Computers in Human Behavior 59 (2016) 9–17

Contents lists available at ScienceDirect

Computers in Human Behavior

journal homepage: www.elsevier.com/locate/comphumbeh

Full length article

Leveraging adjusted user behavior in the detection and prevention of outgoing malicious SMSs in Android devices

Abdelouahid Derhab [a, *], Kashif Saleem [a], Jalal Al-Muhtadi [a, b], Mehmet A. Orgun [c]

[a] Center of Excellence in Information Assurance (COEIA), King Saud University (KSU), Riyadh, Saudi Arabia
[b] College of Computer and Information Sciences (CCIS), King Saud University (KSU), Saudi Arabia
[c] Intelligent Systems Group (ISG), Department of Computing, Macquarie University, NSW 2109, Australia

ARTICLE INFO

Article history:
Received 22 October 2015
Received in revised form
13 January 2016
Accepted 18 January 2016
Available online xxx

Keywords:
Android
SMS malware
Detection
Prevention
User behavior
Usability

ABSTRACT

In this paper, we propose *OnDroid*, a prevention system to defend against outgoing malicious SMSs in Android devices. OnDroid is user-friendly as it considers the user's little understanding of the Android system. It also considers multiple threat scenarios and requires less interaction with the user. For each SMS-sending operation, OnDroid first checks if the mobile device state mismatches the user's behavior. If so, the operation is blocked. Otherwise, it is delayed for a while and the user is notified to confirm or reject the operation. If the SMS is considered normal, the user does not need to take any action and the SMS is sent when the delay expires. Efficiency analysis shows that malicious SMS operations might be missed when the user is unavailable. To deal with this issue, we propose a method by which the user behavior can be adjusted to achieve 100% of malicious SMS prevention. Formal analysis as well as comparative study show that OnDroid offers a good tradeoff between security efficiency and usability.

1. Introduction

The Android operating system is largely popular among mobile device users. According to some several reports (Mawston, July 2014; http://www.idc.com/prodse), Android dominated the Smartphone market with a share of 85% and 82.8% in the second quarter of 2014 and 2015 respectively. Due to this popularity, it has been highly targeted by malware developers, which aim at profiting financially by infecting a large number of mobile devices. A study (Maslennikov, February 2014) has shown that Android accounts for 98.05% of mobile threats. According to F-secure report (Labs, April 2014), 99% of the emerged threats, which were discovered in the first quarter of 2014, were designed to run on the Android operating system. F-secure also reported in 2013 that 81.1% of the mobile malware were financially-motivated and 97.14% of the latter targeted Android OS (Labs, November 2013). Most of the financially-motivated malware perform SMS-sending activities during their attacks. This fact was supported by a report published in 2014 (Labs, April 2014). The report showed that 83% of the Android malwares analyzed by F-secure were performing SMS-sending activities, also called SMS-based attacks. These attacks are charged to the victim's account without his/her consent or knowledge, and can take the following forms:

- *SMS Spam*: It is an unsolicited message sent from the compromised mobile device to advertise goods and products. The spammers send instructions to the compromised devices to launch SMS spam campaigns (Cloudmark, March 2014).
- *SMS Premium rate fraud*: The malware sends an SMS to a premium rate number, which results in transferring costly sums from the user's account to that of the cybercriminal (Labs, April 2014).
- *SMS malware propagation*: The malware sends an SMS containing links to malicious applications in order to infect the recipient devices.
- *SMS privacy attack*: Some malware secretly send through SMS sensitive information stored at the device to the attacker such as: bank account, phone identifiers (IMES, IMSI), and GPS location.

* Corresponding author.
E-mail addresses: abderhab@ksu.edu.sa (A. Derhab), kashif@ksu.edu.sa (K. Saleem), jalal@ccis.edu.sa (J. Al-Muhtadi), mehmet.orgun@mq.edu.au (M.A. Orgun).

http://dx.doi.org/10.1016/j.chb.2016.01.023
0747-5632/

- *SMS flooding attack*: Many unauthorized SMSs are generated to attack a target, which might lead to a Denial-of-Service (DoS) (Traynor et al., 2009).

The applications in Android use APIs to access resources such as: telephony, SMS and other network functions. These APIs are protected through a security mechanism called permissions. The application developer must define the list of requested permissions in the AndroidManifest.xml file. To complete the installation process of the application, the user must grant all the requested permissions. Otherwise, the installation fails. In Android 4.2 (Jelly Bean), when an application tries to send an SMS to a premium-rate number, the user is notified. However, a malware can hide the notification from the user. Starting from Android 4.4 (KitKat), the user has to select only one default SMS application, which is allowed to write to the SMS provider and receive SMSs. The other SMS applications can only read the SMS provider and are notified when an SMS is received. However, this configuration cannot completely prevent the SMS attacks. It has been reported in (Labs, April 2014) that 0.1% of the applications, which are received from Google's Play Store, were found malicious. Therefore, a user might download a compromised SMS application that sends both legitimate and malicious SMSs. If the user selects it as default, the malicious SMSs cannot be detected as they cannot be distinguished from the legitimate ones.

Most of the users have little understanding and knowledge of the Android permission policy. This was confirmed in a survey (Felt et al., 2012), which showed that only 17% of the users look at the requested permissions when installing applications. It has also been observed that developers request more permissions than they actually require (Felt, Chin, Hanna, Song, & Wagner, 2011). As legitimate and malicious applications can request the same types of permissions, it is often difficult for the users to determine during the installation process if the requested permissions are harmful or not. To deal with this issue, many complementary security mechanisms have been proposed to detect or prevent malicious activities. These mechanisms offer different levels of security efficiency and different user interaction durations, and sometimes require users to have some knowledge of Android security.

In order to help the user to accurately distinguish between legitimate and malicious SMS applications, we propose OnDroid, a prevention system against outgoing malicious SMSs in Android devices. In this paper, we consider two types of SMS threats: (a) SMSs generated by a malicious application without the users knowledge, and (b) a compromised SMS application that sends both legitimate and malicious SMSs. The main idea of OnDroid is to find any incoherence (or mismatch) between the user behavior and the Android device state.

The main contributions of the paper are as follows:

1. We determine the main features of the user behavior with respect to the mobile device.
2. We identify the main features characterizing the Android device state.
3. We propose an algorithm that detects any mismatches between the user behavior and the Android device state when performing SMS-sending operations. This algorithm can prevent the occurrence of the first SMS threat.
4. If a match is found, the user has to check that the SMS does not come from the second SMS threat. To do so, we delay the SMS-sending operation and the user is notified to intervene in case the SMS is malicious.
5. We analyze the resilience of OnDroid against attacks and evaluate its detection efficiency.
6. We propose adjustment rules for the user behavior in order to prevent 100% of malicious SMS operations.
7. We provide a comparative study that shows that OnDroid offers a good tradeoff between security efficiency and usability.

The rest of the paper is organized as follows. Section 2 discuses related work. In section 3, we present the system model. The detailed description of OnDroid is given in section 4. Security analysis and efficiency analysis are provided in section 5 and section 6 respectively. In section 7, we present a performance and usability study of security solutions against SMS malware. Finally, section 8 concludes the paper with a summary of our contributions.

2. Related work

The main existing approaches to detect an installed SMS malware on an Android device can be categorized into: *static analysis* and *dynamic analysis*. In the static analysis approaches (Arzt et al., 2014; Batyuk et al., 2011; Enck, Octeau, McDaniel, & Chaudhuri, 2011; Grace, Zhou, Wang, & Jiang, 2012; Schmidt et al., 2009; Seo, Gupta, Sallam, Bertino, & Yim, 2014; Suarez-Tangil, Tapiador, Peris-Lopez, & Blasco, 2014), the application static information (i.e., program syntax, structural properties), which are extracted from the source or the binary code, are used to check if the application contains malicious code. However, obfuscation techniques can be used to avoid detection. In the dynamic approaches (Bierma, Gustafson, Erickson, Fritz, & Choe, 2014; Burguera, Zurutuza, & Nadjm-Tehrani, 2011; Grace, Zhou, Zhang, Zou, & Jiang, 2012; Ham, Kim, Kim, & Choi, 2014; Rasthofer, Arzt, & Bodden, 2014; Salman, Elhajj, Chehab, & Kayssi, 2014; Shabtai et al., 2014; Zhang et al., 2013; Zhou, Zhou, Jiang, & Ning, 2012), the application behavior is observed during its execution to detect if there are any inconsistencies with the normal behavior. The problem with this approach is its computational cost and the possibility of evading the detection.

In another category of approaches, called *policy enforcement*, the behavior of a legitimate application is supposed to follow certain predefined rules. If a rule violation is found, the application is considered as malicious. The policy enforcing approaches are divided into *mitigating* and *preventive*.

The mitigating approach cannot prevent the attack but only detect it, as the rule violation can only be observed after the attack has been carried out. On the other hand, the preventive approach can prevent the occurrence of the attack.

2.1. Mitigating policy enforcement approach

In DroidForce (Siegfried Rasthofer, Lovat, & Bodden, 2014), policies are enforced on Android applications in order to deal with the SMS premium rate fraud and data leakage. However, not all policies can prevent the attacks. For example, one policy could state that “no more than two premium-rate messages per day might be sent to each telephone number”. As each SMS costs money, this policy tries only to mitigate the losses incurred by the attacks and does not prevent them.

Nauman et al. (Nauman, Khan, & Zhang, 2010) proposed Apex, a policy enforcement framework, which allows users to impose and define runtime constraints on the usage of resources by the applications. However, there is no discussion about the ability of the defined constraints to prevent the attacks.

2.2. Preventive policy enforcing approach

Almohri et al. (Almohri, Yao, & Kafura, 2014) proposed DroidBarrier, which is a process authentication model for Android.

DroidBarrier provides legitimate applications with security credentials that are used when the process associated with the application wants to be authenticated. The processes that do not have the credentials fail during their authentication attempts and their corresponding applications are considered as malicious. The main issue in this model is that the user needs to initially determine which applications are legitimate, which is not easy to be done by an average user.

In Derhab et al. (Derhab, Saleem, & Youssef, 2014), each SMS application, which is known by the user, is provided with a cryptographic key to encrypt its SMSs. All the SMSs have to pass through a filter at the kernel level before going out of the device. If the filter can correctly decrypt the SMS, it sends the decrypted message to the wireless communication medium. Otherwise, it prevents the SMS from going out of the device and identifies the application, which sent the malicious SMS. However, each SMS application in this solution has to encrypt all its outgoing SMSs, which requires the modification of the application. Also, it incurs additional processing time and SMS sending time due to encryption and decryption operations.

Xu et al. (Xu, Saïdi, & Anderson, 2012) proposed Aurasium, which automatically repackages an application to attach a user-level policy enforcement code. This code monitors any security violation like: sending SMS to premium numbers. If such a violation occurs, Aurasium displays the phone destination number and the SMS content, and waits for the user to confirm or reject the SMS-sending operation. The major disadvantage of Aurasium is the need to define permissions for each application. In addition, the user has to intervene each time there is an attempt to send SMS.

Sun et al. (Sun, Zheng, Lui, & Jiang, 2014) proposed Patronus, an intrusion prevention system for Android. Patronus defines a list of intrusive transactions that might violate Android security. When there is an attempt to execute a transaction, e.g., sendText, Patronus, using API hooking, suspends the execution of the transaction and checks the existence of any permission associated with this transaction. If Patronus does not find any rule in the policy database, it notifies the user and displays the transaction content (e.g., SMS content and destination number). The transaction will resume execution when it is allowed by the user. Otherwise, it will be blocked. Although Patronus utilizes a preventive mechanism (i.e., API hooking), there is no discussion about the ability of the policy to prevent all the malicious SMSs.

Sakamoto et al. (Sakamoto, Okuda, Nakatsuka, & Yamauchi, 2014) proposed DroidTrack, which notifies the user in case of the possibility of information leakage, i.e., when the application calls an API to collect sensitive information, it then calls another API to communicate externally. The user is alerted and asked to confirm or reject the execution of the API. However, this solution requires user intervention whenever the two APIs are called.

MinDroid (Derhab, Saleem, Youssef, & Guerroumi, 2015) assumes that any malicious application starts sending malicious SMS within T time units after its installation on the mobile device. Based on this assumption, MinDroid requires user intervention only during the first T time units starting from the application installation time. During this time period, the user either confirms or rejects the SMS-sending operations. After the expire of this time, all the known SMS applications are allowed to send SMSs. However, if this assumption is dropped, the SMS is only blocked from applications, which are not known for their SMS-sending activities. MinDroid cannot detect known compromised SMS applications, which only send malicious SMSs after the expire of T.

The SMS spam detector (Al-Omany, Al-Muhtadi, & Derhab, 2015) only checks the first SMS sent from each application. It shows an alert dialog box that asks the user to confirm whether he/she is the sender of the SMS. If the user confirms, the application will be classified as authorized and the SMS operation will be resumed. Otherwise, it will be classified as malicious. Further SMSs, which are sent by authorized applications, are analyzed by an SMS spam filter. If an SMS is found legitimate, it is allowed to leave the device and a notification about the sent SMS is shown in the notification status bar for any further verification by the user. It might happen that the filter incorrectly classifies an SMS spam as ham, and the SMS in this case cannot be blocked. Although the prevention of all malicious SMSs is impossible, the notification status bar helps the user identify the compromised application, which sends both legitimate and malicious SMS.

3. System model and basic idea

3.1. Security model

We make the following assumptions about OnDroid:

1. Android kernel code is initially secure (i.e., not compromised and free from malicious code).
2. OnDroid is installed before installing any SMS application.
3. OnDroid utilizes the Sandboxing feature provided by the Android kernel. This feature enables isolating applications from one another. In Android, each application is assigned a unique user ID (UID) and is run as a separate process. The filesystem access rules do not allow one user (resp., application) to access or modify another user's (application's) files.
4. An attacker cannot obtain root privileges of the mobile device using the privilege escalation attack. Under this attack, when a malware gets access to the root account, it can bypass the sandboxing protection mechanism, and gain control over the operating system and data. OnDroid prevents this attack by using some solutions like the one presented in (Lee, Kim, & Park, 2014) or by using SELinux-enabled kernel, which is available starting from Android 4.3. In SELinux, data are protected from applications that can access the root user by applying a custom security policy, which specifies the applications that have access to the data (Allalouf, Ben-Av, & Gerdov, 2014).

3.2. Attack model

An SMS attack is said to be successful if the malicious SMS can go out of the device. We consider two types of SMS attacks:

- *First attack model*: The malicious application hides its SMS-sending activities from the user, and only sends malicious SMSs.
- *Second attack model*: The user installs a compromised SMS application, i.e., SMS-sending activities are not hidden from the user, but the application also secretly sends malicious SMSs. This kind of threat happens when an attacker repackages a legitimate application and inserts malicious code into it.

3.3. User behavior model

We consider that the user behavior with respect to the mobile device is shaped by two main features.

- *User availability*: This feature has two modes: available and unavailable. The user is said to be available with respect to the mobile device if he/she is near that device, i.e, the user is not faraway from the mobile device and can hear the sound alarms triggered by it. If the user is faraway from the mobile device, he/she is considered as unavailable.

- *User activity*: It defines the actions performed by the user when interacting with the mobile device like: reading information, touching the screen.

4. OnDroid description

An Android application can send SMSs by invoking one of the three SMS-sending methods of the *SmsManager* class, which are: *sendTextMessage*, *sendMultipartTextMessage*, and *sendDataMessage*. OnDroid intercepts in real time the calls to SMS-sending APIs. These calls are invoked by applications with the *SMS_SEND* permission, i.e., the AndroidManifest.xml file of the application contains the following line:

```
<uses-permission android:name
= "android.permission.SEND_SMS"/>
```

The architecture of OnDroid is shown in Fig. 1, and it is composed of the following components:

1. *Known SMS applications*: It contains the list of applications, which are known for their SMS-sending activity, and authorized by the user.
2. *API call interception*: It intercepts the calls to SMS-sending and suspends the execution of the invoking application.
3. *Policy database*: It contains the rules the application must satisfy when an SMS-sending method is invoked, as shown in Table 1.

OnDroid performs the following steps each time the SMS application calls one of the three SMS-sending methods mentioned above:

1. Suspend the execution of the application (arrow (1) in Fig. 1).
2. Check if the rule is satisfied.
3. Block running the application if the rule is violated (arrow (3) in Fig. 1). Otherwise, the SMS-sending is delayed for T time units (arrow (2) in Fig. 1).

Table 1 shows the different rules to apply each time an SMS-sending method is invoked. OnDroid has to check if there are mismatches between the user's behavior and the device state. The normal device state that is coherent with any user, which sends SMS via an SMS application, consists of the following:

- *Screen state*: The screen must be on.
- *Device lock state*: The device must be unlocked.
- *Application visibility*: The application must run in the foreground.
- *Application status*: The application must belong to the *Known SMS applications*.

If any of the above rules are not satisfied, it means that either the user is not available in front of the device, or there are some malicious activities running secretly in the background, or unauthorized hidden activities running in the foreground application. In this case, OnDroid blocks the SMS-sending operation, as shown in Fig. 2. Any malicious application, which adopts the first attack model, will be blocked by OnDroid. This is due to the fact that the malicious application is hiding its sending activities and it is not authorized by the user to send SMS. Therefore, it will be detected once it performs the first SMS-sending operation.

In case all the above rules are satisfied (i.e., State 9 in Table 1), it means that the user is interacting or recently interacted with the device (i.e., the screen is on, the device is unlocked, and is running

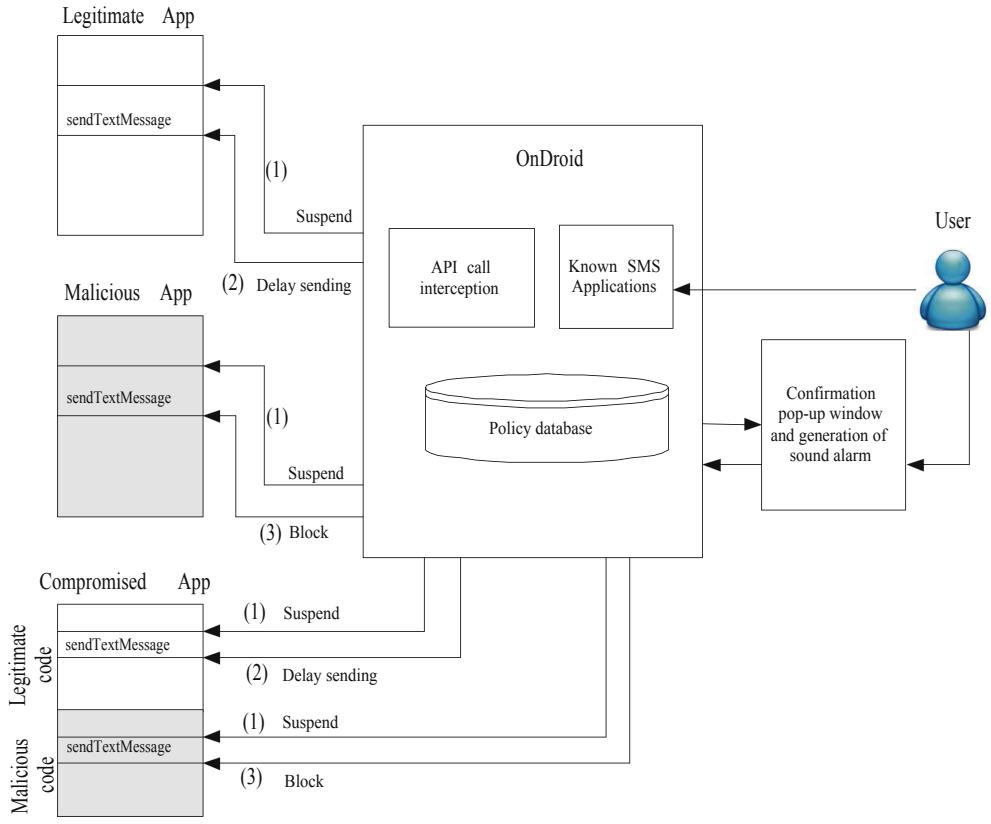

Fig. 1. Architecture of OnDroid

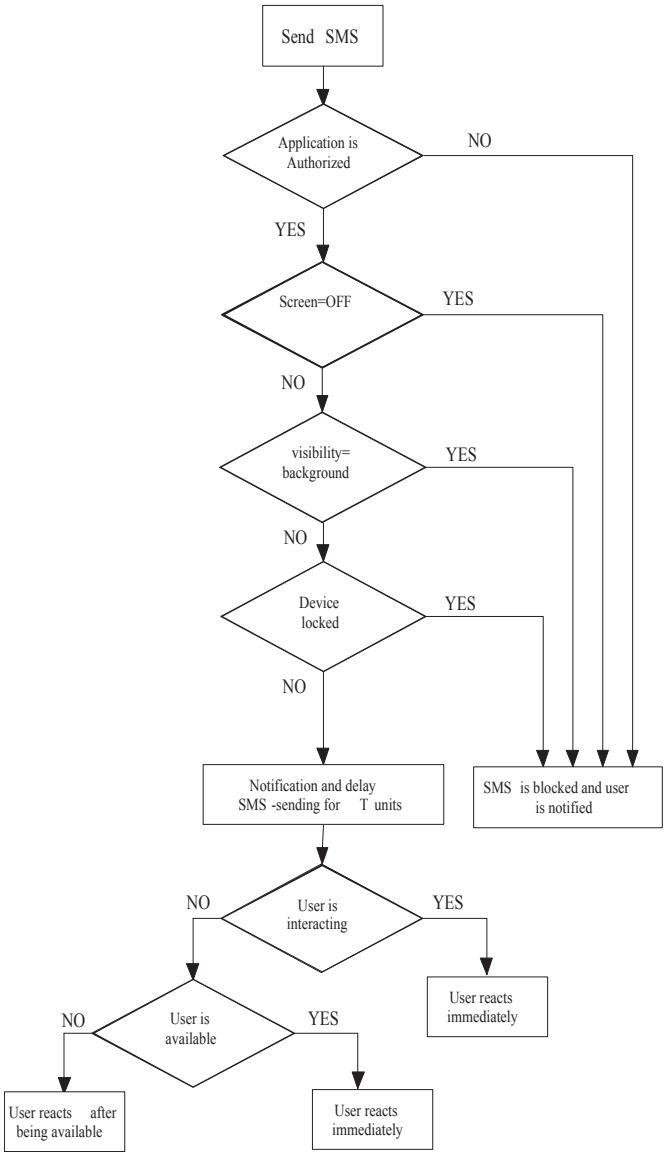

Fig. 2. The flowchart of OnDroid and user interaction cases.

an authorized SMS application in foreground). Here, we distinguish two cases.

- If the application is legitimate, it only sends SMSs originating from the user.
- If the application is compromised, malicious SMSs might be secretly sent while the user is interacting with the application to send his/her legitimate SMSs.

As the defined rules cannot distinguish between legitimate and compromised foreground applications, OnDroid performs the following steps (see Fig. 2):

- It delays the SMS-sending operation for *T* time units. The delay has to be sufficient for the user to intervene if the SMS is malicious.
- It notifies the user using two methods:
- Displaying a notification in the notification status bar, which allows the user who is interacting with the device to read the notification and disable the application that sent the SMS.
- Generating a sound alarm, which allows the user who is available (i.e., near the device) to hear the notification and disable the application that sent the SMS.
- If the user is not available, the attack will stop when the screen becomes off due to user inactivity. When the user interacts again with the device, he/she reads the notification and disables the application that sent the SMS.

The various states assigned to the user and the actions taken by OnDroid in *State 9* are illustrated by a Finite State Machine given in Fig. 3. Transitions are labeled with triggering conditions, which need to hold true in order to switch between the states. Each of the states is described below:

- *ON_Active*: In this state, the user is active, i.e., he/she interacting with the mobile device (i.e., the user is active).
- *ON_Inactive_A*: In this state, the screen is still on and the user is inactive (i.e., the user is not interacting with the mobile device) but he/she is able to hear the sound alarms triggered by this device (i.e., the user is available).
- *ON_Inactive_N*: In this state, the screen is still on but the user is faraway from the mobile device and is not able to hear the sound alarms triggered by this device (i.e., the user is not available).
- *OFF*: In this state, the screen is off.

The screen becomes off if the user does not interact with the device for a time period, called *inactivity time*, which equals the sum of the sojourn time in *ON_Inactive_A* and *ON_Inactive_N* states before transiting to *OFF*. The triggering conditions in the FSM are defined as follows:

- ***User_IN_A***: a predicate that holds true if the user stops interacting with the mobile device but he/she is still available.
- ***User_IN_N***: a predicate that holds true if the user stops interacting with the mobile device and he/she is not available.
- ***User_AC***: a predicate that holds true if the user resumes interacting with mobile device by transiting from *ON_Inactive_A* or *ON_Inactive_N* states to *ON_Inactive_A*.
- ***User_IN_AN***: a predicate that holds true if the user transits from *ON_Inactive_A* state to *ON_Inactive_N* state.

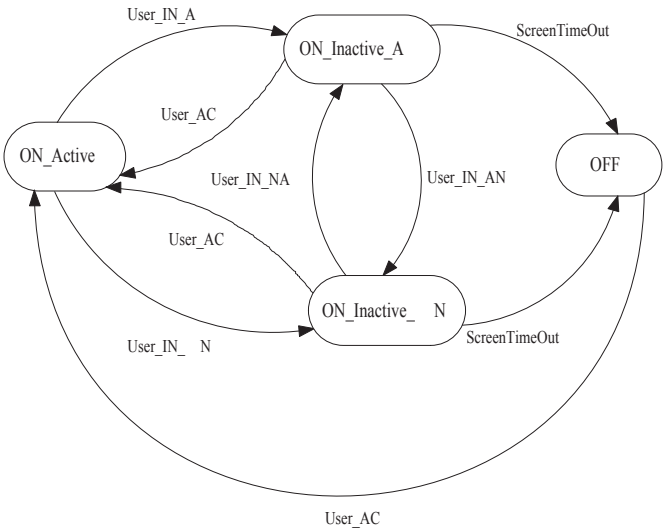

Fig. 3. Finite state machine of OnDroid in state 9.

- ***User_IN_AN***: a predicate that holds true if the user transits from *ON_Inactive_N* state to *ON_Inactive_A* state.
- ***ScreenTimeOut***: a predicate that holds true if the screen becomes off after the expiration of the *inactivity time*.

As described above, OnDroid can prevent the first attack model but it can only detect the second attack model. In the next section, we show how the FSM can be modified to allow as well the prevention of the second attack model.

5. Security analysis

In this section, we analyze the security of OnDroid and discuss its resilience against the following potential attacks:

- *Installation of SMS malware*: A user might be tricked to download and install a malicious SMS application. The installed application starts sending malicious SMS. The latter will be prevented from going out of the mobile device as shown in section 4. As OnDroid is installed before the installation of any SMS application, all the malicious SMS are going be detected whether immediately when the user is available or later when he/she reads the notifications.
- *Compromise of OnDroid code*: An attacker might try corrupting the OnDroid code in order to disable the prevention mechanism offered by OnDroid. The code integrity is preserved by setting the system partition as read-only. In addition, the sandboxing principle ensures that no application can modify the code of OnDroid. Also, protecting the kernel against the privilege escalation attack ensures that the OnDroid code cannot be compromised.
- *Compromise of the policy database*: An attacker might try corrupting the policy database to send malicious SMSs and bypass the SMS-sending rules. As the policy database is only owned and modified by the trusted OnDroid system, there is no way to compromise the policy database. As the Sandboxing principle is used, other applications cannot access the application status and the policy database. In addition, as the Android kernel is protected against privilege escalation attack, a malicious application cannot change the access permissions and corrupt the application status and the policy database.

6. Detection efficiency analysis

As shown in section 4, the SMS-sending operation can only be evaded when the device is in *State 9* and the user is in state *ON_Inactive_N*. The detection efficiency of OnDroid depends on the sojourn time of the user in *ON_Inactive_N* state, and is evaluated in terms of the sojourn probability in all the states except the *ON_Inactive_N* state. We model the user behavior as a continuous-time birth-death Markov chain, and we present two types of chains: worst-case scenario and best-case scenario.

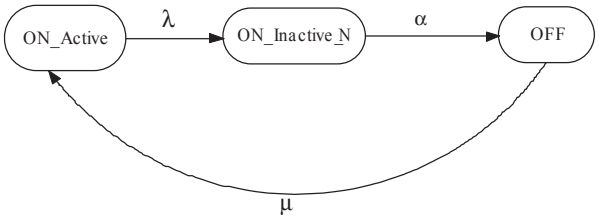

Fig. 4. User behavior model in State 9 (Worst-case scenario).

6.1. Worst-case scenario

The worst-case scenario Markov chain, as shown in Fig. 4 consists of there states: *ON_Active*, *OFF*, and *ON_Inactive_N*. In this scenario, the user is initially in the *ON_Active* state. Then, it transits to the *ON_Inactive_N* state at rate λ. In the *ON_Active_N* state, the user is not available during the inactivity time period, which lasts for $\frac{1}{\alpha}$ time units. After the expire of this time, the user enters *OFF* state, and transits back to state *ON_Active_N* at rate μ.

By assuming an equilibrium in the Markov chain, we apply global balance equations. We denote by P_{ON}, P_N, and P_{OFF} the steady state probability that the user is at states *ON_Active*, *ON_Inactive_N*, and *OFF* respectively. We have:

$$\begin{cases} \lambda P_{ON} = \mu P_{OFF} \\ \alpha P_N = \lambda P_{ON} \\ \mu P_{OFF} = \alpha P_N \\ P_{ON} + P_N + P_{OFF} = 1 \end{cases}$$

By solving the above equations, we get the following:

$$P_N = \frac{\frac{\mu}{\alpha}}{2 + \frac{\lambda}{\alpha}}$$

$$P_{ON} = \frac{\mu}{\lambda\left(2 + \frac{\lambda}{\alpha}\right)}$$

$$P_{OFF} = \frac{1}{2 + \frac{\lambda}{\alpha}}$$

We evaluate the detection efficiency of OnDroid under the following three types of user availability scenarios:

- The expected sojourn times in states P_{ON} and P_{OFF} are equal (Formally, $\lambda = \mu$).
- The expected sojourn time in state P_{ON} is higher than that in state P_{OFF} (Formally, $\lambda < \mu$).
- The expected sojourn time in state P_{ON} is lower than that in state P_{OFF} (Formally, $\lambda > \mu$).

Tables 2–4 show the detection rate of OnDroid in *State 9* as a function of the inverse of the inactivity time period. This rate is calculated as $(1 - P_N)$. The results are presented under low, medium, and high values of α. The lower (resp., higher) α is, the longer (resp., shorter) the user stays in the *ON_Inactive_N* state. We can observe that the probability that a malicious SMS evades detection is high under low values of α and high values of λ and μ. As α decreases (resp., increases), lower (resp., higher) detection probabilities are achieved. Under low values of λ and μ, the user stays in the *ON_Active* and *OFF* states for a longer time, and hence attempts to send malicious SMSs are likely to be blocked by OnDroid or the user.

6.2. Best-case scenario

In the best-case scenario Markov chain, as shown in Fig. 5, the user instead of transiting to state *ON_Inactive_N* from *ON_Active* state, he/she transits to *ON_Inactive_A* state, in which he/she is available during all the inactivity time period. As the user in this scenario does not stay in *ON_Inactive_N* at all, the detection probability will be 100%.

Table 1
SMS-sending rules of OnDroid.

State	Screen state	Application visibility	Device lock state	Authorized	Action
State 1	OFF	Background	locked	YES/NO	Block
State 2	OFF	Background	unlocked	YES/NO	Block
State 3	OFF	Foreground	locked	YES/NO	Block
State 4	OFF	Foreground	unlocked	YES/NO	Block
State 5	ON	Background	locked	YES/NO	Block
State 6	ON	Background	unlocked	YES/NO	Block
State 7	ON	Foreground	locked	YES/NO	Block
State 8	ON	Foreground	unlocked	NO	Block
State 9	ON	Foreground	unlocked	YES	Allow delayed SMS

Table 2
Detection rate ($0.5 \leq \alpha \leq 0.9$).

α	$\lambda = \mu$				$\mu = 1$			$\lambda = 1$		
	1	0.1	0.01	0.001	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.001$	$\mu = 0.1$	$\mu = 0.01$	$\mu = 0.001$
0.5	0.2	0.9714285	0.9996153	0.9999960	0.7142857	0.9615384	0.9960159	0.92	0.992	0.9992
0.6	0.3589743	0.9791666	0.9997311	0.9999972	0.7916666	0.973118	0.9972314	0.9358974	0.9935897	0.9993589
0.7	0.4708994	0.9841269	0.9998015	0.9999979	0.8412698	0.9801587	0.9979649	0.9470899	0.9947089	0.9994708
0.8	0.5535714	0.9875	0.9998475	0.9999984	0.875	0.9847560	0.9984413	0.9553571	0.9955357	0.9995535
0.9	0.6168582	0.989898	0.9998792	0.9999987	0.8989898	0.9879227	0.9987681	0.9616858	0.9961685	0.9996168

Table 3
Detection rate ($1 \leq \alpha \leq 9$).

α	$\lambda = \mu$				$\mu = 1$			$\lambda = 1$		
	1	0.1	0.01	0.001	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.001$	$\mu = 0.1$	$\mu = 0.01$	$\mu = 0.001$
1	0.6666666	0.9916666	0.9999019	0.9999990	0.9166666	0.9901960	0.9990019	0.9666666	0.9966666	0.9996666
2	0.875	0.9977272	0.9999752	0.9999997	0.9772727	0.9975247	0.9997502	0.9875	0.99875	0.999875
3	0.9333333	0.9989583	0.9999889	0.9999998	0.9895833	0.9988962	0.9998889	0.9933333	0.9993333	0.9999333
4	0.9583333	0.9994047	0.9999937	0.9999999	0.9940476	0.9993781	0.9999375	0.9958333	0.9995833	0.9999583
5	0.9714285	0.9996153	0.9999960	0.9999999	0.9961538	0.9996015	0.9999600	0.9971428	0.9997142	0.9999714
6	0.9791666	0.9997311	0.9999972	0.9999999	0.9973118	0.9997231	0.9999722	0.9979166	0.9997916	0.9999791
7	0.9841269	0.9998015	0.9999979	0.9999999	0.9980158	0.9997965	0.9999795	0.9984126	0.9998412	0.9999841
8	0.9875	0.9998475	0.9999984	0.9999999	0.9984756	0.9998441	0.9999843	0.99875	0.999875	0.9999875
9	0.9898989	0.9998792	0.9999987	0.9999999	0.9987922	0.9998768	0.9999876	0.9989898	0.9998989	0.9999898

Table 4
Detection rate ($10 \leq \alpha \leq 100$).

α	$\lambda = \mu$				$\mu = 1$			$\lambda = 1$		
	1	0.1	0.01	0.001	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.001$	$\mu = 0.1$	$\mu = 0.01$	$\mu = 0.001$
10	0.9916666	0.9999019	0.9999990	0.9999990	0.9990196	0.9999002	0.9999900	0.9991666	0.9999166	0.9999916
20	0.9977272	0.9999752	0.9999997	0.9999999	0.9997524	0.9999750	0.9999975	0.9997727	0.9999772	0.9999977
30	0.9989583	0.9999889	0.9999998	0.9999999	0.9998896	0.9999888	0.9999988	0.9998958	0.9999895	0.9999989
40	0.9994047	0.9999937	0.9999999	0.9999999	0.9999378	0.9999937	0.9999993	0.9999404	0.9999940	0.9999994
50	0.9996153	0.9999960	0.9999999	1	0.9999601	0.9999960	0.9999996	0.9999615	0.9999961	0.9999996
60	0.9997311	0.9999972	0.9999999	1	0.9999723	0.9999972	0.9999997	0.9999731	0.9999973	0.9999997
70	0.9998015	0.9999979	0.9999999	1	0.9999796	0.9999979	0.9999997	0.9999801	0.9999980	0.9999998
80	0.9998475	0.9999984	0.9999999	1	0.9999844	0.9999984	0.9999998	0.9999847	0.9999984	0.9999998
90	0.9998792	0.9999987	0.9999999	1	0.9999876	0.9999987	0.9999998	0.9999879	0.9999987	0.9999998
100	0.9999019	0.9999990	0.9999999	1	0.9999900	0.999999	0.9999999	0.9999901	0.9999990	0.9999999

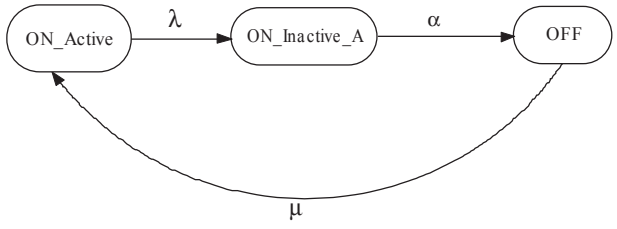

Fig. 5. User behavior model in State 9 (Best-case scenario).

6.3. Full detection of outgoing malicious SMSs

From the results of the worst-case scenario, we can conclude that maximizing the detection probability can be achieved as follows:

- Maximize α, i.e., minimizing the inactivity time period.
- Minimize λ, i.e., maximizing the user activity and interaction with the device.
- Maximize μ, i.e., maximizing the screen off duration.

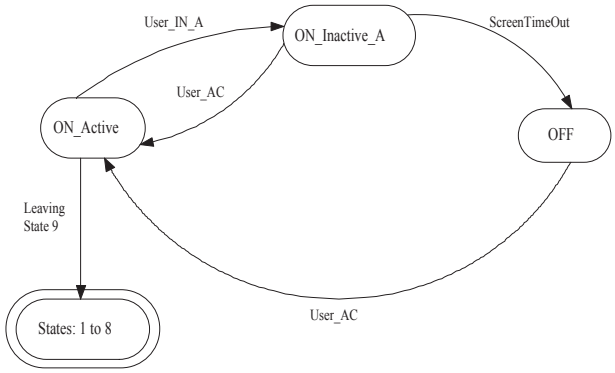

Fig. 6. Finite State Machine of OnDroid in state 9 with adjusted user behavior.

This implies that the user has to change his/her behavior and the interaction time with the device to ensure availability when the attack occurs under State 9. In order to ensure full detection of outgoing malicious SMSs, we propose that the user adjust his/her behavior by adopting the following rule: *Never be in ON_Inactive_N state* as shown in Fig. 6. In other words, when the device is in State 9, the user has to do one of the following actions:

- *Adjusting user activity*: Before leaving the device, the active user has to make sure that the device already left State 9 by closing the foreground SMS application, running it in the background, or locking the device.
- *Adjusting user availability*: The user cannot leave the device until the inactivity time period expires, i.e., the screen becomes off. The user can choose a short inactivity time that does not waste his/her time.

7. Performance evaluation

In this section, we evaluate the performance of OnDroid and compare it with the current policy enforcement security solutions. The performance is evaluated under the following measures:

- *First attack model*: It shows the level of resilience offered by the security solution against the first attack model.
- *Second attack model*: It shows the level of resilience offered by the security solution against the second attack model.
- *User effort*: It shows how much effort (in terms of number of actions) is required for a user to interact with the security solution. For example, reading a notification requires less effort than touching the screen and clicking on one button or many buttons.
- *User interaction easiness*: It shows how easy (in terms of technical complexity) it is for a user with little knowledge in Android security to interact with the security solution. For example, reading a notification that asks to confirm or reject an operation is easier to do than defining security policies for the application.

The results of comparison are shown in Table 5. DroidForce (Siegfried et al., 2014) defines policies that can only mitigate the SMS-sending attack and do not prevent it. The attacker, which knows the policies, can evade detection by performing the attack without exceeding the detection threshold. In DroidForce, the user is required to define policies for each application. However, such a task is not easy for the user to perform. The same thing can also be said about Apex (Nauman et al., 2010), which requires the users to define runtime policies.

As for DroidBarrier (Almohri et al., 2014), IAS (Derhab et al., 2014), and MinDroid (Derhab et al., 2015), the user can initially specify applications known by the user for their SMS-sending activity. The other applications, which secretly send malicious SMSs, are easily detected and blocked. However, they cannot detect or prevent the second attack model as the user might authorize a compromised SMS application. In the case of MinDroid, the second attack model can only be prevented if the malicious SMS is sent before the expire of T time units starting from the application installation time. The effort required by the user to interact with DroidBarrier and IAS is low as it has to define a list of authorized applications. In addition, it is easier for the user to perform this task rather than specifying policies. In MinDroid, the user is required to make additional efforts by intervening for each SMS-sending operation occurring during the first T time units from the application installation time.

As for Aurasium (Xu et al., 2012), Patronus (Sun et al., 2014), and DroidTrack (Sakamoto et al., 2014), they can prevent the first and the second attack models as the user confirms/rejects any attempt to send SMS (e.g., Aurasium and Patronus) or leak information via SMS (e.g., DroidTrack). However, this comes at the expense of usability as the user intervention is required for each SMS-sending attempt. On the other hand, it is easy for the user to intervene as a dialog box is displayed and he/she asked to confirm or deny the SMS content.

The SMS Spam detector (Al-Omany et al., 2015) asks the user to confirm/reject the first SMS from each application. As the first attack model only sends malicious SMS, it is sufficient to notify the user about only the first SMS. The user is also notified about already sent SMS. By checking the notification status bar, the user can detect the

Table 5
Comparison of policy enforcement security solutions.

Rule-based method	First attack model	Second attack model	User effort	User interaction easiness
DroidForce (Siegfried et al., 2014)	No	No	High	No
Apex (Nauman et al., 2010)	No	No	High	No
DroidBarrier (Almohri et al., 2014)	Prevent	No	Low	Yes
IAS (Derhab et al., 2014)	Prevent	No	Low	Yes
MinDroid (Derhab et al., 2015)	Prevent	Prevent: only when the malicious SMS is sent before the expire of T starting from the application installation time	Medium	Yes
Aurasium (Xu et al., 2012)	Prevent	Prevent	High	Yes
Patronus (Sun et al., 2014)	Prevent	Prevent	High	Yes
DroidTrack (Sakamoto et al., 2014)	Prevent	Prevent	High	Yes
SMS Spam (Al-Omany et al., 2015)	Prevent	Detect	Medium	Yes
OnDroid	Prevent	Prevent	Medium	Yes

compromised applications, which send both legitimate and malicious SMSs (i.e., the second attack model). The user interaction effort is medium and it occurs in two cases: (a) when the application sends the first SMS, and (b) when the user checks its notification status bar.

In OnDroid, the user is asked to define the list of applications known for SMS-sending activity. Thus, the other applications, which hide this activity, are blocked (i.e., the first attack model). To deal with the second attack model which occurs only in State 9, the SMS-sending operation is delayed, a notification is added to the notification status bar, and a sound alarm is triggered. If the user has already authorized compromised applications, there will be sufficient time to block this application. To ensure that the user responds to this attack on time, he/she has to adjust his/her behavior by being available during the device's inactivity time or leaving State 9. We consider that the user effort is medium especially if the inactivity time is short. From this comparison, we can conclude that OnDroid can make a good tradeoff between security and usability as it can prevent both attack models and it requires easy interaction and medium effort from the user.

8. Conclusion

In this paper, we have proposed OnDroid, a prevention system against the outgoing malicious SMSs in Android devices. The prevention is based on detecting mismatches between the device state and the user's behavior. If no mismatch is found, the SMS operation is delayed to notify the available user. To deal with the issue of the user being unavailable, we have defined the actions that the user should follow to achieve 100% prevention of malicious SMS-sending operations. Performance evaluation shows that OnDroid offers a good tradeoff between security efficiency and usability. It can prevent the first and the second attack models and requires easy interaction and medium effort from the user. As future work, we plan to extend OnDroid to prevent also the outgoing malicious packets that are originated from mobile malware.

Acknowledgment

The authors extend their appreciation to the Deanship of Scientific Research at King Saud University, Riyadh, Saudi Arabia, for funding this work through the international research group Project no. IRG14-07B.

References

Al-Omany, M., Al-Muhtadi, J., & Derhab, A. (2015). Detection of SMS spam Botnets in mobile devices: Design, analysis, implementation. In *LAP LAMBERT*. Academic Publishing.

Allalouf, M., Ben-Av, R., & Gerdov, A. (2014). Storedroid: sensor-based data protection framework for android. In *International wireless communications and Mobile computing conference (IWCMC 2014)*.

Almohri, H. M., Yao, D. D., & Kafura, D. (2014). Droidbarrier: know what is executing on your android. In *Proceedings of the 4th ACM conference on data and application security and privacy (CODASPY'14)* (pp. 257–264).

Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., et al. (2014). Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *ACM SIGPLAN Notices, 49*(6), 259–269.

Batyuk, L., Herpich, M., Camtepe, S., Raddatz, K., Schmidt, A.-D., & Albayrak, S. (2011). Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within android applications. In *6th international conference on malicious and unwanted software (MALWARE 2011)* (pp. 66–72).

Bierma, M., Gustafson, E., Erickson, J., Fritz, D., & Choe, Y. R. (2014). Andlantis: large-scale android dynamic analysis. In *Security and privacy workshops: Mobile security technologies (MoST)*.

Burguera, I., Zurutuza, U., & Nadjm-Tehrani, S. (2011). Crowdroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on security and privacy in smartphones and mobile devices (SPSM '11)* (pp. 15–26).

Cloudmark. (March 2014). *2013 global messaging threat report*. http://www.cloudmark.com/releases/docs/threat_report/cloudmark-2013-annual-threat-report.pdf.

Derhab, A., Saleem, K., & Youssef, A. (2014). Third line of defense strategy to fight against sms-based malware in android smartphones. In *International wireless communications and mobile computing conference (IWCMC 2014)*.

Derhab, A., Saleem, K., Youssef, A., & Guerroumi, M. (2015). Preventive policy enforcement with minimum user intervention against sms malware in android devices. *Arabian Journal for Science and Engineering*, 1–15.

Enck, W., Octeau, D., McDaniel, P., & Chaudhuri, S. (2011). A study of android application security. In *Proceedings of the 20th USENIX conference on security (SEC'11)*.

Felt, A. P., Chin, E., Hanna, S., Song, D., & Wagner, D. (2011). Android permissions demystified. In *Proceedings of the 18th ACM conference on computer and communications security, ACM* (pp. 627–638).

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

Grace, M. C., Zhou, Y., Wang, Z., & Jiang, X. (2012). Systematic detection of capability leaks in stock android smartphones. In *19th annual network and distributed system security symposium (NDSS)*.

Grace, M., Zhou, Y., Zhang, Q., Zou, S., & Jiang, X. (2012). Riskranker: scalable and accurate zero-day android malware detection. In *Proceedings of the 10th international conference on mobile systems, applications, and services (MobiSys '12)* (pp. 281–294).

Ham, H.-S., Kim, H.-H., Kim, M.-S., & Choi, M.-J. (2014). Linear svm-based android malware detection. In *Frontier and innovation in future computing and communications* (pp. 575–585). Springer.

http://www.idc.com/prodserv/smartphone-os-market-share.jsp

Labs, F.-S. (April 2014). *Mobile threat report: Q1 2014*. http://www.f-secure.com/documents/996508/1030743/Mobile_Threat_Report_Q1_2014.pdf.

Labs, F.-S. (November 2013). *Mobile threat report: July-september 2013*. http://www.f-secure.com/documents/996508/1030743/Mobile_Threat_Report_Q3_2013.pdf.

Lee, H.-T., Kim, D., Park, M., & Cho, S.-J. (2014). Protecting data on android platform against privilege escalation attack. *International Journal of Computer Mathematics*, 1–13. Accepted.

Maslennikov, D. (February 2014). *Mobile malware evolution 2013*. https://www.securelist.com/en/analysis/204792326/Mobile_Malware_Evolution_2013.

Mawston, N. (July 2014). *Android captures record 85% share of global smartphone shipments in q2 2014*. http://www.strategyanalytics.com/default.aspx?mod=reportabstractviewer&a0=9921.

Nauman, M., Khan, S., & Zhang, X. (2010). Apex: extending android permission model and enforcement with user-defined runtime constraints. In *Proceedings of the 5th ACM symposium on information, computer and communications security, ACM* (pp. 328–332).

Rasthofer, S., Arzt, S., & Bodden, E. (2014). A machine-learning approach for classifying and categorizing android sources and sinks. In *Proceedings of the 21st network and distributed system security symposium (NDSS 2014)*.

Sakamoto, S., Okuda, K., Nakatsuka, R., & Yamauchi, T. (2014). Droidtrack: tracking and visualizing information diffusion for preventing information leakage on android. *Journal of Internet Services and Information Security (JISIS), 4*(2), 55–69.

Salman, A., Elhajj, I., Chehab, A., & Kayssi, A. (2014). Daids: an architecture for modular mobile ids. In *28th international conference on advanced information networking and applications workshops (WAINA)* (pp. 328–333).

Schmidt, A.-D., Bye, R., Schmidt, H.-G., Clausen, J., Kiraz, O., Yuksel, K., et al. (2009). Static analysis of executables for collaborative malware detection on android. In *IEEE international conference on communications (ICC '09)* (pp. 1–5).

Seo, S.-H., Gupta, A., Sallam, A. M., Bertino, E., & Yim, K. (2014). Detecting mobile malware threats to homeland security through static analysis. *Journal of Network and Computer Applications, 38*(0), 43–53.

Shabtai, A., Tenenboim-Chekina, L., Mimran, D., Rokach, L., Shapira, B., & Elovici, Y. (2014). Mobile malware detection through analysis of deviations in application network behavior. *Computers & Security, 43*(0), 1–18.

Siegfried, S. A., Rasthofer, Lovat, E., & Bodden, E. (2014). Droidforce: enforcing complex, data-centric, system-wide policies in android. In *Proceedings of the the 9th international conference on availability, reliability and security (ARES 2014)*.

Suarez-Tangil, G., Tapiador, J. E., Peris-Lopez, P., & Blasco, J. (2014). Dendroid: a text mining approach to analyzing and classifying code structures in android malware families. *Expert Systems with Applications, 41*(4, Part 1), 1104–1117.

Sun, M., Zheng, M., Lui, J. C., & Jiang, X. (2014). Design and implementation of an android host-based intrusion prevention system. In *Proceedings of the annual computer security applications conference (ACSAC)*.

Traynor, P., Lin, M., Ongtang, M., Rao, V., Jaeger, T., McDaniel, P., et al. (2009). On cellular botnets: measuring the impact of malicious devices on a cellular network core. In *Proceedings of the 16th ACM conference on computer and communications security (CCS '09)* (pp. 223–234).

Xu, R., Saïdi, H., & Anderson, R. (2012). Aurasium: practical policy enforcement for android applications. In *USENIX Security Symposium* (pp. 539–552).

Zhang, Y., Yang, M., Xu, B., Yang, Z., Gu, G., Ning, P., et al. (2013). Vetting undesirable behaviors in android apps with permission use analysis. In *Proceedings of the 2013 ACM SIGSAC conference on computer & communications security, ACM* (pp. 611–622).

Zhou, W., Zhou, Y., Jiang, X., & Ning, P. (2012). Detecting repackaged smartphone applications in third-party android marketplaces. In *Proceedings of the second ACM conference on data and application security and privacy (CODASPY'12)* (pp. 317–326).