purpose of a permission is to protect the privacy of an Android user. Android apps must request permission to access sensitive user data (such as contacts and SMS), as well as certain system features (such as camera and internet). Depending on the feature, the system might grant the permission automatically or might prompt the user to approve the request [2].

A central design point of the Android security architecture is that no app, by default, has permission to perform any operations that would adversely impact other apps, the operating system, or the user. This includes reading or writing the user's private data (such as contacts or emails), reading or writing another app's files, performing network access, keeping the device awake, and so on.

Android basically considers two categories of permissions: the normal permissions, i.e., the permission that the system automatically grants to the mobile app and the dangerous permissions.

If the app lists dangerous permissions in its manifest (that is, permissions that could potentially affect the user's privacy or the device's normal operation), such as the SEND_SMS permission, but also the *android.permission.RECORD_AUDIO* or the *android.permission.CAMERA*, the user at run-time must explicitly agree to grant those permissions. Once the user grants the permission, the app is able to invoke it whenever and wherever in the code.

For this reason even if an app requires a permission just one time, the permission will be released from the operating system forever.

Starting from these considerations, in this paper we propose a method to automatically detect whether an Android permission is invoked at run-time. The final aim is to show to the user the permission invocation each time it is requested by the mobile application. Android considers several permission as dangerous, in this work we only focus on the permission requests to access the internet connection, to use the device camera and to record audio through the device microphone.

To detect whether a permission is requested we exploit formal methods, in particular the model checking technique. We model an Android applications and with mu-calculus properties we verify whether a mobile app is asking for the camera and audio recording dangerous permissions by using static analysis.

The paper proceeds as follows: in the next section we provide background notions about the model checking and mu-calculus logic largely employed in this paper, in Section 3 the proposed method is described, in Section 4 we present a real-world experiment aimed to demonstrate the effectiveness of the proposed method, in Section 5 a tool to inform the user when an application is asking a permission at run-time is introduced and, finally, in last section conclusion and future research directions are drawn.

[1] https://developer.android.com/training/permissions/requesting

[2] https://developer.android.com/guide/topics/permissions/overview

2 MODEL CHECKING AND MU-CALCULUS LOGIC BACKGROUND

Verification of a software or hardware system involves checking whether the system in question behaves as it was designed to behave. Formal methods have been successfully applied to safety-critical systems. One reason is the overwhelming evidence that formal methods do result in safer systems. In this paper, we show that formal methods are extremely well-suited to spyware detection. First of all, in this section we recall some basic concepts.

Model checking is an formal method for determining if a model of a system satisfies a correctness specification (Clarke et al., 2001). A model of a system consists of a labelled transition system (LTS). A specification or property is a logical formula. A model checker then accepts two inputs, a LTS and a temporal formula, and returns *true* if the system satisfies the formula and *false* otherwise.

A labelled transition system comprises some number of states, with arcs between them labelled by activities of the system. A LTS is specified by:

- A set S of states;
- A set L of labels or actions;
- A set of transitions $T \subseteq S \times L \times S$.

Transitions are given as triples $(start, label, end)$.

In this paper, to express proprieties of the system we use the modal mu-calculus (Stirling, 1989) which is one of the most important logics in model checking.

The syntax of the mu-calculus is the following, where K ranges over sets of actions (i.e., $K \subseteq L$) and Z ranges over variables:

$$\varphi ::= \texttt{tt} \mid \texttt{ff} \mid Z \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid [K]\varphi \mid \langle K \rangle \varphi \mid \nu Z.\varphi \mid \mu Z.\varphi$$

A fixpoint formula may be either $\mu Z.\varphi$ or $\nu Z.\varphi$ where μZ and νZ *binds* free occurrences of Z in φ. An occurrence of Z is free if it is not within the scope of a binder μZ (resp. νZ). A formula is *closed* if it contains no free variables. $\mu Z.\varphi$ is the least fixpoint of the recursive equation $Z = \varphi$, while $\nu Z.\varphi$ is the greatest one. From now on we consider only closed formulae.

Scopes of fixpoint variables, free and bound variables, can be defined in the mu-calculus in analogy with variables of first order logic.

The satisfaction of a formula φ by a state s of a transition system is defined as follows:

- Each state satisfies `tt` and no state satisfies `ff`;
- A state satisfies $\varphi_1 \vee \varphi_2$ ($\varphi_1 \wedge \varphi_2$) if it satisfies φ_1 or (and) φ_2. $[K]\,\varphi$ is satisfied by a state which, for every performance of an action in K, evolves to a state obeying φ. $\langle K \rangle\,\varphi$ is satisfied by a state which can evolve to a state obeying φ by performing an action in K.

For example, $\langle a \rangle\,\varphi$ denotes that there is an a-successor in which φ holds, while $[a]\,\varphi$ denotes that for all a-successors φ holds.

The precise definition of the satisfaction of a closed formula φ by a state s (written $s \models \varphi$) is given in Table 1.

A fixed point formula has the form $\mu Z.\varphi$ ($\nu Z.\varphi$) where μZ (νZ) *binds* free occurrences of Z in φ. An occurrence of Z is free if it is not within the scope of a binder μZ (νZ). A formula is *closed* if it contains no free variables. $\mu Z.\varphi$ is the least fix-point of the recursive equation $Z = \varphi$, while $\nu Z.\varphi$ is the greatest one.

A transition system $\mathcal{T}$ satisfies a formula ϕ, written $\mathcal{T} \models \phi$, if and only if $q \models \phi$, where q is the initial state of $\mathcal{T}$.

We will use the following abbreviations:

$$\begin{aligned}
\langle \alpha_1, \ldots, \alpha_n \rangle\,\phi &= \langle \{\alpha_1, \ldots, \alpha_n\} \rangle\,\phi \\
\langle - \rangle\,\phi &= \langle L \rangle\,\phi \\
\langle -K \rangle\,\phi &= \langle L - K \rangle\,\phi \\
[\alpha_1, \ldots, \alpha_n]\,\phi &= [\{\alpha_1, \ldots, \alpha_n\}]\,\phi \\
[-]\,\phi &= [L]\,\phi \\
[-K]\,\phi &= [L - K]\,\phi
\end{aligned}$$

Some examples of logic properties are now provided. The simplest formulae are just those of modal logic:

$$\langle \alpha \rangle\,\texttt{tt}$$

means that "there exists a transition labelled by the action α".

If we use the two different fixpoints, we obtain the following formulae:

$\mu Z.\,[\alpha]\,Z$: "all the sequences of α-transitions are finite".
$\nu Y.\,\langle a \rangle\,Y$: "there is an infinite sequence of α-transitions".

We can then add a predicate p, and obtain the formula:

$$\nu Y.p \wedge \langle \alpha \rangle\,Y$$

saying that "there is an infinite sequence of α-transitions, and all states in this sequence satisfy p".

With two fixpoints, we can write fairness formulae, such as:

$$\nu Y.\mu X.(p \wedge \langle \alpha \rangle\,Y) \vee \langle \alpha \rangle\,X$$

meaning that "on some α-path there are infinitely many states where p holds".

Changing the order of fixpoints we obtain:

Table 1: Satisfaction of a closed formula by a state.

$p \not\models \texttt{ff}$		
$p \models \texttt{tt}$		
$p \models \varphi \wedge \psi$	iff	$p \models \varphi$ and $p \models \psi$
$p \models \varphi \vee \psi$	iff	$p \models \varphi$ or $p \models \psi$
$p \models [K]_R\,\varphi$	iff	$\forall p'.\forall \alpha \in K.p \xrightarrow{\alpha}_{K \cup R} p'$ implies $p' \models \varphi$
$p \models \langle K \rangle_R\,\varphi$	iff	$\exists p'.\exists \alpha \in K.p \xrightarrow{\alpha}_{K \cup R} p'$ and $p' \models \varphi$
$p \models \nu Z.\varphi$	iff	$p \models \nu Z^n.\varphi$ for all n
$p \models \mu Z.\varphi$	iff	$p \models \mu Z^n.\varphi$ for some n

where:

- for each n, $\nu Z^n.\varphi$ and $\mu Z^n.\varphi$ are defined as:

$$\nu Z^0.\varphi = \texttt{tt} \qquad \mu Z^0.\varphi = \texttt{ff}$$
$$\nu Z^{n+1}.\varphi = \varphi[\nu Z^n.\varphi/Z] \qquad \mu Z^{n+1}.\varphi = \varphi[\mu Z^n.\varphi/Z]$$

where the notation $\varphi[\psi/Z]$ indicates the substitution of ψ for every free occurrence of the variable Z in φ.

$$\mu X.\nu Y.(p \wedge \langle \alpha \rangle\, Y) \vee \langle \alpha \rangle\, X$$

saying "on some α-path almost always p holds."

In this paper, we use CAAL (Concurrency Workbench, Aalborg Edition) (Andersen et al., 2015) as formal verification environment. It is one of the most popular environments for verifying systems. In the CAAL the verification of temporal logic formulae is based on model checking (Clarke et al., 2001).

3 THE METHODOLOGY

The proposed method is aimed to detect whether a sensitive permission is invoked by exploiting model checking with the aim to alert the user whenever the permission is invoked at run-time.

In this section, we present the application of a model checking based approach to detect run-time permission invocation in Android environment. As stated in the introduction, we are focused on Android samples with the ability to use the internet connection, the device camera and microphone to record audio. We describe mobile applications by exploiting Milner's Calculus of Communicating Systems (CCS) (Milner, 1989) language specification and express behavioural properties using mu-calculus branching temporal logic (Stirling, 1989). The methodology, as explained in previous works (Battista et al., 2016; Canfora et al., 2018; Cimitile et al., 2018; Mercaldo et al., 2016a; Fasano et al., 2019a; Bernardeschi et al., 2019), is based on two main steps.

Figure 1 shows the proposed methodology main architecture.

The first step generates a CCS specification starting from *.class* files of the analyzed application written in Java Bytecode. Then, we define a Java Bytecode-to-CCS transformation function. This is defined for each instruction of the Java Bytecode. It directly translates the Java Bytecode instructions into CCS process specifications. The second step aims to investigate Android application behaviours which are successively expressed using mu-calculus logic. We specify the set of properties starting from current literature (Zhou and Jiang, 2012; Mercaldo et al., 2016b) and with the manual inspection of a few samples. The CCS processes obtained in the first step are then used to verify the properties. Codes described as CCS processes are first mapped to labelled transition systems and then verified with a model checker. In our approach, we invoke the Concurrency Workbench of New Century (CWB-NC) (Cleaveland and Sims, 1996) as formal verification environment. When the result of the CWB-NC model checker is *true*, it means that the sample under analysis is malicious, *false* otherwise. We formulated logic rules from the following characterizing behaviours (typical in mobile applications): (i) the exploitation of an internet connection, (ii) the request to use the device camera, (iii) the request to use the device microphone to record audio. The distinctive features of our methodology are:

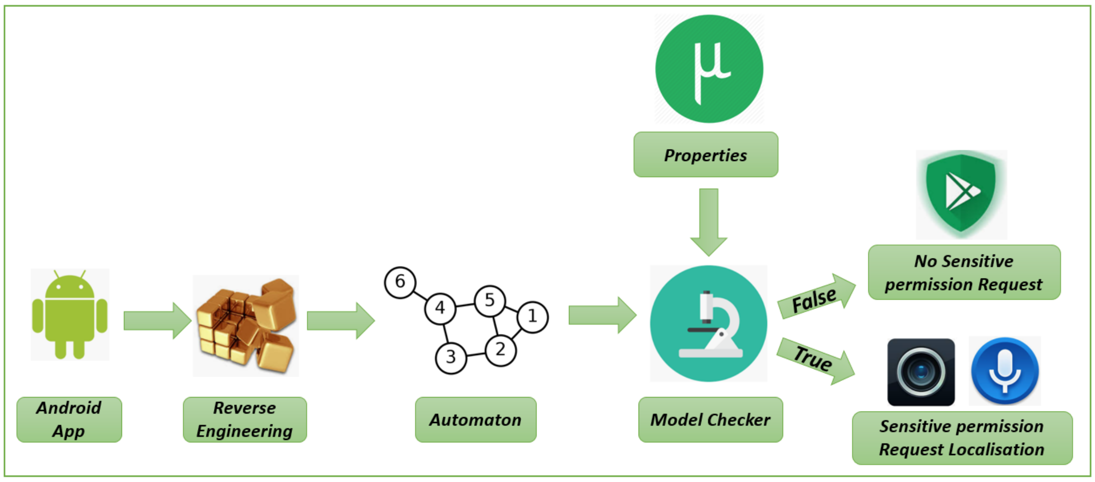


Figure 1: The proposed methodology.

(i) the use of formal methods; (ii) the inspection on Java Bytecode and not on the source code; (iii) the use of static analysis; (iv) the capture of malicious behaviours at a finer granularity. Summing up the methodology, starting from the Java Bytecode application files, we derive CCS processes, which are successively used for checking properties expressing the most common behaviours exhibit by the family samples. Performing automatic analysis on the Bytecode instead of the source code has several advantages: (i) independence of the source programming language; (ii) identification of malicious payload without decompilation even when source code is lacking; (iii) ease of parsing a lower-level code; (iv) independence from obfuscation (Canfora et al., 2018; Cimitile et al., 2018).

In the follow we show a couple of code snippet we analysed to formulate the temporal logic properties for the detection of run-time permissions.

Let us the consider the code snippet shown in Figure 2.

This snippet is belongong to a sample identified by the *da298299c164d2584c4fee96e205bae59e35593b* SHA-1 hash i.e., the *WhatsApp Messenger* app [3], one of the most famous app to send and receive messages, calls, photos, videos, documents, and Voice Messages using the 4G/3G/2G/EDGE connection or Wi-Fi, as available. In detail the snippet shown in Figure 2 is related to the Java code of the *A11* method of the *VoipActivityV2* class belonging to the *com.whatsapp.voipcalling* package. In the snippet there are two permission invocations: the first one is the request for audio recording (i.e., the *android.permission.RECORD_AUDIO* request) and the second one is the permission to use the device camera (i.e., the *android.permission.CAMERA* request).

With the aim to consider another run-time request permission, let us consider the code snippet in Figure 3.

The snippet in Figure 3 is related to a sample identified by the *2f03011c9f7e34ad46d9a26ef47d54b38a4caa49* SHA-1 hash i.e., the *Yahoo Weather* app[4], one of the most widespread app for hourly, 5-day, and 10-day forecasts with details about wind, pressure, and chance of precipitation. In detail the snippet in Figure 3 is related to the Java code of the *X* method of the *bv* class. In this snippet, in the 4-th row, there is the invocation of the *android.permission.INTERNET* permission. Both the analised applications were downloaded from the APKCombo repository[5].

Starting from the snippets in Figures 2 and 3, we propose several formulae: the first one (i.e., ψ) aimed to detect whether there is a request for the *android.permission.RECORD_AUDIO* permission, the second one (i.e., χ) aimed to detect whether there is a request for the *android.permission.CAMERA* permission. The third formula, i.e., φ is aimed to verify whether there is a request for the *android.permission.RECORD_AUDIO* or for the *android.permission.CAMERA* permission. The last formula i.e., ζ, is related to the invocation of the *android.permission.INTERNET* permission at run-time.

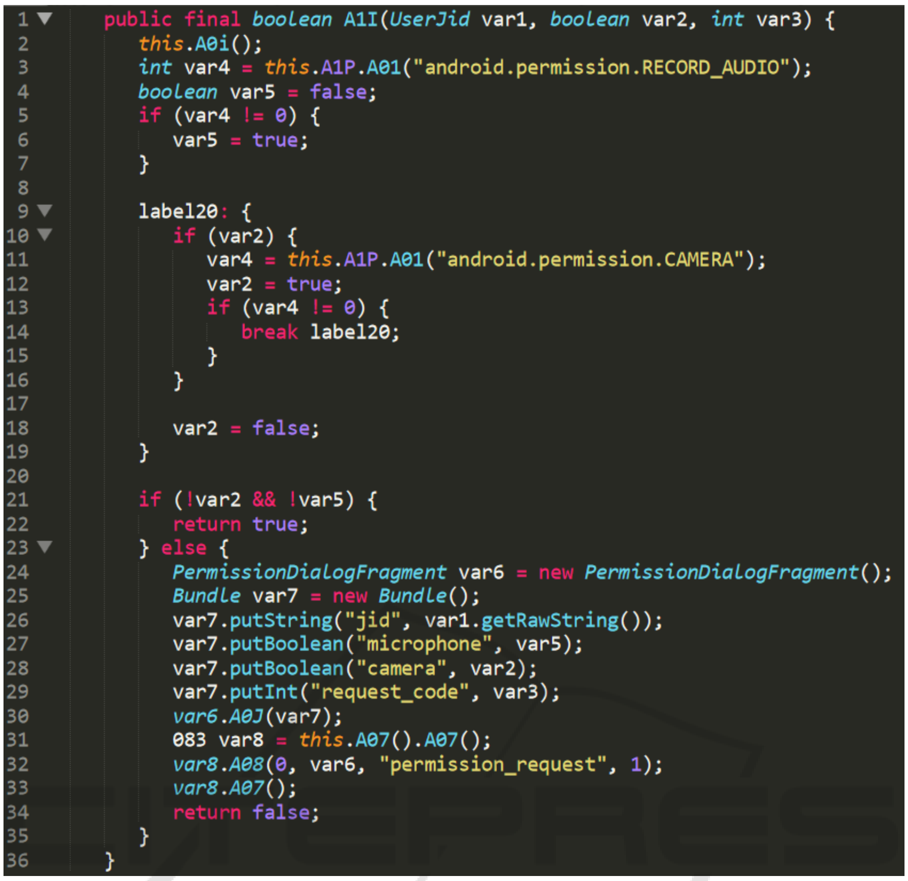
The ψ, χ, φ and ζ are satisfied whether at least in one method of the application under analysis there is an invocation of the considered permission.

The output of the proposed methodology is the list

[3]https://play.google.com/store/apps/details?id=com.whatsapp&hl=it

[4]https://play.google.com/store/apps/details?id=com.yahoo.mobile.client.android.weather&hl=it

[5]https://apkcombo.com/



```
public final boolean A1I(UserJid var1, boolean var2, int var3) {
    this.A0i();
    int var4 = this.A1P.A01("android.permission.RECORD_AUDIO");
    boolean var5 = false;
    if (var4 != 0) {
        var5 = true;
    }

    label20: {
        if (var2) {
            var4 = this.A1P.A01("android.permission.CAMERA");
            var2 = true;
            if (var4 != 0) {
                break label20;
            }
        }

        var2 = false;
    }

    if (!var2 && !var5) {
        return true;
    } else {
        PermissionDialogFragment var6 = new PermissionDialogFragment();
        Bundle var7 = new Bundle();
        var7.putString("jid", var1.getRawString());
        var7.putBoolean("microphone", var5);
        var7.putBoolean("camera", var2);
        var7.putInt("request_code", var3);
        var6.A0J(var7);
        O83 var8 = this.A07().A07();
        var8.A08(0, var6, "permission_request", 1);
        var8.A07();
        return false;
    }
}
```

Figure 2: A code snippet belonging to the WhatsApp Messenger application.

of packages, classes and methods where the considered permission is invoked.

4 EXPERIMENTAL ANALYSIS

In this section, we describe the data-set used in the experimental evaluation and we present the achieved results.

4.1 The Data-set

To evaluate the proposed methodology to detect sensitive permission request we consider a data-set of real-world Android application obtained from the Google Play store[6]. To this aim, we crawled 200 applications from the official Android market belonging to several categories. In particular, within the data-set, 50 applications require the *android.permission.RECORD_AUDIO* permission, 50 applications require the *android.permission.CAMERA* permission, 50 applications require the *android.permission.INTERNET* permission at run-time. The remaining 50 applications do not require these permissions at run-time.

[6]https://play.google.com/store/apps

4.2 Performance Evaluation

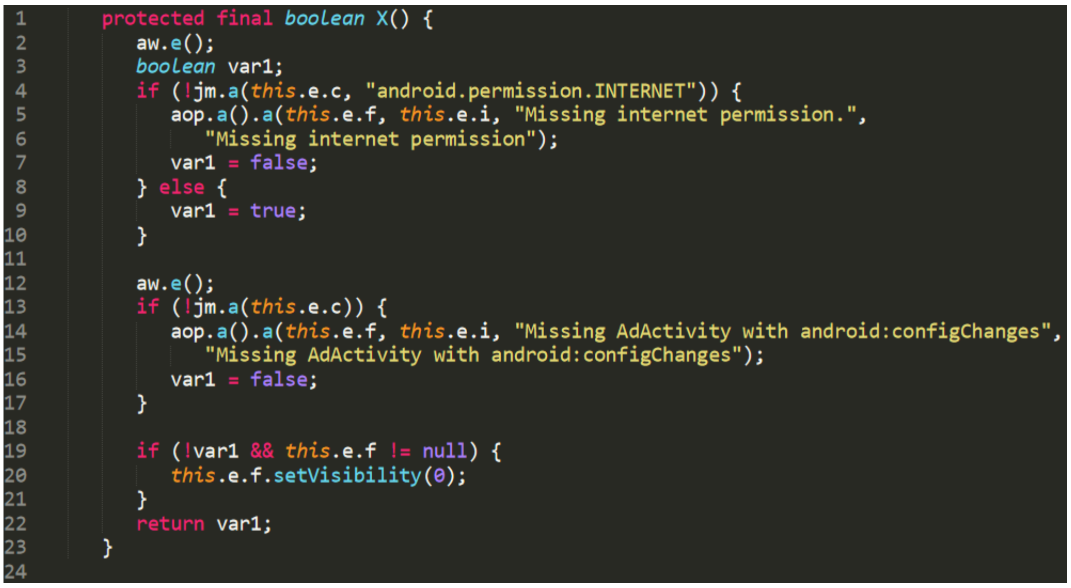
We consider four metrics in order to evaluate the results of the classification: Precision, Recall, F-Measure and Accuracy.

The precision has been computed as the proportion of the examples that truly belong to class X among all those which were assigned to the class. It is the ratio of the number of relevant records retrieved to the total number of irrelevant and relevant records retrieved:

$$Precision = \frac{tp}{tp+fp}$$

where tp indicates the number of true positives and fp indicates the number of false positives.

The recall has been computed as the proportion



```
protected final boolean X() {
    aw.e();
    boolean var1;
    if (!jm.a(this.e.c, "android.permission.INTERNET")) {
        aop.a().a(this.e.f, this.e.i, "Missing internet permission.",
            "Missing internet permission");
        var1 = false;
    } else {
        var1 = true;
    }

    aw.e();
    if (!jm.a(this.e.c)) {
        aop.a().a(this.e.f, this.e.i, "Missing AdActivity with android:configChanges",
            "Missing AdActivity with android:configChanges");
        var1 = false;
    }

    if (!var1 && this.e.f != null) {
        this.e.f.setVisibility(0);
    }
    return var1;
}
```

Figure 3: A code snippet belonging to the Yahoo Weather application.

of examples that were assigned to class X, among all the examples that truly belong to the class, i.e., how much part of the class was captured. It is the ratio of the number of relevant records retrieved to the total number of relevant records:

$$Recall = \frac{tp}{tp+fn}$$

where *tp* indicates the number of true positives and *fn* indicates the number of false negatives.

The F-Measure is a measure of a test accuracy. This score can be interpreted as a weighted average of the precision and recall:

$$F\text{-}Measure = 2 * \frac{Precision*Recall}{Precision+Recall}$$

The Accuracy is defined as the ratio of number of correct predictions to the total number of input samples:

$$Accuracy = \frac{tp+tn}{tp+tn+fp+fn}$$

where *tp* indicates the number of true positives, *tn* indicates the number of true negatives, *fp* indicates the number of false positives and *fn* indicates the number of false negatives.

Table 2 shows the results we obtained for the computed metrics.

Table 2: Classification Results.

Formula	Precision	Recall	F-measure	Accuracy
φ	1	1	1	1
ζ	1	1	1	1

From the results shown in Table 2 it emerges that for both the φ and the ζ formulae, the proposed methodology is able to correctly detect and localize the code snippet where a permission is invoked at run-time.

5 TOOL DESIGN

In this section we describe the design of the tool we propose to inform the user about the permission that the code running on its mobile device is requiring (see Figure 4).

Starting from the output of the analysis of the proposed method, the idea is to advise - at run-time - the user, by showing her the permission that the Android operating system is releasing, as shown in Figures 5 and 6.

Through a reverse engineering process (Fasano et al., 2019c; Cimitile et al., 2017; Martinelli et al., 2018) from the Android application the source code of the application under analysis is obtained.

From the output of the proposed methodology (shown in Figure 1) the set of the methods with run-time permission is obtained: in particular from the proposed methodology we are able to obtain the package, the class and the method where a permission is invoked at run-time. In each of these methods a Toast[7] is added. In Android, a Toast is a graphical interface element able to provide simple feedback about an op-

[7]https://developer.android.com/guide/topics/ui/notifiers/toasts

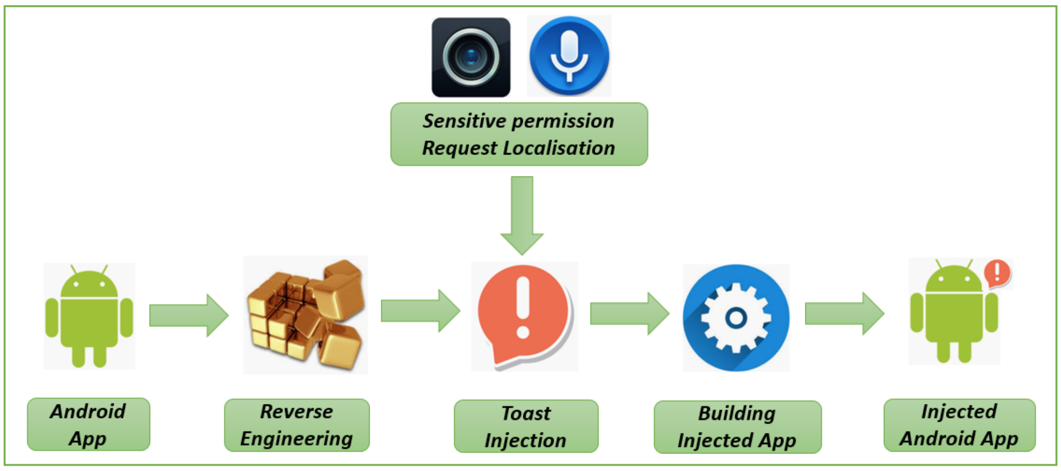


Figure 4: The tool design.

eration in a small popup. It only fills the amount of space required for the message and the current activity remains visible and interactive and automatically disappears after a timeout.

Once the small popups for each method invoking at run-time a permission are injected, the application is built with the aim to obtain the app executable. In this way the user can be advised each time the app is asking for a permission at run-time.

To show how the designed tool can be helpful to make aware the user about the permissions requested by the application, we show two screenshots about two different applications (i.e, *Whatsapp Messenger* and *Yahoo Weather*) when the Toasts are injected. In these examples we manually injected the Toast in the code localised by the proposed methodology, with the aim to show the usefulness of the designed tool exploiting the methodology proposed in this work.

Figure 5 shows the *Whatsapp Messenger* screen when the user is trying the make a voice call.

From the screenshot in Figure 5 it is possible to see the injected Android Toast advising the user that the app is requiring for the *android.permission.RECORD_AUDIO* permission.

The second screenshot we propose is related to the Yahoo Weather application and it is shown in Figure 6.

The screenshot in Figure 6 is related to the home screen of the device, where it is visible to *Yahoo Weather* widget. In Android, widgets are miniature application views that can be embedded in other applications (for instance the Home screen) and receive periodic updates. Clearly also widget can ask for run-time permission, as in this case: in fact when the user try to update the weather information, the widget is requiring the *android.permission.INTERNET* as

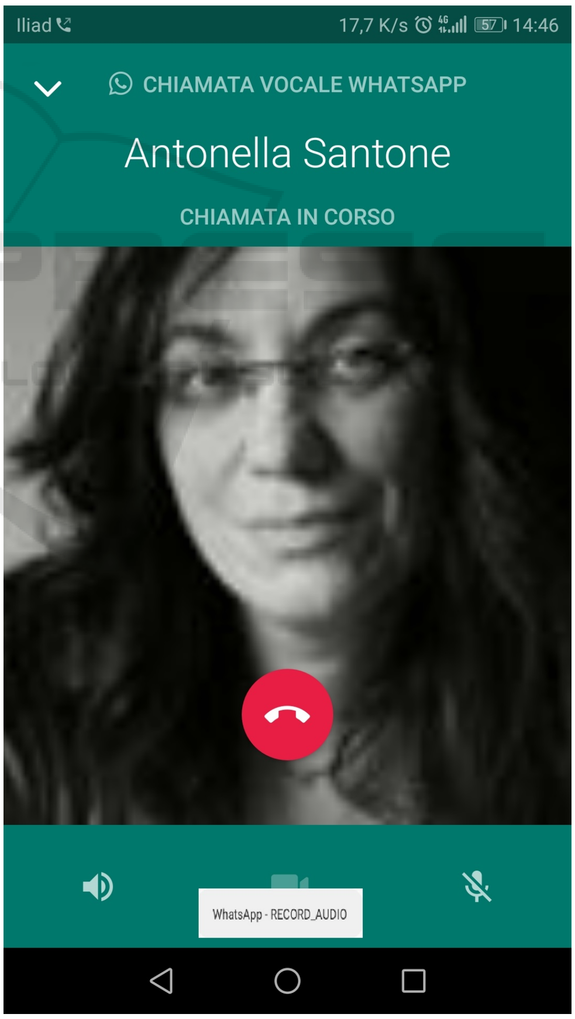


Figure 5: A screenshot of the WhatsApp application when the *android.permission.RECORD_AUDIO* permissions is requested.

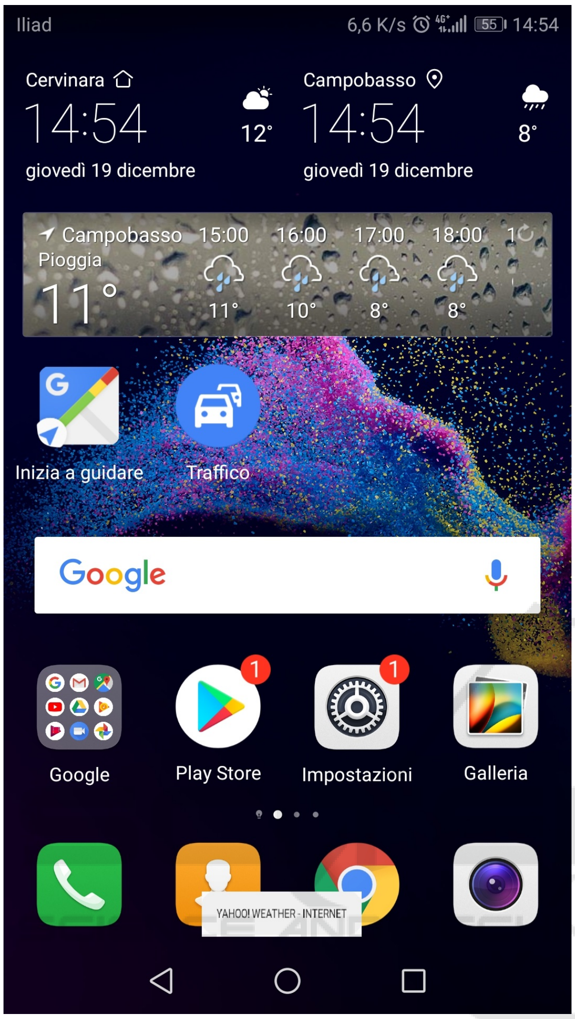


Figure 6: A screenshot of the *Yahoo Weather* application when the *android.permission.INTERNET* permission is requested.

demonstrated by the message appeared in the Toast.

6 CONCLUSION AND FUTURE WORK

In this paper, we proposed a static methodology aimed to detect the usage of run-time permission in Android environment. We model Android applications in term of CCS models and, by exploiting model checking technique, we verify whether a set of permission in invoked. In this work we focused on three permissions i.e., *android.permission.CAMERA*, *android.permission.RECORD_AUDIO* and *android.permission.INTERNET* permissions. We experiment a data-set composed by 200 applications downloaded by Google Play, obtaining an accuracy equal to 1. Moreover we propose a tool, based on the results obtained by the proposed methodology, aimed to inform the user at run-time when an application is asking for permissions.

As future work, first of all we plan to implement and to release the designed tool in the Android official market. Moreover, we plan to extend the temporal logic formula set (Francesco et al., 2014; Ceccarelli et al., 2014; Cimitile et al., 2018; Canfora et al., 2018; Santone, 2002; Santone, 2011; Barbuti et al., 2005; Gradara et al., 2005), with the aim to detect the full set of permission provided at run-time by the Android framework.

ACKNOWLEDGMENTS

This work has been partially supported by MIUR - SecureOpenNets and EU SPARTA and CyberSANE projects, the Formal Methods for IT Security Lab[8], and the MOSAIC Research Center[9] at the University of Molise.

REFERENCES

Andersen, J. R., Andersen, N., Enevoldsen, S., Hansen, M. M., Larsen, K. G., Olesen, S. R., Srba, J., and Wortmann, J. K. (2015). CAAL: concurrency workbench, aalborg edition. In *Theoretical Aspects of Computing - ICTAC 2015 - 12th International Colloquium Cali, Colombia, October 29-31, 2015, Proceedings*, volume 9399 of *Lecture Notes in Computer Science*, pages 573–582. Springer.

Barbuti, R., De Francesco, N., Santone, A., and Vaglini, G. (2005). Reduced models for efficient ccs verification. *Formal Methods in System Design*, 26(3):319–350.

Barrera, D., Kayacik, H. G., Van Oorschot, P. C., and Somayaji, A. (2010). A methodology for empirical analysis of permission-based security models and its application to android. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 73–84. ACM.

Battista, P., Mercaldo, F., Nardone, V., Santone, A., and Visaggio, C. A. (2016). Identification of android malware families with model checking. In *ICISSP*, pages 542–547.

Bernardeschi, C., Mercaldo, F., Nardone, V., and Santone, A. (2019). Exploiting model checking for mobile botnet detection. *Procedia Computer Science*, 159:963–972.

[8]https://dipbioter.unimol.it/ricerca/laboratori/metodi-formali-per-la-sicurezza-informatica/

[9]https://dipbioter.unimol.it/ricerca/laboratori/centro-di-ricerca-mosaic/

Canfora, G., Martinelli, F., Mercaldo, F., Nardone, V., Santone, A., and Visaggio, C. A. (2018). Leila: formal tool for identifying mobile malicious behaviour. *IEEE Transactions on Software Engineering*.

Canfora, G., Medvet, E., Mercaldo, F., and Visaggio, C. A. (2016). Acquiring and analyzing app metrics for effective mobile malware detection. In *Proceedings of the 2016 ACM International Workshop on International Workshop on Security and Privacy Analytics*. ACM.

Ceccarelli, M., Cerulo, L., and Santone, A. (2014). De novo reconstruction of gene regulatory networks from time series data, an approach based on formal methods. *Methods*, 69(3):298–305.

Chakraborty, D. and Chattopadhyay, M. (2020). Assignment tracking on android platform. In *Information and Communication Technology for Sustainable Development*, pages 491–499. Springer.

Cimitile, A., Martinelli, F., Mercaldo, F., Nardone, V., and Santone, A. (2017). Formal methods meet mobile code obfuscation identification of code reordering technique. In *2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 263–268. IEEE.

Cimitile, A., Mercaldo, F., Nardone, V., Santone, A., and Visaggio, C. A. (2018). Talos: no more ransomware victims with formal methods. *International Journal of Information Security*, 17(6):719–738.

Clarke, E. M., Grumberg, O., and Peled, D. A. (2001). *Model checking*. MIT Press.

Cleaveland, R. and Sims, S. (1996). The NCSU concurrency workbench. In *Computer Aided Verification, 8th International Conference, CAV '96, New Brunswick, NJ, USA, July 31 - August 3, 1996, Proceedings*, pages 394–397.

Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P., and Sheth, A. N. (2014). Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):5.

Enck, W., Ongtang, M., and McDaniel, P. (2008). Mitigating android software misuse before it happens.

Fasano, F., Martinelli, F., Mercaldo, F., Nardone, V., and Santone, A. (2019a). Spyware detection using temporal logic. In *5th International Conference on Information Systems Security and Privacy, ICISSP 2019*, pages 690–699. SciTePress.

Fasano, F., Martinelli, F., Mercaldo, F., and Santone, A. (2019b). Energy consumption metrics for mobile device dynamic malware detection. *Procedia Computer Science*, 159:1045–1052.

Fasano, F., Martinelli, F., Mercaldo, F., and Santone, A. (2019c). Investigating mobile applications quality in official and third-party marketplaces. In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering*, pages 169–178. SCITEPRESS-Science and Technology Publications, Lda.

Felt, A. P., Greenwood, K., and Wagner, D. (2010). The effectiveness of install-time permission systems for third-party applications. *University of California at Berkely, Electrical Engineering and Computer Sciences, Technical report*.

Felt, A. P., Ha, E., Egelman, S., Haney, A., Chin, E., and Wagner, D. (2012). Android permissions: User attention, comprehension, and behavior. In *Proceedings of the eighth symposium on usable privacy and security*, page 3. ACM.

Foster, J. (2020). Who decides what is allowed? user interactions and permissions use on android. *ACM SIGAda Ada Letters*, 39(1):71–71.

Francesco, N. d., Lettieri, G., Santone, A., and Vaglini, G. (2014). Grease: a tool for efficient "nonequivalence" checking. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 23(3):24.

Gradara, S., Santone, A., and Villani, M. L. (2005). Using heuristic search for finding deadlocks in concurrent systems. *Information and Computation*, 202(2):191–226.

Huang, C.-Y., Tsai, Y.-T., and Hsu, C.-H. (2013). Performance evaluation on permission-based detection for android malware. In *Advances in Intelligent Systems and Applications-Volume 2*, pages 111–120. Springer.

Jeon, J., Micinski, K. K., Vaughan, J. A., Reddy, N., Zhu, Y., Foster, J. S., and Millstein, T. (2011). Dr. android and mr. hide: Fine-grained security policies on unmodified android. Technical report.

Kelley, P. G., Consolvo, S., Cranor, L. F., Jung, J., Sadeh, N., and Wetherall, D. (2012). A conundrum of permissions: installing applications on an android smartphone. In *International conference on financial cryptography and data security*, pages 68–79. Springer.

Martinelli, F., Marulli, F., and Mercaldo, F. (2017a). Evaluating convolutional neural network for effective mobile malware detection. *Procedia computer science*, 112:2372–2381.

Martinelli, F., Mercaldo, F., Nardone, V., Santone, A., Sangaiah, A. K., and Cimitile, A. (2018). Evaluating model checking for cyber threats code obfuscation identification. *Journal of Parallel and Distributed Computing*, 119:203–218.

Martinelli, F., Mercaldo, F., and Saracino, A. (2017b). Bridemaid: An hybrid tool for accurate detection of android malware. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 899–901. ACM.

Mercaldo, F., Nardone, V., and Santone, A. (2016a). Ransomware inside out. In *Availability, Reliability and Security (ARES), 2016 11th International Conference on*, pages 628–637. IEEE.

Mercaldo, F., Nardone, V., Santone, A., and Visaggio, C. A. (2016b). Download malware? no, thanks: how formal methods can block update attacks. In *Proceedings of the 4th FME Workshop on Formal Methods in Software Engineering, FormaliSE@ICSE 2016, Austin, Texas, USA, May 15, 2016*, pages 22–28. ACM.

Mercaldo, F., Nardone, V., Santone, A., and Visaggio, C. A. (2016c). Ransomware steals your phone. formal methods rescue it. In *International Conference on Formal Techniques for Distributed Objects, Components, and Systems*, pages 212–221. Springer.

Mercaldo, F., Visaggio, C. A., Canfora, G., and Cimitile, A. (2016d). Mobile malware detection in the real world. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 744–746. ACM.

Milner, R. (1989). *Communication and concurrency*. PHI Series in computer science. Prentice Hall.

Pearce, P., Felt, A. P., Nunez, G., and Wagner, D. (2012). Addroid: Privilege separation for applications and advertisers in android. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pages 71–72. Acm.

Santone, A. (2002). Automatic verification of concurrent systems using a formula-based compositional approach. *Acta Informatica*, 38(8):531–564.

Santone, A. (2011). Clone detection through process algebras and java bytecode. In *IWSC*, pages 73–74. Citeseer.

Stirling, C. (1989). An introduction to modal and temporal logics for ccs. In *Concurrency: Theory, Language, And Architecture*, pages 2–20.

Tchakounté, F. (2014). Permission-based malware detection mechanisms on android: Analysis and perspectives. *Journal of Computer Science*, 1(2).

Tchakounté, F., Dayang, P., Nlong, J., and Check, N. (2014). Understanding of the behaviour of android smartphone users in cameroon: application of the security. *Open J. Inf. Secur. Appl*, pages 9–20.

Tramontana, E. and Verga, G. (2019). Mitigating privacy-related risks for android users. In *2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 243–248. IEEE.

Zhou, Y. and Jiang, X. (2012). Dissecting android malware: Characterization and evolution. In *Proceedings of 33rd IEEE Symposium on Security and Privacy (Oakland 2012)*.