Droid Mood Swing (DMS): Automatic security modes based on contexts

Md Shahrear Iqbal, Mohammad Zulkernine

School of Computing
Queen's University, Kingston, Ontario, Canada
{iqbal, mzulker}@cs.queensu.ca

Abstract. Smartphones are becoming ubiquitous and we use them for different types of tasks. One problem of using the same device for multiple tasks is that each task requires a different security model. To address this problem, we introduce Droid Mood Swing (DMS), an operating system component that applies different security policies to detected security modes automatically. DMS uses a context manager that tracks the context of the phone from the available sensors. DMS then determines the security mode from the contexts and can impose a number of security measures, namely fine-grained permissions, an intent firewall, a context-aware SD card filesystem, and a permission verification system. The permission verification system uses machine learning techniques to detect suspicious apps and anomalous permission requests. DMS also provides an API that enables third-party developers to make their apps behave differently in different modes. DMS is designed especially for end users and does not compromise the usability of the phone. Device vendors will be able to control configurations (a switching logic and security policies) of the modes through DMS. We implement DMS using the Android Open Source Project (AOSP) and evaluate it in terms of portability, functionality, security, and operational overheads. The evaluation results show that DMS offers a more secure smartphone operating system without incurring any noticeable overhead.

Keywords: context-dependent security, smartphone security and privacy, Android security, mobile malware

1 Introduction

Smartphones are already an integral part of our daily life. People use their smartphones for tasks that require different levels of security, e.g., writing emails, surfing the web, listening to music, watching videos, playing games, performing financial transactions, and creating reports. Companies now allow BYOD (Bring Your Own Device) policy, which lets the employees bring and use their own smartphones for accessing confidential business resources. As a result, companies want the security of their resources and the ability to manage their devices remotely. To address the problem, researchers proposed isolated environments [24, 39] (e.g., using virtualization techniques) on the device with different

controllable security properties. However, we observe a lack of research proposing similar solutions for end users. Users also have their own set of apps for personal use and they do not want to compromise their privacy due to malware or apps provided by their workplaces. Moreover, with the advent of the internet of things (IoT), people are expected to use their smartphones to control other devices (smart TVs, smart fridges, etc.) and new smartphone payment services (e.g., Google Wallet and Apple Pay) require extra security measures.

Despite all the threats, the security model of smartphone operating systems distributed to the normal users is changing at a slow pace. Since a single device is being used by mostly security unaware users, we strongly believe that automatic detection of the device context and switching to different security modes to protect user resources are now a necessity.

In this paper, we propose Droid Mood Swing (DMS), a system that applies fine-grained access control to the detected security mode. To detect the security mode, DMS uses Flamingo [23] that maintains a cache of security contexts and parameters to be used by operating system components and third-party applications. For example, if the user is using the camera app inside his or her home, the phone will switch to a mode where captured resources will be saved securely. It is designed in a way so that device vendors can manage the modes and their configurations. We define a number of security modes that cover almost all the necessary tasks of a regular user. A language is also developed to automate the process of configuring different modes. Each mode will have a configuration file which describes the access control policies of the mode.

DMS can apply fine-grained access control which consists of the ZoneDroid [22] tool and a number of security measures (called "restrictions" in this paper). ZoneDroid realizes the concept of application zones to sandbox a group of applications. Each zone has policies to control the behavior of the apps. However, in ZoneDroid, users have to customize zones and policies by themselves and there was no concept of security contexts. In this paper, we automate the process of configuring each mode to reduce user involvement, thus improving usability.

In particular, this paper makes the following contributions:

- We propose DMS that can switch to multiple security modes based on the detected context and applies fine-grained access control to satisfy the security requirements of each mode.
- We develop several restrictions to facilitate access control, namely an intent firewall (IPC restriction), a context-aware SD card filesystem (file access restriction), and a permission verification system (permission restriction).
- We develop a configuration language to automate the process of configuring ZoneDroid and restrictions without requiring any input from users.
- We provide an API for app developers so that apps can be programmed to behave correctly in different security modes and honor the security policies of each mode.

To implement DMS, we use the open source Android operating system. However, the concept of DMS is not restricted to any particular smartphone operating system. All operations of DMS are completely transparent to users. We

also evaluate DMS in terms of performance and storage overhead and show that they are negligible.

In today's highly connected environment, a system like DMS can considerably improve the security of a regular user. Device vendors (Samsung, Google, OnePlus, Asus, Sony, etc.) can use DMS to manage security modes and ensure the safety of device resources.

The remainder of the paper is organized as follows. Section 2 provides the necessary technical background on Android permission model. We illustrate the design and operation of DMS in Section 3. We evaluate DMS in Section 4 and describe the related work in Section 5. Finally, we conclude in Section 6 with a little discussion on the limitations and future work.

2 Background

Security in the smartphone ecosystem begins from the application market so that malware cannot enter the device through markets. Most markets review submitted applications and provide a mechanism to sign them. Smartphone operating systems also provide a layered approach towards protection. Normally, they consist of a lower level kernel and a middleware. For example, Apple iOS uses the darwin kernel and a middleware written in C.

2.1 Android security

Android is a Linux-based open source operating system and consists of the Linux kernel (with over 250 patches for Android [1]), a Java middleware (called the Android framework), and stock applications (phone, contacts, etc.). Android security is mainly built upon a permission-based mechanism which restricts accesses to device resources. In this subsection, we provide a description of the permission model of Android Marshmallow which introduced a new enforcing technique.

Permissions in Android. Android uses permissions to protect system components, APIs, and resources. A permission is simply a unique text string. There are more than hundred permissions [2] defined in the Android operating system. In addition to the Android defined permissions, application developers can declare customized permissions to protect their resources.

A permission can be associated with one of the following four protection levels [3]: normal, dangerous, signature, signature-Or-System. According to `developer.android.com`, normal permissions are low-risk permissions and dangerous permissions are for sensitive resources. Normal permissions are given at install time and cannot be revoked by the user. However, for dangerous permissions, users are notified at runtime. An application can continue only when the user allows the requested permission. More importantly, now users can revoke dangerous permissions later.

In Android, each application is assigned a unique user id (UID). Based on the UID, the kernel provides the application sandboxing. In addition, Android

permission mechanism enforces access control in two levels. In one level, the `system_server` process (in Android framework) ensures that the calling component has the necessary permission. In another level, a number of permissions are enforced by the underlined Linux's discretionary access control (DAC). We call these permissions "granted permissions".

When an application process is created by the activity manager, it maps the granted permissions to the corresponding groups. The group IDs are then passed to the `zygote` process which forks itself and sets appropriate group IDs. `Zygote` is a daemon which is started by the `system init` and responsible for the creation of new processes. These permissions are given to the virtual machine process and dynamic permission checks will not occur for some of these permissions. As an example, the INTERNET permission in Android is mapped to the Linux `inet` user group and consequently, internet access is controlled by the underlying Linux kernel.

3 DMS Architecture and Operation

This section describes the architecture and operation details of DMS. One of the goals of DMS is to make the smartphone operating system security-aware. DMS does this without creating multiple personas or compartments. In many of the related research, creating separate compartments for abstracting data and apps is common. However, in our opinion, those approaches require far more user involvement. DMS switches to different security modes based on the detected context. Once a mode is activated, DMS can restrict certain app behaviors to protect resources. Here, data and apps are not isolated. Rather, we modify the filesystem and other OS components to deny access intelligently. The overhead of doing so is much less in comparison to other compartmentalization techniques. DMS does not require any input from users. A description of different components of DMS along with its architecture is given in the following subsections.

3.1 DMS architecture

The architecture of DMS is presented in Figure 1. DMS Manager is the controller of DMS and connects with vendors to get configuration files. To switch modes, it uses context information from the Flamingo context manager [23]. Flamingo defines a smartphone's security context as the degree of threat to the device's resources. It uses different phone sensors to determine the context and a number of security parameters. Flamingo exports these parameters and manages a cache to reduce power consumption (by avoiding redundant recalculation).

Based on the configuration, the DMS Manager uses the ZoneDroid Manager to modify zones and policies. All changes are written to an SQLite database named DMS.db. To implement features of DMS, we modify a number of operating system components. The components call the DMS Manager before performing their intended tasks. DMS also has a native service which communicates with other native components (e.g., the SD card filesystem) and performs actions that require root privileges (e.g., issues iptables command).

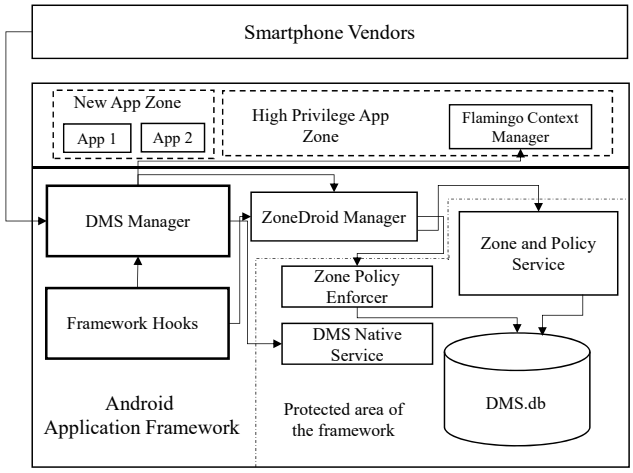

Fig. 1: The architecture of DMS.

3.2 DMS security modes

Each security mode is a unique combination of zones, policies, rules, and restrictions. Restrictions are components (that can be activated or deactivated) to restrict certain phone features. Below, we define these terms.

Definition 1 (Security Mode). *A mode $m = (L_m, Z, T, s)$ is defined by a label L_m, a set of zones Z, a set of Restrictions T, and a security level s. A configuration function $f_{mc}(m, s) : M \rightarrow C$ maps the mode and the security level to a unique configuration, where M is the set of modes and C is the set of configurations.*

Definition 2 (Zone). *A zone $z = (L_z, P, A)$ is defined by a label L_z, a set of policies P, and a set of Applications A. An application in the device can be assigned to only one zone at any given time. The zone policy enforcer function f_{zpe} defines the complete set of conditions under which an application in zone z is allowed to call another component or system API.*

Definition 3 (Policy). *A policy p is a set of rules R that defines the conditions under which an application is granted a number of permissions. A policy checker function $f_{pc}(o)$ is defined as $r_1 \wedge r_2 \wedge r_3 \wedge \cdots \wedge r_n \rightarrow \{permit, deny\}$, where o is a permission and n is the number of rules in the policy. In the case of a conflict, the rule with the deny will prevail.*

Definition 4 (Rule). *A rule r takes the form (o, V, e), where for a permission o, we can denote a set of attributes and values and an action e. Here, V is a set of 2-tuples of the form $< attribute, value >$.*

For example, the rule (SEND_SMS, {<TIME, 8AM_TO_5PM>,<PHONENO, N>}, DENY) restricts apps in a zone to send SMSs to the number N from 8AM to 5PM.

Default security modes. Default security modes of DMS are based on security contexts detected by Flamingo. Flamingo detects the following contexts: home, office, and outdoor. Moreover, in the home context, the phone can be in two subcontexts (context within another context), namely casual and private. In the office context, the phone detects whether the user is in a meeting. In addition, Flamingo identifies another two subcontexts (side-loaded, financial) and a number of security parameters. The parameters are *Location, Place-type, Type-of-user-activity, Is-moving, In-use, Is-locked, Type-of-active-app, Is-side-loaded, Network-type, Is-network-encrypted, Is-camera-on, Is-mic-on, Is-storage-encrypted*, and *Number-of-trusted-devices*. Based on the context, subcontexts, and parameters, DMS provides the following security modes:
Home-casual. In this mode, the smartphone is located in user's home. DMS blocks all dangerous permissions for office apps (camera, mic, etc.).
Home-private. DMS switches to this mode whenever a user turns on the camera or the mic inside his or her home. Files saved in this mode will be denied access from any other mode.
Home-financial. DMS activates this mode when the user opens a financial app. DMS restricts unencrypted network and inter-process communication in this mode.
Office-casual. DMS activates this mode when the location of the phone is office. All dangerous permissions are blocked for personal apps.
Office-private. DMS switches to this mode if the *In-meeting* security parameter is true. All background sensor accesses are blocked in this mode.
Office-financial. Similar to the Home-financial mode.
Outdoor-casual. If the smartphone is not in office or home, DMS activates the outdoor-casual mode.
Outdoor-financial. Similar to the Home-financial mode. In addition, if the *Place-type* is a place with a point of sale (POS) terminal (grocery stores, malls, restaurants, etc.) and an NFC payment app is active, DMS blocks all network accesses (internet, NFC, Bluetooth) to other apps. Inter-process communication is also restricted.

3.3 Fine-grained permissions

DMS uses ZoneDroid [22] that provides an efficient solution to control a group of applications easily. Each zone provides a certain level of privileges. By default, DMS creates the following zones: New, Trusted, Untrusted, High privilege, Office, and Restricted (for malware). The separation of application zones is analogous to the separation of industrial and residential areas in a smart city [21] where each area has their own security policies and a person has to adhere to the policies based on his or her location. It is worth mentioning that all system apps go to the Trusted app zone and all newly installed apps go to the New app zone. Users have to move them to either the Trusted zone or the Untrusted zone. Users should also keep in mind that default policies are liberal for apps in the Trusted and High Privilege zone (e.g., antiviruses).

DMS can deny permissions based on the following three attributes: time, phone number, and folder location. Attributes and their values become members of the set V. The set V and a decision to allow or deny make a permission fine-grained. Each fine-grained permission forms a rule r and a number of rules (R) constitute a policy p as described in the previous Subsection. We list a sample policy with four rules in Listing 3.1. The policy denies access to location, contacts, and all folders other than FOLDER1 from 10PM to 8AM. It also restricts sending SMSs to phone number N.

DMS zone operations. Using ZoneDroid, DMS can create/edit/delete zones, policies, and rules. For example, DMS can create a new zone, rename a zone, or move apps from one zone to another. It can create a new policy with multiple rules, or edit/add/delete rules in an existing policy. DMS can also disable a zone. Disabling a zone will block all the apps that belong to the zone from executing.

```
{ACCESS_FINE_LOCATION,{<TIME,10PM_8AM>},DENY}
{READ_CONTACTS,{<TIME,10PM_8AM>},DENY}
{SEND_SMS,{<TIME,ALWAYS>,<PHONENO,N>},DENY}
{WRITE_EXTERNAL_STORAGE,{<TIME,10PM_8AM>,<FLOC,FOLDER1>},ALLOW}
```

Listing 3.1: A policy with four rules.

3.4 Context-aware filesystem

We modify the Android SD card filesystem (written in C) to make it context-aware. Android uses FUSE [5] to emulate FAT on SD card. The modified filesystem connects with the DMS native service using an abstract Unix domain socket to get the value of the current mode. It then writes the information in the extended attribute of the underlying ext4 filesystem. If file access restriction is enabled, the SD card will deny access to files that are not created in the current mode.

3.5 Inter-process communication (IPC) firewall

A technique is developed to allow blocking of all inter-process communications to and from a zone and to and from any particular app. In Android, all intents pass through an intent firewall to allow custom rules for IPC to be applied. We modify the file and now the intent firewall consults with DMS before allowing any intent if IPC restriction is enabled. This restriction can be useful in a scenario where a benign app has vulnerabilities that other malicious apps can exploit via IPC.

3.6 Restrict network

DMS can block communications to and from the internet per application, per zone or per mode. For example, if the current network is detected as insecure by Flamingo, a mode can block part of the system from communicating with

the internet. DMS native service implements the network blocking mechanism by issuing iptables rules.

3.7 Permission verification

The permission verification system allows DMS to block anomalous permissions. The steps of the permission verification are as follows:

1. DMS collects information from Google Play and the VirusTotal [34] website and applies machine learning classification to detect suspicious (probably malicious) apps and anomalous permission requests. VirusTotal is a website that analyzes applications by more than 60 well-known antiviruses and gives a score that tells how many antiviruses have recognized the app as malicious.
2. DMS separates the collected information based on app categories. For example, in Google Play, there are more than 50 categories. Some examples are Education, Personalization, Lifestyle, Entertainment, Music & Audio, and Travel & Local.
3. DMS trains a classifier for each category that predicts the suspiciousness of new apps. Here, DMS considers an app suspicious if its VirusTotal score is more than 0.
4. DMS also determines the permissions that are not in the set of top 30 most used permissions of the non-suspicious apps. These are the anomalous permission requests.

Features. DMS uses permissions and review scores as features for the classifiers. Permission usage is a good way to cluster well-behaved applications and used in the existing literature [20]. The feature review score is calculated from the actual review score from the app market (which is an average) and the number of reviews as a low review count may bias the classifier. To normalize this impact, we use the following formula [12] to calculate the score:

$$review\ score = Ps + 5(1 - P)(1 - e^{\frac{-q}{Q}})$$

Here, s is the review score and q is the number of reviews. After some experiments, we use $P = 0.7$ and $Q = 5,000$ as these values give a satisfactory feature importance for the review score.

Classifiers. DMS can use most of the common classifiers for supervised learning. In this work, we investigate the following classifiers: Naive Bayes, Support Vector Machine (SVM) with Radial Basis Function (RBF) kernel, Decision Tree, K-Nearest Neighbors, and Random Forest [13]. To compare the effectiveness of the classifiers, we report the precision, recall, and F_1 score.

3.8 DMS configuration language

The configuration language can describe a set of actions to be performed when switching modes. It supports creating/deleting/disabling zones, moving applications between zones, and applying a set of policies to any zone. It can describe which restrictions should be activated on the current mode and the conditions of switching the mode. We demonstrate some actions in Listing 3.2.

```
CHECK SWITCHING-CONDITION:
        IF SECURITY-PARAM IS TRUE/FALSE
        MESSAGE USER "SWITCHING TO MODE MODE-NAME NOT POSSIBLE, SECURITY-
            PARAM IS TRUE/FALSE"

SCOPE MODE-NAME:
        RESTORE ORIGINAL
        UPDATE
        CREATE ZONE: ZONE-NAME
        DELETE ZONE: ZONE-NAME
        DISABLE ZONE: ZONE-NAME
        MOVE APPS: FROM ZONE-NAME1 TO ZONE-NAME2: ALL
        MOVE APPS: FROM ZONE-NAME1 TO ZONE-NAME2: ALL EXCEPT CURRENT
        APPLY POLICY:
                ZONE ZONE-NAME1:
                        {READ_CONTACTS,{<TIME_ALWAYS>,DENY}
                        {GET_ACCOUNTS,{<TIME_ALWAYS>,DENY}
                ZONE ZONE-NAME2:
                        POLICY: POLICY-NAME1
        RESTRICT IPC ZONE-NAME3
        RESTRICT IPC ZONE-NAME4 APP-NAME1, APP-NAME2
        RESTRICT FILE-ACCESS
        RESTRICT NETWORK IF SECURITY-PARAM IS TRUE/FALSE
        RESTRICT PERMISSION
```

Listing 3.2: Examples of actions in the DMS configuration language.

3.9 DMS developer API

DMS provides an API for the developers. Using the API, app developers can determine the policies and restrictions of the current mode and make their apps behave accordingly.

4 Evaluation

In this section, we describe the evaluation results of DMS. First, we evaluate the classifiers for the permission verification system. Then, we evaluate DMS in terms of portability, functionality, security, and operational overheads. We implement DMS by modifying the Android Open Source Project (Marshmallow version 6.0.1 r17 MMB29V). We deploy the resulted operating system to a Google Nexus 5. It has Qualcomm MSM8974 Snapdragon 800 CPU (Quad-core 2.3 GHz), Adreno 330 GPU with 2GB memory, and the following sensors: accelerometer, gyroscope, magnetometer, light, proximity, pressure, and GPS.

4.1 Evaluation of the classifiers for the permission verification system

To detect anomalous permission requests and suspicious apps, we need an appropriate permission request classifier. The classifier should identify most of the suspicious apps (maximize the recall) and also needs to be reasonably accurate (low false positives). To select the best classifier, we calculate the effectiveness of the classifiers on the ground truth dataset.

Ground truth. We select 14,674 apps from the androzoo [7] Android database. All the apps belong to the PERSONALIZATION category. PERSONALIZATION is one of the top 10 categories in Google Play and androzoo has the highest number of suspicious apps in this category. Among 14,674 apps, 10,316 apps are benign (VirusTotal score is 0) and 4,358 apps are suspicious (VirusTotal score is more than 5).

To build the dataset, we write a `node.js` script to visit the selected apps in Google Play. We collect the details and permission list of all the apps. From the app details, we only consider the application review score and the review count. We then divide the dataset into two parts: Training and Testing. Some details on the ground truth dataset are listed in Table 1.

Table 1: Number of mobile apps selected for the ground truth dataset.

Number of Apps	Training	Testing
Benign	7,000	3,316
Suspicious	3,000	1,358
Total	10,000	4,674

Select the appropriate classifier. From the ground truth dataset, we generate the features and then use the selected classifiers (described in Subsection 3.7) to classify an app as either benign or suspicious. Table 2 shows the comparison of the precision, recall and F_1 score of the various classifiers.

Table 2: Performance of different machine learning classifiers. For each classification algorithm, we report the precision, recall, F_1 score, and support.

Algorithm	Class	Precision	Recall	F_1 Score	Support
NaiveBayes	0	0.95	0.04	0.08	3,316
	1	0.30	0.99	0.46	1,358
	avg/total	0.76	0.32	0.19	4,674
SVM	0	0.86	0.96	0.91	3,316
	1	0.86	0.63	0.73	1,358
	avg/total	0.86	0.86	0.86	4,674
DecisionTree	0	0.93	0.93	0.93	3,316
	1	0.84	0.83	0.83	1,358
	avg/total	0.90	0.90	0.90	4,674
15kNN	0	0.93	0.93	0.93	3,316
	1	0.83	0.82	0.83	1,358
	avg/total	0.90	0.90	0.90	4,674
RandomForest	0	0.94	0.94	0.94	3,316
	1	0.85	0.85	0.85	1,358
	avg/total	0.91	0.91	0.91	4,674

In the ground truth data, RandomForest has the highest average precision and recall of 0.91 and 0.91. It also has the highest F_1 score of 0.91. NaiveBayes and SVM have poor recall values compared to other classifiers. In conclusion, we decide to use the Random Forest classifier.

Discussion on machine learning. DMS permission verification system is used to perform a second-level verification from the user if the sought permission is

from an app which is either classified as suspicious or the permission is in the anomalous permission list. If turned on, DMS blocks unusual permissions and notifies the user. If the user wants, he or she can allow the permission and let the app perform its task. The type of the problem (i.e., providing suggestions to users) encouraged us to use machine learning techniques and the results of machine learning algorithms are often much more accurate than human-crafted rules. It gives us a quick overview on the nature of the apps and anomalous permissions. Here, the chosen Random Forest classifier correctly classifies 85% of the suspicious apps.

4.2 Portability

Other than the Google Nexus lines of devices, all manufacturers ship their own versions of Android. They provide a custom experience of Android which requires modifications to the AOSP project. The modifications required for implementing DMS components can be applied to the AOSP project easily. Notably, DMS uses the modified permission mechanism of Android which was introduced in version 6.0. As a result, DMS can only be implemented in Android version 6.0 and above. However, this does not impact the execution of apps that are developed for older versions of Android.

4.3 Functionality

Security mode switching. To test the effectiveness of DMS, we move the phone to home, office, and outdoor. Also, a banking app is used as a financial app. DMS successfully detects the eight modes described in Subsection 3.2 and applies mode configurations. No applications are crashed (including the open one) during a mode switch. This is because changes in the zone configuration are applied directly to the DMS.db database and restrictions are enforced in the framework layer of Android. However, an already allowed permission may be rejected in the new mode. Applications that are built for Android version 6.0 and above handle the case gracefully and often ask for the permission again. Users can decide to click on the "Do not ask again" checkbox to prevent further permission requests.

Fine-grained permissions and restrictions. We develop a simple application performing the following sensitive operations: initiate network connections, access user's photos, access the contact list, and access the camera. We install the app in the Untrusted zone. We observe that when the app opens the phone's camera inside the home, the phone switches to the *Home-private* mode. All the images taken in this mode are saved securely via the context-aware filesystem. The app cannot send them over the internet as networks are restricted in this mode. As soon as the app closes the camera, the phone switches to the *Home-casual* mode and the app has no longer access to the images taken. Also, in the *Home-private* mode, IPC is restricted for apps inside the Untrusted zone. As a result, the app cannot share the captured images via IPC to other apps that can leak the images. When the app tries to access the phone's contact list, DMS

blocks the request and notifies the user. We then move the phone to an outside cafe (where the network is open and unencrypted) and try to open the banking app. DMS gracefully blocks the internet for the app and notifies the user.

4.4 Security analysis

DMS adds an additional layer of security on top of Android's middleware. Modification to the SD card filesystem ensures the security of the external storage. In this subsection, we discuss some of the attacks on smartphone middleware and how DMS improves the scenario.

Assumptions. We completely trust the Android kernel and middleware. We consider a strong adversary whose goal is to access the sensitive user data as well as to use the device as a victimized attacker.

Over-privileged third-party apps, libraries, and sensory malware. Many third-party apps ask unnecessary permissions to access device information which threatens user privacy [6]. Developers also use third-party software development kits (analytics, social networking, etc.) and ad-libraries without knowing the details of their code. Unfortunately, Android always grants a full set of permissions to third-party libraries. Unintended accesses to users' private data by the complex and often obfuscated libraries make it hard for developers to estimate their correct behavior [31].

Sensory malware try to use the data collected from the phone's sensor to infer different important information (user password, location history, etc.) [16,25,38].

DMS always maintains two zones of newly installed apps and untrusted apps. Apps in these zones must adhere to the policies of the zones in different security modes. As a result, asking more permissions will not yield any benefit until the user moves the applications to the trusted zone. Sensory malware are also deemed ineffective as DMS rejects their requests to access sensors from the Untrusted zone.

Confused deputy and collusion attacks. In confused deputy attacks, malware leverage unprotected interfaces of benign apps. For example, a malicious app can use the vulnerable service of a fancy SMS app by a novice developer and send SMSs through it without having the SMS permission [15,40]. In a collusion attack, two malicious apps are involved. Individually, their permission sets are not malicious. However, they collude using covert or overt channels to gain a permission set which can be used to perform unintended tasks [26,29]. In both the cases, DMS can be effective if such applications are sent to a zone where IPC is restricted between apps and zones. However, users' knowledge about the apps is necessary in this case.

Being a victimized attacker. Internet users are often victimized by malicious attackers. Some attackers infect and use innocent users' devices (by making them a part of a botnet) to launch large-scale attacks without the users' knowledge. Similar to the desktop computers, smart devices (Android phones, TVs, etc.) can also be a part of such botnets and help launch large-scale low-noise attacks (e.g., DDoS, click-fraud, spam). They often perform their malicious task when the device is not busy (in the night). In the existing version of Android, users

can not block internet access (it is a permission with protection level normal). However, DMS can block internet access to a zone (containing untrusted apps) when the device is not being used (e.g., from 11PM to 7AM).

4.5 Operational overheads

In this subsection, we evaluate DMS in terms of performance and storage usage. We also evaluate the overhead of the SD card filesystem. In each case, we show that there is a very little to negligible overhead.

Table 3: Individual test scores from the AnTuTu benchmarking app.

Test Group	Score	
	Stock	DMS
3D	8,640.8	8,726.2
UX	17,949.8	18,007.6
CPU	16,143.4	16,922.8
RAM	5,249.8	6,823.2
Total	47,983.8	50,479.8

Performance. We quantify performance using the popular AnTuTu benchmarking app [4] available from Android markets. The app tests CPU and memory performance, 2D/3D graphics, Disk I/O, Multitasking, etc. It gives a score for each test which can be used to compare relative performance between devices.

The benchmarking app runs concurrently with the standard set of Android apps that launched at boot. Based on the official Android source code (6.0.1), these apps are launcher, contacts (and its provider process), photo gallery, dialer, MMS, and settings.

All numbers from the benchmarking app are averaged over 5 runs. Table 3 shows the comparison of scores resulted from the app. The score of the stock version is slightly lower due to the higher number of Google services running in it compared to the DMS version. However, the score differences are not really significant and it is clear that performance is not hampered by activating DMS.

The main runtime overhead results from the zone policy enforcement mechanism. Every time Android checks for a permission, our hook in the `checkPermission` function of the package manager will execute the zone policy enforcer function f_{zpe}. Here, in Figure 2, we measure the actual running time of the policy enforcer function. For this experiment, we use a policy denying all the dangerous permissions as a worst-case scenario. In a total of 456 calls, the average running time was 7.91ms and the standard deviation was 5.71ms. As we understand, the occasional spikes in the running time are the result of the high CPU usage during that time. However, this delay will vary mode to mode as each mode may have different policies. Overall, an 8ms delay (in the worst-case) in the `checkPermission` function will not be noticeable by users.

Storage usage. We modify a few system files in the Android framework. However, this does not result in a change (in terms of size) in the final operating system size. Also, the DMS.db database contains only textual information and

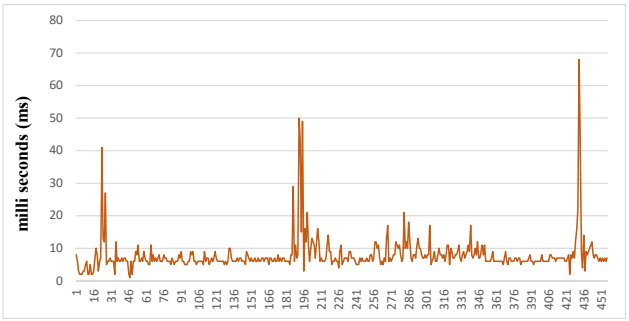

Fig. 2: Running time of the DMS policy enforcer function F_{zpe} in milliseconds.

nowadays, most devices are equipped with 16 or more gigabytes of storage space. Hence, DMS's storage requirement can be fulfilled by modern smartphones.

Table 4: Overhead of the context-aware SD card filesystem.

# of files	SD card fs	modified SD card fs
	time in seconds	
10	0.11	0.18
100	1.27	1.80
1,000	8.9	15.44

SD card overhead. The socket communication between the filesystem and the DMS native service introduces a delay in file operations. Every time an app creates a file or tries to open a file, the filesystem connects to the native service to acquire information about the current mode. To measure the overhead, we execute a shell script that creates, edits, and deletes files.

We run the script a number of times (writing 10, 100, and 1,000 files) and find that the overhead is negligible up to 100 files. No app will access more than a few files in a real-world scenario. Table 4 shows the comparison of time resulted from running the shell script.

5 Related Work

Researchers proposed different types of extensions to enhance the security of the smartphone operating systems. Several papers analyzed the Android permission model [8, 19, 27, 36, 37] and identified some of its shortcomings. Their study highlights that the permission model was coarse-grained and not very user customizable. In response, researchers proposed different types of extensions to the permission mechanism of Android. Most of the solutions proposed in the literature (e.g., [14, 17, 18, 28, 35]) require modification to the Android framework and/or the underlying Linux kernel. In contrast, a number of solutions [9–11] proposed an alternative approach that integrates security policy enforcement into the application layer. DMS belongs to the former category.

Lange et al. [24] implemented a generic operating system framework for secure smartphones called L4Android. Their framework hosts multiple virtual ma-

chines to separate secure and non-secure applications. Each VM hosts its own version of Android. L4Android mainly focuses on the security of the sensitive applications (e.g., financial and work-related apps). Moreover, it relies on the hardware virtualization support, which is not yet practical for smartphones. In contrast, DMS is designed for end users to provide a more secure phone to protect their resources (photos taken, location history, etc.).

Conti et al. [17] proposed CRePE that can enforce fine-grained policies based on the context of the phone. Similarly, Schreckling et al. [30] introduced a real-time user-defined policy enforcement framework for Android. The main drawback of these frameworks is that they require a lot of user control for their operation. In [33], Vecchiato et al. showed that the majority of the users neglect important and basic security configurations in Android. In DMS, security modes will be managed automatically. However, there will be options to modify the configurations for advanced users.

Smalley et al. [32] implemented the mandatory access control (MAC) in Android. They showed that the mandatory access control is able to thwart some of the well-known malware attacks reported in the literature. DMS differs significantly from MAC as it does not associate access control with the operating system users (normally apps in Android). DMS changes access control policies based on the detected security context and applies policies to a group of apps (the zones).

Zhauniarovich et al. [39] proposed a system called Moses that supports multiple security profiles on smartphones. Moses is based on the old permission model of Android and only supports a handful of restrictions and contexts. It creates a completely different persona for each context. DMS supports a comprehensive power-efficient security context manager and enables automatic switching to security modes. DMS uses the new permission model of Android that Google introduced in Android Marshmallow (version 6.0). DMS ensures security and privacy through smart restrictions without creating multiple personas. As a result, DMS is more resource efficient and users do not have to maintain separate app profiles for each persona. Moreover, Moses is designed from a perspective where corporates can create and manage security profiles. DMS's security modes are automatic and designed to protect the resources of end users.

6 Conclusion

In this paper, we present the design and implementation of Droid Mood Swing (DMS), an automated security mode management system for smartphones. DMS uses existing Android's permission model to implement a more secure and usable operating system. DMS can control application groups (called zones) through configuration files provided by device vendors. Application zones are a way to create app containers without any virtualization techniques which are heavy on hardware. Security modes are activated based on the security context of the phone to protect device resources in different use cases. DMS also implements an intent firewall, a context-aware file system, and a permission verification system.

DMS enables users to use a single device for multiple types of tasks securely. All operations of DMS are completely transparent to users.

Our implementation of DMS on a real device (Nexus 5) showed its effectiveness and minimal impact on user experience. DMS automatically takes security actions like restrict network, restrict IPC, restrict file access, and deny sending SMSs to a phone number. The permission verification system is able to identify 85% of the suspicious apps and ask users for additional verifications. Our security analysis proves DMS's effectiveness against over-privileged third-party apps and libraries. DMS is also effective against confused deputy and collusion attacks. In the worst case, DMS's policy checking incurs an 8ms delay and the delay caused by the SD card filesystem is minimal.

One limitation of DMS is that device vendors control modes and security policies which may be unacceptable by some users (for privacy reasons). In our opinion, for the general users, it is a good compromise to ensure security. Moreover, all these modes and configurations are editable by advanced users. We continue to work on the anonymization of the data sent from the device to maintain user privacy.

Acknowledgment

This work is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Canada Research Chairs (CRC) program.

References

1. Android kernel features. `http://elinux.org/{Android}_Kernel_Features`, accessed: 2017-08-03
2. Android permission. `http://developer.android.com/reference/android/Manifest.permission.html`, accessed: 2016-08-30
3. Android permission categories. `http://developer.android.com/guide/topics/manifest/permission-element.html`, accessed: 2015-11-09
4. Antutu benchmark. `http://www.antutu.com/en/index.shtml`, accessed: 2016-02-09
5. Filesystem in userspace. `https://en.wikipedia.org/wiki/Filesystem_in_Userspace`, accessed: 2017-03-09
6. Report: Android and iOS apps both leak private data, but one is definitely worse for the enterprise. `http://www.techrepublic.com/article/report-android-and-ios-apps-both-leak-private-data-but-one-is-definitely-worse-for-the-enterprise/`, accessed: 2017-03-09
7. Allix, K., Bissyandé, T.F., Klein, J., Le Traon, Y.: Androzoo: Collecting millions of android apps for the research community. In: Proceedings of the 13th International Conference on Mining Software Repositories. pp. 468–471. ACM (2016)
8. Andriotis, P., Sasse, M.A., Stringhini, G.: Permissions snapshots: Assessing users' adaptation to the Android runtime permission model. In: Proceedings of the International Workshop on Information Forensics and Security (WIFS). IEEE (2016)

9. Backes, M., Bugiel, S., Hammer, C., Schranz, O., von Styp-Rekowsky, P.: Boxify: Full-fledged app sandboxing for stock Android. In: Proceedings of the 24th USENIX Security Symposium. pp. 691–706. USENIX (2015)
10. Backes, M., Gerling, S., Hammer, C., Maffei, M., von Styp-Rekowsky, P.: Appguard–enforcing user requirements on android apps. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 543–548. Springer (2013)
11. Bianchi, A., Fratantonio, Y., Kruegel, C., Vigna, G.: Njas: Sandboxing unmodified applications in non-rooted devices running stock Android. In: Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices. pp. 27–38. ACM (2015)
12. Bogaerts, M.: Algorithm to calculate rating based on multiple reviews (using both review score and quantity). `https://math.stackexchange.com/questions/942738/algorithm-to-calculate-rating-based-on-multiple-reviews-using-both-review-score` (Sep 23 2014), accessed: 2017-09-09
13. Breiman, L.: Random forests. Journal of Machine Learning 45(1), 5–32 (2001)
14. Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., Sadeghi, A.R.: Xmandroid: A new Android evolution to mitigate privilege escalation attacks. Technical Report TR-2011-04, Technische Universität Darmstadt (2011)
15. Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., Sadeghi, A.R., Shastry, B.: Towards taming privilege-escalation attacks on Android. In: Proceedings of the Network and Distributed System Security Symposium (NDSS). The Internet Security (2012)
16. Cai, L., Chen, H.: Touchlogger: Inferring keystrokes on touch screen from smartphone motion. Hot topics in security (HotSec) 11, 9–9 (2011)
17. Conti, M., Nguyen, V.T.N., Crispo, B.: Crepe: Context-related policy enforcement for Android. In: Information Security, pp. 331–345. Springer (2011)
18. Enck, W., Gilbert, P., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation. pp. 393–407. USENIX Association (2010)
19. Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D.: Android permissions demystified. In: Proceedings of the 18th ACM Conference on Computer and Communications Security. pp. 627–638. ACM (2011)
20. Gorla, A., Tavecchia, I., Gross, F., Zeller, A.: Checking app behavior against app descriptions. In: Proceedings of the 36th International Conference on Software Engineering. pp. 1025–1035. ACM (2014)
21. Iqbal, M.S., Zulkernine, M.: Sam: A secure anti-malware framework for smartphone operating systems. In: Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC 2016). pp. 1–6. IEEE (2016)
22. Iqbal, M.S., Zulkernine, M.: Zonedroid: Control your droid through application zoning. In: Proceedings of the 11th International Conference on Malicious and Unwanted Software (MALCON). pp. 113–120. IEEE (2016)
23. Iqbal, M.S., Zulkernine, M.: Flamingo: A framework for smartphone security context management. In: Proceedings of the 32nd ACM Symposium on Applied Computing (ACM SAC). pp. 563–568. ACM (2017)
24. Lange, M., Liebergeld, S., Lackorzynski, A., Warg, A., Peter, M.: L4Android: a generic operating system framework for secure smartphones. In: Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices. pp. 39–50. ACM (2011)

25. Lin, C.C., Li, H., Zhou, X.y., Wang, X.: Screenmilker: How to milk your Android screen for secrets. In: Proceedings of the Network and Distributed System Security Symposium (NDSS) (2014)
26. Marforio, C., Ritzdorf, H., Francillon, A., Capkun, S.: Analysis of the communication between colluding applications on modern smartphones. In: Proceedings of the 28th Annual Computer Security Applications Conference. pp. 51–60. ACM (2012)
27. Nauman, M., Khan, S., Zhang, X.: 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. pp. 328–332. ACM (2010)
28. Russello, G., Conti, M., Crispo, B., Fernandes, E.: Moses: supporting operation modes on smartphones. In: Proceedings of the 17th ACM Symposium on Access Control Models and Technologies. pp. 3–12. ACM (2012)
29. Schlegel, R., Zhang, K., Zhou, X.y., Intwala, M., Kapadia, A., Wang, X.: Soundcomber: a stealthy and context-aware sound trojan for smartphones. In: Proceedings of the Network and Distributed System Security Symposium (NDSS). vol. 11, pp. 17–33 (2011)
30. Schreckling, D., Köstler, J., Schaff, M.: Kynoid: real-time enforcement of fine-grained, user-defined, and data-centric security policies for Android. in Information Security Technical Report 17(3), 71–80 (2013)
31. Seo, J., Kim, D., Cho, D., Kim, T., Shin, I.: Flexdroid: Enforcing in-app privilege separation in android. In: Proceedings of the Network and Distributed System Security Symposium (NDSS). pp. 1–53 (2016)
32. Smalley, S., Craig, R.: Security enhanced (se) Android: Bringing flexible mac to Android. In: Proceedings of the 20th Annual Network and Distributed System Security (NDSS) Symposium. vol. 310, pp. 20–38 (2013)
33. Vecchiato, D., Vieira, M., Martins, E.: Risk assessment of user-defined security configurations for Android devices. In: 27th International Symposium on Software Reliability Engineering (ISSRE). pp. 467–477. IEEE (2016)
34. VirusTotal: Virustotal is a free service that analyzes suspicious files and urls and facilitates the quick detection of viruses, worms, trojans, and all kinds of malware. `https://www.virustotal.com/` (2017), accessed: 2017-08-03
35. Wang, X., Sun, K., Wang, Y., Jing, J.: Deepdroid: Dynamically enforcing enterprise policy on Android devices. In: Proceedings of the 22nd Annual Network and Distributed System Security Symposium (NDSS'15) (2015)
36. Wei, X., Valler, N.C., Madhyastha, H.V., Neamtiu, I., Faloutsos, M.: Characterizing the behavior of handheld devices and its implications. Computer Networks (2017)
37. Xu, W., Zhang, F., Zhu, S.: Permlyzer: Analyzing permission usage in android applications. In: Proceedings of the 24th International Symposium on Software Reliability Engineering (ISSRE). pp. 400–410. IEEE (2013)
38. Xu, Z., Bai, K., Zhu, S.: Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors. In: Proceedings of the 5th ACM conference on Security and Privacy in Wireless and Mobile Networks. pp. 113–124. ACM (2012)
39. Zhauniarovich, Y., Russello, G., Conti, M., Crispo, B., Fernandes, E.: Moses: supporting and enforcing security profiles on smartphones. IEEE Transactions on Dependable and Secure Computing 11(3), 211–223 (2014)
40. Zhou, Y., Jiang, X.: Dissecting Android malware: Characterization and evolution. In: Proceedings of the IEEE Symposium on Security and Privacy (SP). pp. 95–109. IEEE (2012)