ZoneDroid: Control your Droid through Application Zoning

Md Shahrear Iqbal and Mohammad Zulkernine
School of Computing
Queen's University, Kingston, Ontario, Canada
{iqbal, mzulker}@cs.queensu.ca

***Abstract*—Research has shown that the android permission model was insufficient for providing protection against malicious behaviors of the untrusted third-party applications. To improve this scenario, Google modified the permission model in the recent Android version. However, in our analysis, it is still not an ideal option to enforce fine-grained access control.**

In this paper, we propose an extension and implementation of the Android permission model, ZoneDroid, to control a set of applications easily by creating multiple application zones (i.e., application groups). It is an approach to control application groups by modifying the Android permission model. All other previous approaches focused on restricting individual applications or creating separate user profiles. ZoneDroid minimizes security and privacy risks with a finer granularity of restrictions. Users can also control multiple devices using the cloud. Different zones (high privilege, trusted, new, restricted, etc.) have different runtime policies and enforce fine-grained access control. The ability to control application groups efficiently can be a valuable addition to the existing Android permission model.

Experiments show that ZoneDroid is effective against information leak and it can protect the device from becoming a part of a botnet. ZoneDroid offers much less user action when controlling multiple applications and its performance overhead is negligible.

I. Introduction

During the last few years, smartphones have been replacing traditional mobile phones. Smartphone sensors, third party applications and the availability of the mobile internet have increased the risk of security and privacy significantly.

Among all the smartphone operating systems, Android alone occupies over 86% market share in the second quarter of 2016 [1]. Moreover, Android-powered devices such as cars, fridges, televisions, point of sale (POS) terminals, and ATM booths are expected to flood user markets within a few years. Unfortunately, up until October 2015, Android's permission model was an "allow all or no install" model. There was no way to revoke a permission once an app is installed without uninstalling the app. From Android Marshmallow (version 6.0), Google adopted the iOS approach and now, some of the sensitive permissions can be revoked after installation.

Operating systems normally trust applications once they are installed by the users. However, in our view, different applications need different levels of trust. Sometimes, a group of applications may require a similar level of trust. For example, untrusted third-party apps require a stricter control over them than the apps from the trusted sources. Also, if a user has multiple devices, he or she may want to apply the same fine-grained policies to some of his or her other devices. However, the existing permission model of Android is not sufficient for providing all these features.

In [2], we proposed a secure anti-malware framework for the smartphone operating systems (SAM) based on the concept of smart cities. In the smart city wheel proposed in [3], a smart city has smart people, smart environment, smart government, smart economy, smart living, and smart mobility. Similar to a smart city, the phone will have smart apps, smart application zones, smart framework manager, smart access control, smart protection, and smart mobility. The framework manager will act as the government and other components will prevent malicious activities as well as monitor, report, and control malware.

In this paper, we propose an extension of the Android's permission model which we call ZoneDroid. ZoneDroid implements the concept of application zones which is a part of the prevention techniques proposed in our smartphone anti-malware framework. The separation of application zones is analogous to the separation of industrial and residential areas in an urban city. Each area may have their own security policy and a person has to adhere to the policies based on his or her location. ZoneDroid provides an efficient solution to control a group of applications easily (e.g., block a number of applications from accessing the internet from 12 am to 7 am with a few touch). It acts like a sandbox for a group of applications.

In an Android device with ZoneDroid, applications reside in any one of the zones and must adhere to the policies of that zone. Users can create and modify zones and policies and easily move applications from one zone to another. Any device of a user can control all of his or her other devices (e.g., control an Android tv's policy with the smartphone).

ZoneDroid can also be effective against a number of malware that steal users' confidential information or misuse the services of smartphones. Normal users are victimized by attackers who often disguise malware in seemingly useful apps. These malware compromise security and privacy, gain control of the user devices and often become a part of a botnet. Billions of Android devices can be used as a cyber weapon to launch large-scale attacks without the users' knowledge (e.g., click-fraud, DDoS, and email spam). Using ZoneDroid, users can block internet access to cripple the bots

while continue using the functionalities that do not require internet access. It is worth mentioning that internet access cannot be restricted using the current version (7.0 Nougat) of Android. An efficient fine-grained access control can also limit the effect of malicious apps in smartphones.

In particular, We design ZoneDroid to create multiple application zones with different levels of constraints and enforce fine-grained access control. ZoneDroid supports a way to easily manage multiple devices. Also, zone and policy configuration files can be shared between users. Novice users can then download configuration files and apply to their own custom zones. ZoneDroid offers an increased protection against a number of malware that try to steal private information or misuse smartphone services. We evaluate a prototype of ZoneDroid and the result shows that it adds useful features to the existing permission model while not affecting the performance.

The remainder of the paper is organized as follows. Section II provides the necessary technical background on Android's permission mechanism. Section III presents our proposed extension of the permission mechanism. We illustrate the design and operation of ZoneDroid in Section IV. We evaluate ZoneDroid in Section V and describe the related work in Section VI. Finally, we conclude in Section VII with a little discussion on the limitations and future work.

II. Permission in Android

Android uses permissions to protect components, system APIs, and resources. A permission is simply a unique text string. There are currently 138 permissions [4] defined in the Android operating system. In addition to the Android defined permissions, application developers can also declare their own customized permissions to protect their sensitive resources.

A permission can be associated with one of the following four protection levels [5]: **Normal**, **Dangerous**, **Signature**, **Signature-or-system**. According to `developer.android.com`, **normal** permissions are low risk permissions and **dangerous** permissions are for user's private resources.

Previously, at install-time, a user is shown with a list of permissions. The user must either grant or deny all of these permissions together. After the user approves the permission request and installs the application, the application owns its permissions throughout its lifetime. There is no way to revoke a permission other than to uninstall the app. However, from Android Marshmallow (version 6.0), Google changed the permission model. Now, "normal" permissions that an application asks are given at install time (the list of permissions is not shown to the user anymore). However, for dangerous or system 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 or system permissions later.

All "dangerous" permissions belong to some groups. To minimize user interaction, Android allows all the permissions belong to a group if the user allowed one of the permissions of that group previously. For example, there are seven permissions in the PHONE group. If an app asks for any of the seven permissions and the user allows it, then all the other six permissions in the PHONE group will be automatically allowed for that specific app in the future.

A. Permission enforcement techniques

In this subsection, we explain how the Android permission mechanism actually works to restrict access to resources. In Android, each application is assigned a unique user id (UID) in contrast to traditional desktop operating systems where applications execute with the privileges of the invoking user. Based on the UID, Android enforces access control rules which govern the application sandboxing.

Android 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, 16 permissions as defined in the platform.xml are enforced by the underlined Linux's discretionary access control (DAC) mechanism. We call these permissions **granted permission**.

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 set 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. For that reason, internet permission cannot be blocked by modifying the Android framework.

III. An extension to the permission mechanism

Our long-term research goal is to make the operating system malware-aware. There will be surveillance and detection components of the operating system that will use ZoneDroid to automatically assign zones to the installed apps and move them from one zone to another when necessary. This will minimize user interaction and require less user knowledge about the maliciousness of the apps. The system will ask users only necessary security decisions and avoid showing them all permission requests. In this work, we explore the problem space in two parts. First, we extend Android's permission model to accommodate fine-grained permissions. Then, we implement the model as a part of the Android framework. In this section, we describe how we extend the existing Android permission model.

To accommodate fine-grained permissions, we define rules, policies, application zones, and zone policy enforcer.

A rule makes an existing permission more fine-grained. For any permission, it allows to add a number of restrictions

based on time, phone number, folder path, or any custom counter (e.g., the number of SMSs sent).

Definition 1. ***Rule.*** *A rule r takes the form (o, V, e), where for a permission option 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>}, deny) restricts apps in a zone to send SMSs from 8AM to 5PM.

A policy consists of a number of rules. In other word, a policy can enforce fine-grained access control to more than one permissions.

Definition 2. ***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 a deny will prevail.*

Each application must belong to a zone where each zone enforces a number of policies.

Definition 3. ***Zone:*** *A Zone $z = (L, P, A_z)$ is defined by a label L, a set of policies P, and a set of Applications A_z. An application in the device can be assigned to only one zone at any given time. A zone association function $F_z(a) : A \rightarrow Z$ maps an application to a unique zone, where A is the set of applications and Z is the set of zones. Another function $F_p(z) : Z \rightarrow P$ returns all policies associated with the zone z.*

The Android `system_server` consults with the zone policy enforcer before allowing any permissions. It enforces fine-grained access control to all the applications that belong to a particular zone.

Definition 4. ***Zone Policy Enforcer****. The zone policy enforcer Z_{pe} defines the complete set of conditions under which an application a_1 in zone z is allowed to call another component c or system API s. a_1 can communicate with c or s if and only if either 1) there is no policy associated with the zone that denies the permission required by c or s or 2) there is no permission associated with c or s or 3) the permission associated with c or s is a normal permission and is already granted or 4) the user has already granted the permission required by c or s.*

IV. ZoneDroid design and operation

Based on the definitions described in the previous section, we implement ZoneDroid. ZoneDroid consists of three parts: the Android framework components, the ZoneDroid app and the multi-device management using the cloud. In Figure 1, we show the components of ZoneDroid and how they interact with each other. ZoneDroid framework components are ZoneDroid manager, zone and policy service, zone policy enforcer, and an SQLite database "Applications, Zones and Policies". The last three components are placed in the protected area of the framework. They cannot be accessed directly from the upper layer components. In addition, ZoneDroid adds a number of hooks to the activity and package manager. The ZoneDroid app is an administrative app that resides in the "High privilege app zone". Users can control their zones and policies and enable cloud synchronization using the app. Cloud enables a user to control multiple devices from any other devices. In the following subsections, we provide the details of these components.

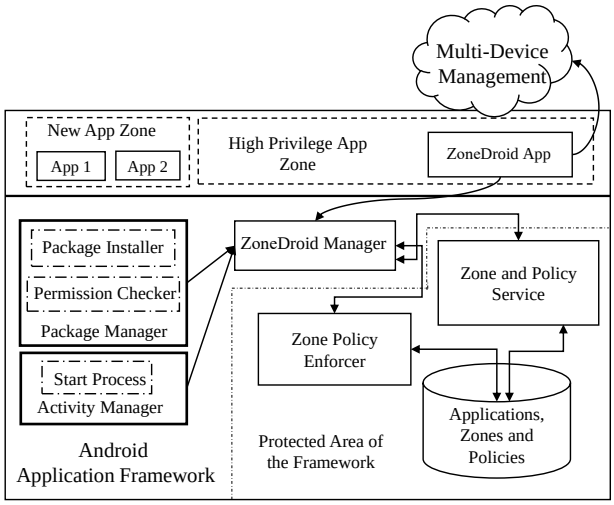

Fig. 1: ZoneDroid Architecture.

A. ZoneDroid Manager

In Android, apps cannot access system services directly because of the sandboxing. As a result, we implement ZoneDroid manager as an interface between the protected and external components (e.g., the ZoneDroid app). The manager and the zone and policy service communicate using the binder IPC mechanism. By default, when a user starts his or her phone for the first time, the ZoneDroid manager creates the following zones: New App Zone, Trusted App Zone, High Privilege App Zone, Restricted App Zone, Uninstalled App Zone. In the new app zone, a number of dangerous permissions are blocked. A user can send a malicious app to the "Restricted App Zone" where all the sensitive permissions are blocked. Here, we like to mention that the movement of apps from one zone to another is recorded in the SQLite database of ZoneDroid. We do not move the application's code or data physically.

B. Zone and policy service

The zone and policy service is implemented as a protected system service that starts with the `system init` and cannot be blocked or stopped by the user or any other third party apps. Stopping this service will crash the operating system.

The zone and policy service is responsible for creating/editing/deleting policies and zones. We store the zone information along with their associated apps and policies in the SQLite database.

C. Zone policy enforcer

The zone policy enforcer enforces the zone policies to the applications in that zone. We insert a number of hooks to the existing Android framework so that before allowing any permission, the Android permission mechanism calls the ZoneDroid manager to validate it. The manager, in turn, calls the `checkZonePermission` function of the policy enforcer to decide whether a given permission should be allowed based on the zone policies.

We also hook the Android package installer so that when a user installs an app, the package installer calls the ZoneDroid manager to register the app in the "New App Zone". The policy enforcer divides all the android permissions into three groups, namely, runtime permissions, granted permissions without the internet, and the internet permission. For each group, it enforces the policies in a different way. The details are given below.

1) Allowing runtime permissions: For all runtime permissions, Android verifies that the permission is granted for the application and it is not currently being revoked by the user. We implement a hook just before Android's own permission check so that the permission can be denied if it is not allowed according to the zone policies. Applications will not crash in this case. Rather, they will ask for the permission again gracefully. However, the access will be denied until the user removes the policy blocking the access or moves the app to a different zone where the permission is allowed.

2) Allowing granted permissions: ZoneDroid controls the "normal" granted permissions by hooking the process creation method of applications. It modifies the `gids` (Linux's supplementary group id) that the activity manager uses to fork `zygote` and create a new application instance. These `gids` are calculated from the application's requested permissions. We hook the function `startProcessLocked` from the activity manager service to remove all the `gids` that are not allowed in the zone. One problem with this approach is that the application will crash when it attempts to use the removed functionality. For example, if a user blocks Bluetooth for a set of apps, the activity manager will not send the "`net_bt`" group id to `zygote`. As a result, these apps will crash if they try to access Bluetooth in the specified time period of the policy.

3) Allowing internet permission: Although internet permission is one of the granted permissions, we treat this permission differently. In our view, it is a very dangerous permission that the users may want to restrict. However, we do not want the apps to crash as unsetting the `inet` group will make them crash everytime they try to access internet. Accessing internet for various reasons is very common in modern apps and we want the user to continue using the apps to access other functionalities. Also, we want a finer control over the internet connection similar to the runtime permissions. For example, we want to block internet for a specific time of day in a zone. To make this scenario possible, we use the time module of the `iptables` [6] tool. In this case, the apps will not crash. However, any internet request will timeout in the specified time period. Consequently, ZoneDroid does not allow the SET_TIME permission for any third-party applications so that apps cannot change the device's time. Since the modification of `iptables` rules requires root privilege, we implement a utility service for that. The service starts with the `system init` and performs the necessary modifications to the iptables rules. The zone policy enforcer sends the modification requests to this utility service.

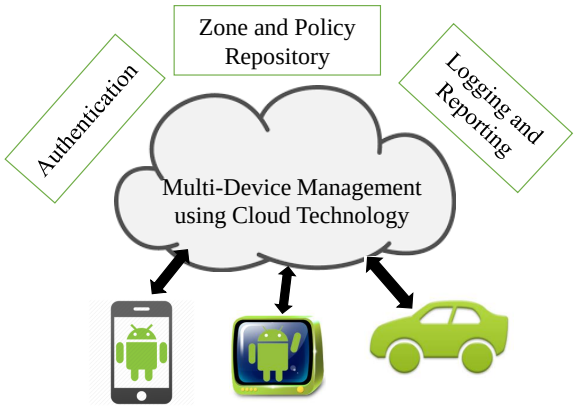

Fig. 2: ZoneDroid's multi-device management architecture.

D. Multi-device management

In Figure 2, we demonstrate the multi-device management functionality of ZoneDroid. There are three modules in this component, namely, **authentication**, **zone and policy repository**, and **logging and reporting**. First, we have the authentication module. It authenticates the ZoneDroid app so that the app can continue communicating with the cloud. The authentication module uses the device credentials and a password to authenticate a device. Second, there is a zone and policy repository that stores device specific zone and policy configuration files. A user will be able to view and edit zones and policies of any of his or her devices. Third, there is a logging and reporting module that logs all activities of the ZoneDroid app. Also, the framework components are programmed to send periodic health messages to the logging system. It is worth mentioning that the multi-device management functionality will only be available if the user turned on the cloud synchronization feature using the ZoneDroid app. When a user turns on the feature, the ZoneDroid app sends all the configuration files from a user's device to the cloud server. The server also notifies the ZoneDroid app when a new configuration is available for a device. Users can also share the policy files which can be downloaded and applied to a zone.

E. ZoneDroid app

The ZoneDroid app is the interface of ZoneDroid. Users can browse the existing zones and policies using the app. It provides functionalities to create/edit/delete zones and policies. The app keeps a list of permissions that the user can

block with rules. After creating a number of policies, a user can create multiple zones, each with different policies. Then, a user can move a set of applications to a zone to apply the policies to the set. By turning on cloud synchronization, users will be able to view and modify zones and policies of other devices too as described in the previous subsection. If the ZoneDroid app finds an updated version of policy or configuration files, it downloads the information and applies to the local database. Here, we like to mention that the Android framework always consults with the local database of ZoneDroid.

V. Implementation and Evaluation

In this section, we evaluate ZoneDroid in terms of access control, usability, and operational overheads. We implement a prototype of ZoneDroid by modifying the Android Open Source Project (Marshmallow version 6.0.0_r1 MRA58K as of 2015/10/17). We use ODROID c1+ [7] as our development board. ODROID c1+ is an ARM device from the company Hardkernel [8] and it has Amlogic Cortex-A5 1.5 GHz quad-core CPU, Mali-450 MP2 GPU, and 1 GB DDR3 SDRAM. We implement the multi-device management functionality using a Microsoft Azure PAAS Instance (Model D2: 2 Core, 7GB RAM, 100GB SSD, Windows Server 2012) and an Azure SQL DB Instance (2 TB).

In our opinion, ZoneDroid is easily deployable. 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 ZoneDroid framework components can be applied to the AOSP project using our patch which is only 502.9KB in size. Notably, we modified features which are only available from Android Marshmallow. As a result, ZoneDroid can 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.

Resource name	Protected by ZoneDroid	Protected by Existing Android
Bluetooth	✓	x
Internet	✓	x
Infrared	✓	x
Network State	✓	x

TABLE I: A limitation of the existing permission model in restricting access to certain resources.

A. Fine-grained access control

To demonstrate the limitations of the existing permission model and the effectiveness of ZoneDroid's fine-grained access control, first, we show a number of resources that ZoneDroid can protect while the existing permission cannot. After that, we show how fine-grained permissions add useful features to the existing permission model.

Table I lists a number of resources that the existing permission model cannot protect. These resources require permissions that belong to the "normal" permission group. In our view, these resources are sensitive and there should be a way to restrict access to them. ZoneDroid can protect these resources from the third-party untrusted apps using appropriate zone policies.

#	Scenario	Restricted by ZoneDroid	Restricted by Existing Android
1	Restrict a number of apps to send SMS/call to a list of numbers	✓	x
2	Restrict internet access to a number of apps from 11pm to 9am	✓	x
3	Block calls from a specific number from 11pm to 8am	✓	x

TABLE II: Examples of fine-grained access control scenarios that the existing permission model fails to offer.

As mentioned before, the existing Android permission model is not fine-grained. For example, a user may install a third-party audio recorder and a news app for extra features. However, he or she may not want to allow the access to microphone and internet all the time. There is no way to limit their access based on time using the existing permission model. Using ZoneDroid, a user can create a zone and a policy with two rules restricting the access. In Table II, we demonstrate a number of similar scenarios where fine-grained access control adds value to the existing model.

B. Usability

To show that ZoneDroid is easier to use, we choose a number of resources that we want to protect. These resources require dangerous permissions and enforced by Android's runtime permission check. We download 12 apps from `https://apkpure.com/` that access the resources. In Table III, we list the resources and the apps. In the existing model, the user has to perform permission revoking for each app separately (5 permissions for 12 apps, 60 taps). Using ZoneDroid, the user can create a zone with the selected apps and apply a single policy (with five rules, listed in Listing 1). This illustrates the fact that ZoneDroid can protect resources with fewer user interactions. No apps were crashed during this test.

```
1 {READ_PHONE_STATE,{<TIME_ALWAYS>,DENY}
2 {ACCESS_FINE_LOCATION,{<TIME_ALWAYS>,
      DENY}
3 {READ_CONTACTS,{<TIME_ALWAYS>,DENY}
4 {GET_ACCOUNTS,{<TIME_ALWAYS>,DENY}
5 {SEND_SMS,{<TIME_ALWAYS>,DENY}
```

Listing 1: A policy with five rules.

1) Attack scenarios where ZoneDroid can be effective: Since Android is used in a myriad of devices (smartphone, tv, set-top box, POS terminal, ATM machine, etc.), the malware threat is growing at a rapid pace. Nowadays, malware

Resource	Permission	Apps in the zone
IMEI	READ_PHONE_STATE	com.bigos.androdumpper, in.codeseed.audify, org.eyeslave.bdradio, com.llapps.blendercamera, com.turner.pocketmorties, com.appthink.deeplifequotes, HinKhoj.Dictionary, fm.clean, com.emu.dream, com.bookmark.money, com.chmodsoft.perfectvocal, de.ub0r.android.smsdroid
Phone #	READ_PHONE_STATE	
location	ACCESS_FINE_LOCATION	
contacts	READ_CONTACTS	
account	GET_ACCOUNTS	
SMS/MMS message	SEND_SMS	

TABLE III: Usability test of ZoneDroid.

writers want to misuse the existing functionalities (call, SMS, internet, etc.) of a device or sell user information.

ZoneDroid's application zoning and custom policies can restrict eavesdrop and information stealing. By default, ZoneDroid installs all the new applications in a restrictive "new app zone". Users need to manually move the applications to a different zone if he or she wants any particular high-privilege feature. This process makes the user more cautious about the app and its permissions.

Internet users are often victimized by malicious attackers. Some attackers infect and use innocent users' machines (by making them a part of a botnet) to launch large-scale attacks without the users' knowledge. Similar to the desktop computers, smartphones can also be a part of such botnets and help launch large-scale low noise attacks (e.g., DDoS, click-fraud). Botnets can be thwarted by restricting internet usage of apps that do not need the internet for their core functionalities. In existing version of Android, users can not block internet access. To show ZoneDroid's effectiveness against botnets, we create a simple number game which performs click-fraud [9]. We install the app in an Android TV which remains on 24/7. The existing permission model of Android will allow the app to perform illegal activities using the internet. ZoneDroid provides an easy way to block the internet for the app (running on the Android TV) from a smartphone. The app can continue to provide its functionalities while all its internet requests time out.

2) Apps built for older versions of Android: ZoneDroid does not negatively affect older apps. We install 21 apps built for older versions of Android and execute them in our device. As long as they are given the permissions they ask, the apps execute without any problem. However, from Android Marshmallow, older apps crash if any of the sought permissions is denied by the user. It is the behavior of Android from version 6.0 and has nothing to do with ZoneDroid.

C. Operational overheads

In this subsection, we evaluate ZoneDroid in terms of performance, power consumption, memory, and storage usage. In each case, we show that there is a very little to negligible overhead. ZoneDroid efficiently acts as a virtual sandbox for a group of apps.

We quantify performance using a popular benchmarking app (AnTuTu) available from the Android stores. 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. All numbers from the benchmarking app are averaged over 10 runs unless stated otherwise. We also evaluate the running time of the `checkZonePermission` function which is the only function that is called repeatedly while the device is running.

The benchmarking app runs concurrently with the standard set of Android 6.0 apps that launched at boot. Based on the official Android source code (6.0), these apps are launcher, contacts (and its provider process), photo gallery, dialer, MMS, and settings. Vendors customize this list and add more apps. On our ODROID c1+, there were 23 apps (including standard apps) on the device when we run the benchmarking app. We do not kill any pre-loaded apps.

Test Group	Test	Score	
		Stock	ZoneDroid
UX	Multitask	3022	3127
	Runtime	1357	1377
CPU	CPU integer	1546	1563
	CPU float-point	1477	1512
	Single-thread integer	1125	1151
	Single-thread float-point	952	973
RAM	RAM operatin	1378	1405
	RAM speed	1586	1554
GPU	2D graphics	869	892
	3D graphics	2474	2507
I/O	Storage I/O	1459	1680
	Database I/O	625	620
	Total	17870	18361

TABLE IV: Individual test scores from the AnTuTu benchmark app.

1) Performance: ZoneDroid's main performance overhead results from the zone policy enforcement mechanism. Everytime Android checks for a permission, our hook in the `checkPermission` function will execute the function `checkZonePermission` from the zone policy enforcer. To measure ZoneDroid's overhead, first, we run the AnTuTu [10] benchmarking app on the stock (unmodified) and ZoneDroid version of the Android marshmallow.

We run the tests a number of times and find no significant difference in the test scores. Table IV shows the comparison of scores resulted from the benchmarking app. It is clear from these scores that the overall performance is not hampered by activating ZoneDroid.

2) Power consumption: As smartphones are limited in battery capacity, we measure the overhead of ZoneDroid in terms of power consumption using the ODROID smart

power [11] which is a dc power source. However, it has a data output port that provides the consumed power at the rate of 10Hz. We install a contact app and access contacts every 3 seconds for a period of 24 hours for both the stock and the modified versions. We use a UI automation tool [12] to automate the process.

We find that the stock version consumed 6336 milliampere-hour (mAh) and the ZoneDroid version consumed 6720 mAh during that period. This is an extreme scenario where we access the zone policy enforcer every 3 seconds for 24 hours. Nonetheless, the difference in the measured power is within an acceptable limit in our understanding. Nowadays, smartphones are often equipped with a 3000 mAh battery and ZoneDroid will not hamper the battery life of an average user.

3) Memory and storage: With the current implementation, ZoneDroid does not incur any memory overhead. We modify only two system files, namely, framework.jar and services.jar. However, the changes in sizes (2.76K bytes for the framework.jar and 11.15K bytes for the services.jar) of the two files are negligible. To examine the size of the "Applications, Zones and Policies" SQLite database, we install a total of 99 apps and create three policies with different rules. We find that after installing the apps the size of the SQLite database is 49.1K bytes. Nowadays, most devices are equipped with 16 or more gigabytes of storage space. As a result, in our opinion, ZoneDroid's storage requirement is negligible.

VI. Related Work

Some research papers reported the analysis of the Android permission model [13]–[17] 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 enhance the security of the Android operating system. Most of the solutions proposed in the literature (e.g., [18]–[24]) require modification to the Android framework and/or the underlying Linux kernel. In contrast, some other solutions [25]–[30] proposed an alternative approach that integrates security policy enforcement into the application layer. ZoneDroid belongs to the former category.

Apex [31] modified the Android permission system to constrain runtime app behavior. However, there are a number of differences between Apex and ZoneDroid. First, the permission model for which Apex was designed is changed now. Also, the main objective of Apex is to restrict the usage of phone resources per application. We exclusively focus on the creation of application zones so that a user can control a set of applications easily.

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. Though the mandatory access control (MAC) provides a fine-grained access control, many Android malware are over privileged and they cannot be blocked by MAC.

Lange et al. [33] implemented a generic operating system framework for secure smartphones called "L4Android". Their framework hosts multiple virtual machines 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. Moreover, it relies on the hardware virtualization support, which is not yet practical for smartphones. Several other authors [24], [34], [35] proposed virtualization-based techniques to better sandbox Android apps that require modification to the Android framework. However, in Boxify [36] and NJAS [37], the authors proposed a sandboxed execution of Android apps on an unmodified system. Although ZoneDroid requires a little modification to the Android framework, it does not use any virtualization technique which is heavy on hardware.

Wang et al. [38] proposed an enterprise-level security policy enforcement mechanism DeepDroid, through which enterprise administrators can dynamically enforce fine-grained access control policies. Their design is specific to the enterprise environment and they enforce policies per application. A number of similar solutions target governments and enterprises [9], [24], [34], [39]. ZoneDroid is designed for the end users who have multiple Android devices and want to control a group of apps easily on all devices.

VII. Conclusion and limitations

Android is the most popular smartphone operating system and its usage is continuously increasing. Moreover, companies are releasing many other devices with Android. As a result, security and privacy is a major concern among Android users. Unfortunately, current Android security model does not allow a user to enforce fine-grained access control.

In this paper, we present an extension of the existing Android permission model, ZoneDroid, to group Android applications and implement the concept of zoning. It provides a virtual sandboxing mechanism for a set of applications with no performance overhead. Applications reside in any of the zones and ZoneDroid controls all of them by applying fine-grained policies. ZoneDroid can restrict access to resources based on time, phone number or folder location. We modify the Android permission mechanism and add a number of new Android framework components. We also develop a cloud backend for the system so that the users can view and control policies and zones of more than one devices.

Suspicious or untrusted apps can be grouped easily using ZoneDroid. Our experiments suggest that grouping and restricting (with fine-grained access control) similar untrusted third-party apps minimizes the risk of information leak and the risk of unknowingly becoming a part of a botnet. Additionally, it requires fewer user interactions to control apps when they are grouped.

ZoneDroid assumes that users are knowledgeable about the activities of the installed apps and will create zones according to their needs. However, ZoneDroid is a part of our anti-malware framework and we are currently working on adding monitoring and behavioral analysis functionality to

Android so that it can monitor the application behavior and automatically move apps to different zones using ZoneDroid.

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] V. Woods and R. van der Meulen, "Gartner says five of top 10 worldwide mobile phone vendors increased sales in second quarter of 2016," http://www.gartner.com/newsroom/id/3415117, accessed: 2016-04-20.

[2] M. S. Iqbal and M. Zulkernine, "Sam: A secure anti-malware framework for smartphone operating systems," in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC 2016)*. IEEE, 2016.

[3] B. Cohen, "What exactly is a smart city?" http://www.fastcoexist.com/1680538/what-exactly-is-a-smart-city, accessed: 2015-03-06.

[4] "Android permission," http://developer.android.com/reference/android/Manifest.permission.html, accessed: 2016-08-30.

[5] "Android permission categories," http://developer.android.com/guide/topics/manifest/permission-element.html, accessed: 2015-11-09.

[6] G. N. Purdy, *Linux iptables pocket reference*. O'Reilly Media, Inc., 2004.

[7] "Odroid c1+," http://www.hardkernel.com/main/products/prdt_info.php?g_code=G143703355573, accessed: 2015-07-09.

[8] "Hardkernel," http://www.hardkernel.com/main/main.php, accessed: 2016-02-03.

[9] M. S. Iqbal, M. Zulkernine, F. Jaafar, and Y. Gu, "Fcfraud: Fighting click-fraud from the user side," in *Proceedings of the 16th IEEE International Symposium on High Assurance Systems Engineering (HASE 2016)*. IEEE, 2016, pp. 157–164.

[10] "Antutu benchmark," http://www.antutu.com/en/index.shtml, accessed: 2016-02-09.

[11] "Odroid smart power," http://www.hardkernel.com/main/products/prdt_info.php?g_code=G137361754360, accessed: 2015-07-09.

[12] "Android view client," https://github.com/dtmilano/AndroidViewClient, accessed: 2015-07-09.

[13] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*. ACM, 2011, pp. 627–638.

[14] T. Vidas, N. Christin, and L. Cranor, "Curbing android permission creep," in *Proceedings of the Web 2.0 Security and Privacy workshop (W2SP) 2011*, vol. 2, 2011.

[15] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "Pscout: analyzing the android permission specification," in *Proceedings of the ACM Conference on Computer and Communications Security*. ACM, 2012, pp. 217–228.

[16] W. Xu, F. Zhang, and S. Zhu, "Permlyzer: Analyzing permission usage in android applications," in *Proceedings of the 24th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2013, pp. 400–410.

[17] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to android," in *Proceedings of the 17th ACM conference on Computer and Communications Security*. ACM, 2010, pp. 73–84.

[18] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "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*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 393–407.

[19] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.-R. Sadeghi, and B. Shastry, "Towards taming privilege-escalation attacks on android." in *NDSS*. The Internet Security, 2012.

[20] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, and A.-R. Sadeghi, "Xmandroid: A new android evolution to mitigate privilege escalation attacks," *Technical Report TR-2011-04, Technische Universität Darmstadt*, 2011.

[21] M. Conti, V. T. N. Nguyen, and B. Crispo, "Crepe: Context-related policy enforcement for android," in *Information Security*. Springer, 2011, pp. 331–345.

[22] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*. ACM, 2009, pp. 235–245.

[23] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel, "Semantically rich application-centric security in android," *Security and Communication Networks*, vol. 5, no. 6, pp. 658–673, 2012.

[24] G. Russello, M. Conti, B. Crispo, and E. Fernandes, "Moses: supporting operation modes on smartphones," in *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*. ACM, 2012, pp. 3–12.

[25] M. Backes, S. Gerling, C. Hammer, M. Maffei, and P. von Styp-Rekowsky, "Appguard–enforcing user requirements on android apps," in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2013, pp. 543–548.

[26] B. Davis and H. Chen, "Retroskeleton: retrofitting android apps," in *Proceedings of the 11th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2013, pp. 181–192.

[27] B. Davis, B. Sanders, A. Khodaverdian, and H. Chen, "I-arm-droid: A rewriting framework for in-app reference monitors for android applications," *Mobile Security Technologies*, vol. 2012, 2012.

[28] J. Jeon, K. K. Micinski, J. A. Vaughan, A. Fogel, N. Reddy, J. S. Foster, and T. Millstein, "Dr. android and mr. hide: fine-grained permissions in android applications," in *Proceedings of the 2nd ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM, 2012, pp. 3–14.

[29] S. Rasthofer, S. Arzt, E. Lovat, and E. Bodden, "Droidforce: Enforcing complex, data-centric, system-wide policies in android," in *9th International Conference on Availability, Reliability and Security (ARES)*. IEEE, 2014, pp. 40–49.

[30] R. Xu, H. Saïdi, and R. Anderson, "Aurasium: Practical policy enforcement for android applications." in *Proceedings of the USENIX Security Symposium*, 2012, pp. 539–552.

[31] M. Nauman, S. Khan, and X. Zhang, "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, 2010, pp. 328–332.

[32] S. Smalley and R. Craig, "Security enhanced (se) android: Bringing flexible mac to android," in *Proceedings of the 20th Annual Network and Distributed System Security (NDSS) Symposium*, vol. 310, 2013, pp. 20–38.

[33] M. Lange, S. Liebergeld, A. Lackorzynski, A. Warg, and M. Peter, "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*. ACM, 2011, pp. 39–50.

[34] J. Andrus, C. Dall, A. V. Hof, O. Laadan, and J. Nieh, "Cells: a virtual mobile smartphone architecture," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 2011, pp. 173–187.

[35] C. Wu, Y. Zhou, K. Patel, Z. Liang, and X. Jiang, "Airbag: Boosting smartphone resistance to malware infection," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Internet Security, 2014.

[36] M. Backes, S. Bugiel, C. Hammer, O. Schranz, and P. von Styp-Rekowsky, "Boxify: Full-fledged app sandboxing for stock android," in *Proceedings of the 24th USENIX Security Symposium*. USENIX, 2015, pp. 691–706.

[37] A. Bianchi, Y. Fratantonio, C. Kruegel, and G. Vigna, "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*. ACM, 2015, pp. 27–38.

[38] X. Wang, K. Sun, Y. Wang, and J. Jing, "Deepdroid: Dynamically enforcing enterprise policy on android devices," in *Proceedings of the 22nd Annual Network and Distributed System Security Symposium (NDSS'15)*, 2015.

[39] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.-R. Sadeghi, and B. Shastry, "Practical and lightweight domain isolation on android," in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM, 2011, pp. 51–62.