CoDRA: Context-based Dynamically Reconfigurable Access Control System for Android

Nanda Kumar Thanigavelan, Ethiopia Nigussie, Antti Hakkala, Seppo Virtanen, and Jouni Isoaho

Department of Future Technologies, University of Turku, Finland

Abstract

We present an access control system for Android that offers context-based dynamically congurable restrictions, ne-granular policy and ability to enforce various policy congurations at different levels of system operation. This system is called *CoDRA*. The fine grained policy and policy diversification are achieved through the application of context based on resource features. Policies are established and classified (system-wide and application-wise) after careful examination on application activities. The dynamic generation and enforcement of policies enables greater protection for open resources, e.g., sensors. *CoDRA* enforces different policy configuration per user through its integration of multiuser support in Android. A simple graphical control panel is provided for policy administration. *CoDRA* performance and overhead were analysed by testing 55 popular applications in Nexus 5 and 9 devices. The results proved that *CoDRA* successfully fulfilled its objectives by introducing 1-20ms executional overhead. It occupied about 800kB memory for policy storage and 5kB of memory for every additional user context space. The evaluation also proved that the tested applications did not exhibit any adverse effects during execution even with full restriction.

Keywords: Context-based Access Control, Android, Mobile Security, dynamic policy configuration

1. Introduction

Mobile devices have begun to dominate the market for communication and online service consumption among the individual users and organizations. Rapid growth in number and diversity of available applications further drives the popularity of mobile devices as the device of choice [1, 2]. Gartner predicts that by 2018 more than 50% of users prefer either a tablet or a smartphone as their first choice for all internet activities [3]. Apart from personal usage, organizations that apply Bring Your own Devices (BYoD) and Bring Your own Technology (BYoT) IT policies are encouraging employees to use their own devices and technology to access enterprise resources. The combination of device popularity, poor user security awareness, and vulnerabilities in the operating system (OS) make mobile devices a potential and lucrative target for attackers. Numerous security measures, such as anti-malware software, firewall, intrusion detection system, access control system and spam filters can be employed to protect mobile devices. This work contributes to improving the security of mobile devices through the development of context based highly fine-grained and dynamically reconfigurable access control system.

CoDRA is built on top of Android 6.0 version. Android is the most popular mobile OS with 40% of market share [4]. Android is a customized Linux-based open source mobile operating system [5] for mobile devices. The ability to access mobile device resources (like camera, messages, contacts and Wi-Fi information) through APIs has simplified the application development process resulting in a large number of applications. Easy resource accessibility along with an increase in application usage without sufficient security awareness may cause leakage of users private/sensitive information and financial loss. In addition, the use of third party libraries in application development paves way for abusing the permissions granted to the host applications [6].

There are efforts made by Android developers and the research community to address the security limitations. Android offers two major security features: sandboxing and the permission model. Sandboxing provides application isola-

tion through the Linux access control and process protection mechanisms while the permission model controls resource accessibility of the applications. In the permission model, numerous permissions are provided to regulate the resource accessibility [7] but they lack sufficient levels of granularity and appropriate context awareness. SE Linux was adopted to improve overall security of Android devices especially by strengthening the sandboxing. SE Linux offers better device protection but it is unable to deliver context sensitive resource controls, for example, restricting network association based on configured network security. Starting from Android version 6.0, the ability to revoke granted permissions and to request for approval upon first access were introduced, aiming to offer greater flexibility for users to control their devices. Yet, it fails to deliver a context aware highly refined policy for controlling device activities.

Android has attracted the interest of the research community, and plenty of research has been carried out to identify and improve on its security limitations [8, 9, 10, 11, 12]. Several systems are proposed with an intention of preserving privacy and security [13, 14, 15, 16, 17, 18, 19, 20]. These solutions operate either at system or application level but not both. In addition, they do not solve the issues with dynamic resource access, dynamic policy enforcement, and the policy revocation model. Furthermore, existing access control system lack the capability to prevent permission verification abuses and interface access. Though context aware access control systems have been proposed in the past, they commonly relies on environment contexts, such as location, time and connected networks [21, 22]. These approaches fail to capitalize on the context of other resource features, which can be further explored for the betterment of security and policy refinement.

In order to address the issues discussed above, we present *CoDRA* - an access control system that offers context-based highly fine-grained policy and dynamically reconfigurable control to enforce various policy configurations at different levels of system operation. *CoDRA* is an extension of the current Android permission model. It is designed, implemented and tested within the Android operating system in order to demonstrate *CoDRA*'s capability to enhance se-

curity and privacy with minimal overhead. The main contributions of *CoDRA* are as follows:

1. context-based control: a security system that imposes restrictions based on resource contexts. Resource contexts are identified and their appropriate OS handlers are modified to accommodate and apply the restrictions through policies.
2. policy classification: a set of policies are developed by examining the activities of the device and its installed applications. They are classified either as system-wide or application-wise to simplify the policy enforcement process and diversification.
3. dynamic policy generation and enforcement: introduction of dynamic enforcement along with the static enforcement on resources leads to greater control over a device.
4. enhanced permission revocation: prevention of permission verification abuse and exploitation of forced user consent with policy controls that allow user defined provision of fabricated permission data to applications.
5. secure multiuser environment: creation of several policy sets in order to have customized control for each user by capitalizing on multiuser support in Android.

The rest of the paper is structured as follows. The motivation and the objectives of *CoDRA* are discussed in 2. The assumed threat model is presented in Section 3. In Section 4, the components of the *CoDRA* along with its software architecture are presented while policy design and implementation are explained in Section 5. The evaluation of *CoDRA* and a comparison with related approaches are discussed in Sections 6 and 7, respectively. Finally, conclusions are presented in Section 8.

Table 1: Example of refined control and its outcomes

Example activities	**Required changes**	**Desired outcomes**
Connecting only to secure networks	Network association enforcement based on security attribute, e.g., EAP or WPA2	No accidental use of insecure networks and better control over network association
Controlling application installation based on its origin	Restriction on installation process using origin attribute	No installation of malicious applications through side installation channels
Preventing device fingerprinting and accessing sensitive data	Control over resource access	No unauthorized resource access
Preventing system configuration changes by other users	User based access restriction on system settings	Creation of secure multiuser environment

2. Motivation and Objectives

2.1. Motivation

Though Android has enhanced its security in the last three versions, still it has limitations that can be exploited by adversaries and developers. Information about the user, and the data that are being stored or processed by an application, can be gathered by analyzing the device activities. The analysis process helps adversaries and developers to construct a behavioral data that may reveal the users identity, interest and behaviors, which results in privacy violations. Using the behavioral data, an adversary can devise a specific attack strategy against a user. There are several scenarios where existing security measures fail to achieve policy granularity, dynamic enforcement of policies, and greater control over resources. A few scenarios and the corresponding actions that are required to be implemented for improved security are listed in Table 1.

The security implications of the following limitations stimulated us to design and implement *CoDRA*.

2.1.1. Unsecured/Open Resources

The normal ways of device fingerprinting are based on IMEI, mobile number, Wi-Fi address and other relevant information. However, adversaries can employ other means to study the mobile usage and user behaviors based on the following activities:

- Numbers of application installation/uninstallation including its type such as gaming and social networking.
- Number of Wi-Fi connections, including network name and security.
- Mobile usage patterns based on the USB connection for charging or file transfers and also on screen active and inactive periods, and
- Information from sensors such as accelerometer, gyroscope and magnetometer.

These activities can be easily monitored by any application developer, as they do not require any permission. Even if they need prior permissions, they are not graded as dangerous by permission model. In addition, Android also has open resources, such as sensors, which may result in security breaches and privacy violations if they are used without permission. An example code listing for obtaining the list of system and user installed application is given in the listing 1.

Listing 1: Code of obtaining installed applications in Android

```
PackageManager packageManager = this.getPackageManager();
List<PackageInfo> packageInfoList =
    packageManager.getInstalledPackages(0);
for (PackageInfo packageInfo : packageInfoList) {
  Log.i(TAG, ((packageInfo.applicationInfo.flags &
      ApplicationInfo.FLAG_SYSTEM) != 0) ? "System App - " : "User
      App - " + String.valueOf (applicationInfo.loadLabel
      (packageManager)));
}
```

Several studies have been conducted to prove the exploitation of sensor resources [23, 24, 25, 26, 27, 28, 29]. The authors in [23, 24, 25, 26] have proved that the location of the users can be determined from unprotected sensor information. It has also been shown that user activities such as walking and jogging can be inferred using the same information [27, 28, 29]. In broadcast, only critical broadcasts are secured with permissions and the rest are unsecured such as battery-related, package-related and user-related. The authors in [30, 31] showed that most of the malware also targets non critical broadcasts. The usage or exploitation of these resources without user consent may have serious security and privacy implications.

2.1.2. Permission Classification

Android's system permissions are classified into four protection levels: normal, dangerous, system and signature [7][32]. Normal level stands for low-risk permission with minimal risk to the system, the user or other application. It is granted automatically without user's explicit approval. Dangerous level is defined as high-risk permission that can negatively impact the user and may not be automatically granted to the requesting application [33]. Under the given definitions, permissions to access messages, contacts and location are classified as dangerous as they can cause serious privacy issues. Unfortunately, each permissions are separately treated and classified based on individual impact. If they were assessed collectively, several permissions might have been appeared in the dangerous category. For example, ACCESS_WIFI_STATE permission is categorized as normal. It is required to scan and retrieve list of (configured) networks [34]. Using the obtained network information along with the help of location services [35] and sensor information, an application can determine the user location without having location access. Two scenarios are explained through the given example: a) privacy threat due to permission classification and b) acquiring location and users physical activities without obtaining location permissions.

Listing 2: Code of misusing permission verification mechanism

```
private static int READ_CALL_LOG_CODE = 3;
//verify the availability of granted status
if (checkSelfPermission( Manifest.permission.READ_CALL_LOG)
  != PackageManager.PERMISSION_GRANTED) {
  //if unavailable, prompt user to grant
  requestPermissions(new String[]
      {Manifest.permission.READ_CALL_LOG}, READ_CALL_LOG_CODE);
}
//Will be called after the user response
@Override
public void onRequestPermissionsResult(int requestCode, String[]
    permissions, int[] grantResults) {
  super.onRequestPermissionsResult(requestCode, permissions,
      grantResults);
  if (requestCode == READ_CALL_LOG_CODE) {
    if (grantResults.length > 0
      && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
        Log.i(TAG, "Permission granted");
    } else {
      //displays floating message
      Toast.makeText(getApplicationContext(), "Application cannot
          run without permission", Toast.LENGTH_LONG).show();
      //closes the application
      finish();
    }
  }
}
```

2.1.3. Abuse of Permission Verification

Android provides methods to detect current status of the permission, such as checkCallingOrSelfPermission() and checkCallingPermission(). The best programming practices for developers on the usage of permission are also provided by Android in order to follow them during application development [36, 37]. It is not possible to expect that everyone will follow the recommended practices. Therefore, it is highly possible that the provided verification methods can be

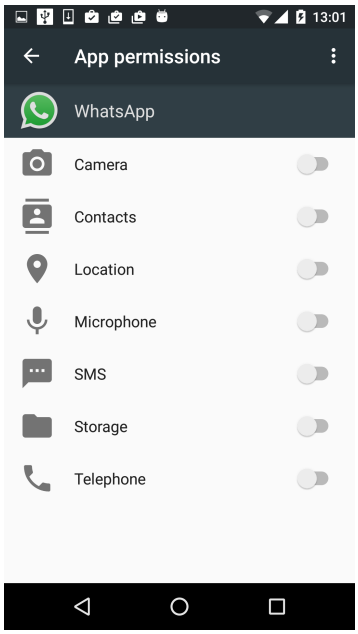

Figure 1: Revocable permissions

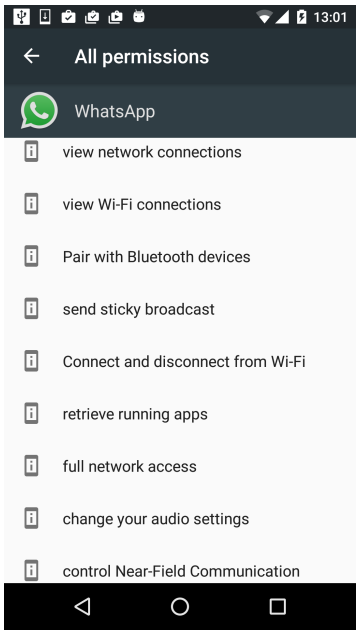

Figure 2: Non-revocable permissions

misused to force users to grant all the requested permissions to use the application. An example code list that prevents the usage of an application until user grants access to call log is given in listing 2.

2.1.4. Runtime Permission Revocation

Starting from Android 6.0, a runtime permission revocation feature is introduced to empower users. Using revocation feature, users can change the status (grant or revoke) of a permission required by an application at any time. Unfortunately, the permissions that are classified as dangerous can be controlled through the revocation mechanism. For example, RECEIVE_BOOT_COMPLETED permission is required to detect the device (re)boot event. It cannot be revoked since it is categorized as normal. The revocable and partial list of non-revocable permissions of WhatsApp application is given in the Figure 1 and Figure 2 respectively. Furthermore, the runtime revocation feature can easily be negated through the abuse of permission verification methods.

The above discussed limitations can be easily implemented by any developers

or adversaries to obtain sensitive information that may lead to privacy and security violations. *CoDRA* is aimed to overcome the above discussed limitation and to prevent the exploitation of resources. In addition, *CoDRA* needs to have multiuser support in order to enforce different policy sets on each user account. Access control systems supporting a multiuser environment is necessary since a user can maintain separate user accounts based on his/her needs and also may share the device with others.

2.2. OBJECTIVES

The underlying objective of *CoDRA* is to provide an effective security extension to Android operating system with low operational overhead. The design of *CoDRA* is guided by the following objectives:

1. *Context-based refined polices:* to model system policies based on resource usage as well as features that leads to achieve higher granularity in policies.
2. *Preventing permission verification abuse:* to provide a novel mechanism that prevent and overcome the possible threats which force users to grant all required permission to the application in the Android versions 6.0 and higher. This is necessary as it will negate the abuse of runtime grant or revoke permission feature.
3. *Dynamic restriction enforcement:* to bring every resource under restriction control (including the open resources). The enforcements will be created and applied dynamically upon on initial access.
4. *Dynamic policy configuration:* to improve the runtime revocation feature by allowing users to revoke both normal and dangerous category permissions.
5. *Retain existing security measures and exploit latest functionalities:* taking advantage of the new and improved Android functionalities while preserving the existing security features for the benefit of the entire system.

Moreover, an acceptable operational overhead and a control panel for policy administration are the additional objectives of *CoDRA*.

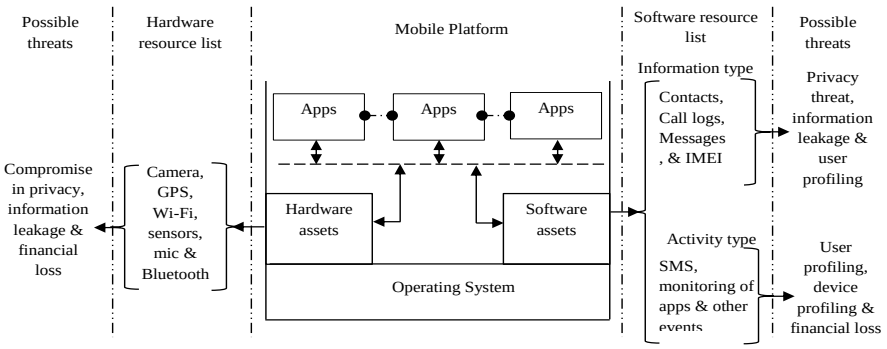

Figure 3: Device activities and its exploitation

3. Threat Model and Assumptions

In *CoDRA*, it is assumed that any application which tries to access any of the devices resources is a threat to the device operations and to the information available in the device. An adversary can learn the device or user behavior besides device identification by accessing sensors information, verifying installed applications, frequency of charging the battery and display active or inactive period. The possible threats from analyzing device activities are illustrated in Figure 3. These behavioral patterns may help adversary to devise user specific attack strategy like social engineering attack who are using social network applications frequently. Hence, any activity related to device or user fingerprinting is classified as malicious.

The application developers can take advantage of existing permission verification means to obtain user consent for all requested permissions in order to use the application. They can also employ other means to acquire sensitive information without using appropriate permissions. In multiuser operation, the secondary users may modify the settings that may cause security breach. To avoid these scenarios, we assume that the possibility of exhibiting malicious behaviors (for example, over-claiming of permissions) by application developers or by any device users other than the primary user is highly likely.

In *CoDRA*, it is assumed that device rooting is a strong adversary. That is, the device will not be rooted in any circumstances and no application is granted with root permission. We also assume that users and administrators

who are configuring polices are fully trusted and they are required to possess the minimum required knowledge on system functionality and its principles. For example, the users are able to understand the security protocols (WEP, WPA2 and EAP) used in wireless networks, use of contacts, sensors and broadcast events. Finally, we assume that the stacks of Android operating system including all services, system applications, kernel space and *CoDRA* are fully trusted.

4. SYSTEM ARCHITECTURE

4.1. CoDRA and its components

In Android, all requests are forwarded to their appropriate services, handled by the lower layers. The requests from the applications are directed towards the standard libraries which in turn initiate system calls to the kernel. This request handling flow reveals the critical focal point for the implementation of enforcement system. For example, Android has provision to install applications in different ways, for instance, Google Play and side loading. Every installation requests are forwarded to the PackageManagerService, which is responsible for application installation. In this case, *CoDRA* will intercept the installation calls at appropriate places and decides further course of action as per the defined policies. The customized architecture of the Android with *CoDRA* and its interaction with various components are shown in the Figure 4. *CoDRA* has five major components: control panel, handlers, enforcers, synchronizers and repository.

Control panel is implemented as a system application to provide a simple interface for policy administration. The control panel operation relies on synchronizers as it operates at the application layer.

Synchronizers acts as a liaise between the control panel and the other blocks of the system. Every control panel requests are directed towards the synchronizers irrespective of its purpose. Since control panel sits on top of the Android Framework, it is not allowed to access or modify contents in the system folders.

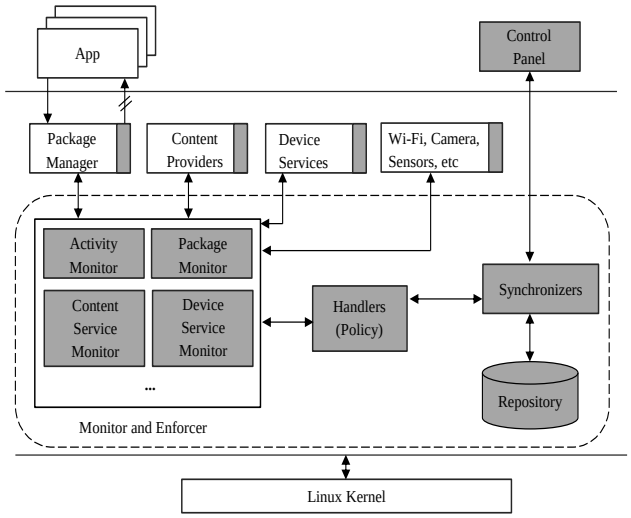

Figure 4: Customized Android Architecture with *CoDRA*. Fully shaded boxes represents newly added components and partially shaded represents modified components

In order to overcome the access restriction hurdle, it directs the requests to the synchronizers which in turn execute the requests on behalf of the control panel and return the results. Synchronizers do not accept request from any other applications except the control panel.

Handlers mainly comprises of policy handlers. It plays a crucial role in the system since the communications towards policies has to be passed through the handlers. Policy handlers are responsible for providing appropriate decision for the enforcers to act upon the intercepted resource request calls. Whenever an application tries to access a resource by calling API methods, the request access are intercepted after they passed the Android default security model and forwarded to the policy handlers by the resource monitors. The policy handlers return the status for the request based on the configured policies.

Monitors and Enforcers are primarily responsible for strengthening the security of devices by monitoring the request calls. Enforcers perform most of the core functionalities of the *CoDRA*, such as policy adherence verification and returning spurious information. It has dedicated monitors for system services in order to extend the services normal operations and to establish relation with the *CoDRA*.

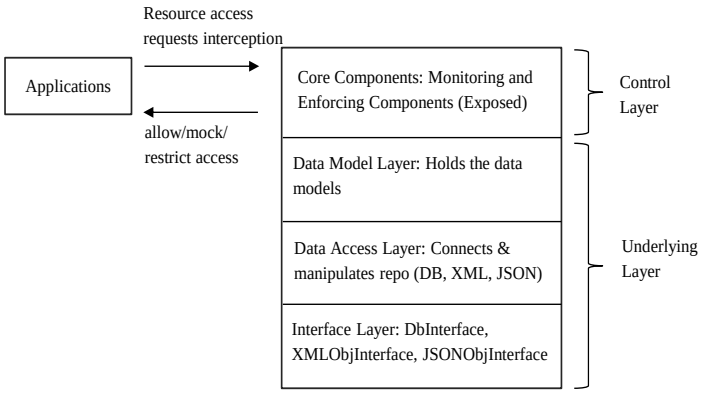

Figure 5: Access Control Software Architecture

Repository for storing and managing policies. Though the repository does not perform any functions, it plays an important role in the architecture as the other components rely on the information available in the repository to execute their intended actions.

4.2. Software Architecture

CoDRA is implemented as a layered architecture in order to reduce the complexities in development, maintenance and future enhancements. It allows to have complete control over the individual components and the entire system. With layered architecture, system components can be added or removed easily. Furthermore, it helps to restrict the lower layers visibility from the higher layers. The software architecture of *CoDRA* is shown in Figure 5.

The entire software architecture is grouped into two layers: Control and Underlying layers. The Control layer contains the implementation of the handlers, enforcers and synchronizers. The visibility of the components except the monitoring and enforcing tasks are hidden from other applications and services. Enforcers components are hooked into the Android system services while synchronization component provides interface to the control panel.

The underlying layer contains the basic classes required to manipulate the repository. It has three sublayers: Model, Access and Interface. Model sublayer represents the data models of the database tables and XML structures. The Access sublayer manipulates the repository through object references that are

provided by the Interface sublayer. The Control layer components use the data models to store or retrieve the information through the Access layer methods

5. Policy Design and Implementation

Policies are essential for any access control system as it determines the efficiency, acceptance and usage of the system. To design a policy, there are three critical factors that need to be established:

1. Source identification
2. Determining the target that requires protection and
3. Means to reach the target

The target is always the device in the mobile security environment, to be precise, the information available in the device is the target. Communication, information management and possible misuses are the primary purposes to reach the target. The sources are the applications residing in the device and they are also responsible for information generation as well as storage. In multiuser device, source can also includes the users as well. Hence, a policy has to be designed in a flexible manner to control most of the ways to reach the target while allowing the source to perform its legitimate operations.

5.1. Policy Design

In *CoDRA*, the policy is designed on the basis of context. Context-based access control systems are not new to the computer security. Context can be related to any information that represents the circumstances of an entity, such as identity, location and activities, and it may affect the definitions of the systems at different levels [38]. Though the perspective of the context can be different, researchers have conducted several experiments and proposed different models [39, 40, 41, 42, 43]. In our work, the concept of context is defined as an "attribute/feature role" of an entity unlike the "environment role" as in [21, 44].

In environment context based access control systems, policies are designed on the basis of the operating environment of an entity. The operating environment

Table 2: Example: context-aware policies for Wi-Fi and contacts information

	Wi-Fi	**Contact information**
Environment based context	Disconnect Wi-Fi if the battery is lower than 10%.	Deny contacts access if the application uses internet.
Attribute based context	Connect Wi-Fi only if it has EAP security.	Deny applications to access contact information if they are sourced from Google Play and has internet.

can be a place where the device is located at a given point of time, available energy for its operation or the connected network. In *CoDRA*, policies are built on the basis of an attribute or an operation that any applications can use or perform. To have a clear distinction on the adapted meaning of context between *CoDRA* and environment based works, two example cases are considered and presented in Table 2.

In the table, the policies for network association and accessing contacts information are given for the two context types. Wi-Fi policy based on environment context states that the network association is possible only if the device has sufficient battery level while the feature based policy dictates that the network association is not possible if the network has failed to exhibit or satisfy the given attribute, that is, the configured security (EAP) to safeguard the network. As can be seen from Table 2, the contact policy using the environment role directs a system to prevent any application from manipulating contacts information if it requires internet for its execution whereas the feature based policy specifically prohibits the applications to access contact information that are installed from Google Play and has internet permission. These two example

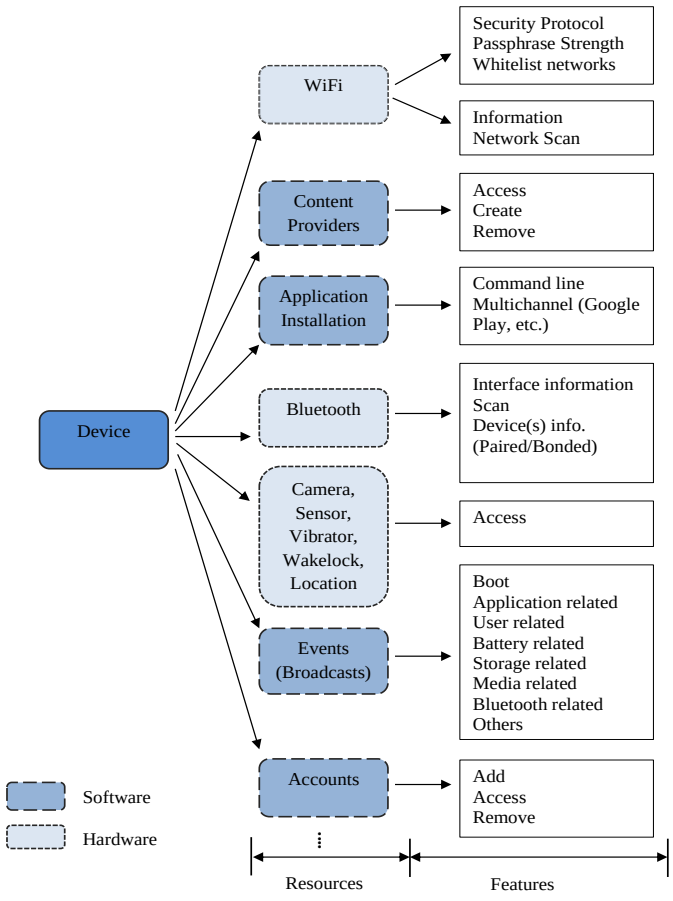

Figure 6: Relation between the resources and its features

cases clearly illustrate that the approach of *CoDRA* policy design is different from the environment role based approaches. Furthermore, they also show the degree of granularity and the difference in policy construction between the two approaches. With *CoDRA*'s refined granularity, it is possible to apply a restriction on certain feature of a resource.

In *CoDRA*, the hardware and software components, such as Wi-Fi, Bluetooth, sensors, camera and application installers are collectively referred as "resources". Entities are defined as the list of resources available in the device. Controlling access to the resources alone does not guarantee the expected policy granularity. In order to achieve context-based highly refined policies, the key traits of resource activities are classified as "features". These actions allow to establish policies, which conform to *CoDRA*'s context definition. Example list of resources and its features are shown in Figure 6. In order to simplify the policy design and implementation, the activities of the applications and device

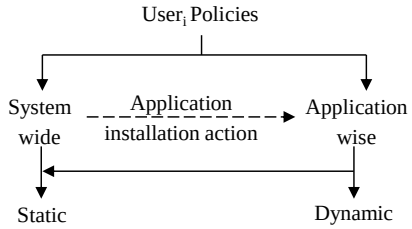

Figure 7: Classification of enforcement and its relation on device user accounts basis

as well as the type of enforcements are classified.

5.1.1. Activity Classification

To achieve control over the device, application activities along with other device activities has to be regulated. In *CoDRA*, activities are classified into two: application level and system level. Application level activities are operations required by any application for its execution while system level activities refer to the operations that affect the entire device. For example, application installation is a system level activity. Application level policies are useless in the installation process as the existence of an application is determined by the outcome of an install action.

5.1.2. Enforcement Classifications

Enforcements are classified based on their type and activity. The different enforcement classes in *CoDRA* are given in Figure 7. Application-wise enforcement regulates the activities of an application while system-wide enforcement controls the activities that cause system wide impact. The advantage of having activity based enforcement classifications is to achieve flexible control over the resources. Depending on the preference, a user can enforce either all or specific enforcement category.

In Android, to obtain MAC or IP address, an application has to obtain ACCESS_WIFI_STATE permission whereas the list of installed applications or sensor information is open to any application. Open resource access do not possess any threat to the user or device operations but when they are combined with secured resources, the possibility of increase in threat level is highly likely.

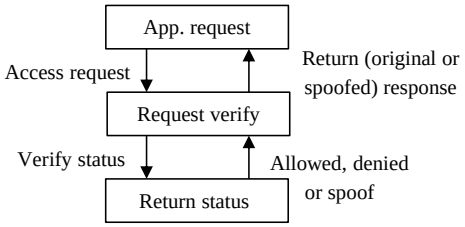

Figure 8: Enforcement - Static Approach

In order to treat both secured and open resources equally, the enforcements are further grouped into two classifications: static and dynamic (see Figure 7). Dynamic enforcement is useful for applying restrictions during the initial access request and to secure the resources that are not protected by the Android Permission model. Static enforcements are applied to the protected resources during application installation. The execution flows of static and dynamic enforcement are given in Figure 8 and 9. The dynamic approach creates policy upon an initial resource access even for the protected resources. Additionally, it requires a supplementary task for policy existence verification to determine the presence of a policy. In static approach, the additional task is not required and each application will have a set of policies corresponding to the protected resources prior to the execution. It is possible to avoid static enforcement and employ only dynamic enforcement throughout *CoDRA* as it has greater advantage than static. But to avoid an unnecessary additional task and to have a minimum number of policies prior to the execution of any application, both approaches are adopted.

5.2. Policy Implementation

The policy implementation of the *CoDRA* is accomplished by intercepting the resource access request calls at appropriate places. *CoDRA* maintains its own repository in the system partition for policy storage and management. It will be created during the first boot and the required permissions are applied to secure the repository from being accessed by other services. During repository

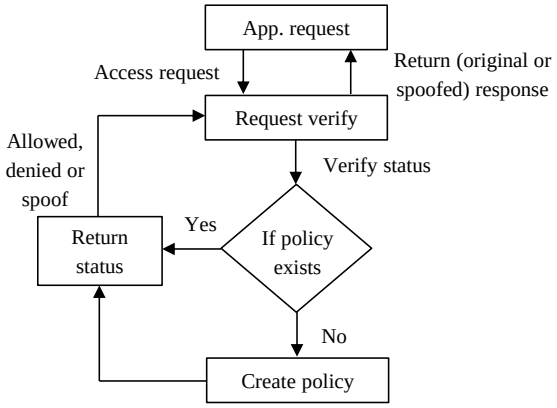

Figure 9: Enforcement - Dynamic Approach

creation, it will create a template for application-wise enforcement and a common structure for system-wide restrictions. By default, the status of all policies is set to active. The template can be changed through the control panel. This marks the initial point for system-wide enforcement.

When an application installation is detected, *CoDRA* registers a callback interface and temporarily suspends further execution of the installation process. Then, it will create a separate policy structure from the application details, such as UID, package name and permissions using the enforcement template in the repository. After a successful creation of policy structure, the callback allows the installation process to resume its execution. This action marks the initial point for application-wise restriction.

The dedicated monitors and enforcers attached to the resource will capture the request calls. It will forward the necessary information (i.e., UID, USERID, package name and requested resource feature) to the handlers. The handlers will verify the policy against the request call and return the decision. Based on the received decision, monitors and enforcers will perform the operation (such as returning genuine/spurious results and restrict access) and returns the status to the application. In addition, monitors and enforcers are also responsible for proper functioning of the applications. They will return the status so that the receiving applications assumes that the initiated request is

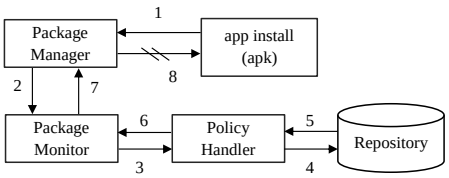

Figure 10: System wide enforcement: application installation restriction. The numbers 1-8 represent the modified execution flow.

completed successfully. This step is very important for access control systems as restricting requests should not create any malfunction.

System-wide enforcement covers events such as device wipe, network association, user creation and application installation. The process of system wide enforcement is described using an example of application installation. The restriction on application installation is given in Figure 10. The PackageInstallerActivity in packageInstaller application will be called to install the application, it will call InstallAppProgress which in turn hand over the control to PackageManagerService (1). PackageMonitor intercept the process and determine (2) the next course of action. The PackageMonitor will access the appropriate policy in repository through Handlers (3, 4, and 5), and receives the decision from Handlers (6). Then, the received decision is forwarded to the PackageManagerService (7). It will either proceed or abort installation (8) depending on the received status. Similarly, the rest of the request calls to appropriate services are intercepted and tested against configured restrictions.

Application-wise enforcement applies to any application activities. It can be either static or dynamic enforcement. The request calls of the application activities are intercepted by the monitors at appropriate places. For instance, request to access call logs are intercepted by the monitor attached to the content providers service. It will verify the status of the request against the given rule and decide further actions. For example, in dynamic case, when an application tries to register a listener for events (like ACTION_PACKAGE_ADDED and ACTION_PACKAGE_REMOVED), monitor will check the rule existence. If the rule is unavailable for the application, it will create a new rule and attach to the application.

Multiuser Support CoDRA takes advantage of the multiuser feature in An-

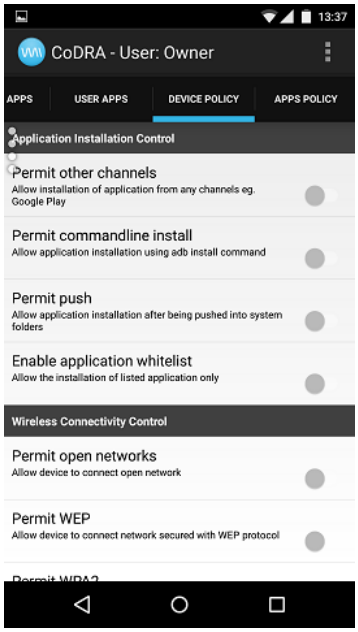

Figure 11: *CoDRA*: Device Policy Tab

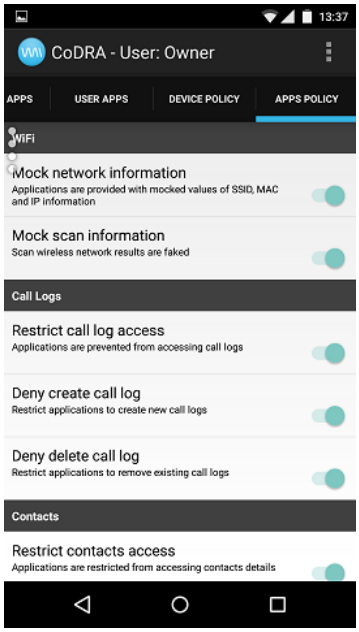

Figure 12: *CoDRA*: Apps Policy Tab

droid to fulfill the mutiuser policy configuration requirement. Since UserManagerService is responsible for handling all communications related to multiuser, a monitor is attached to intercept the incoming request and apply restrictions per user. Imposing restriction on applications at user basis becomes simpler because USERID can be obtained from UID as every request call contains UID.

5.3. Implementation of Control Panel

The design of the control panel plays an important role in the usage of *CoDRA*. Accessibility of the control panel, policy administration and maintenance through the control panel has to be relatively easy. The control panel is implemented as a part of the operating system and the configuration of policies, (i.e., activation and de-activation), is designed as a simple switch.

Tab based user interface is employed in order to provide applications information (including system applications) as well as to simplify the configuration of system-wide and application-wise enforcements. *CoDRA*'s control panel has a total of four individual tabs and each tab represents a unique operation. The operations of all tabs are described briefly as follows:

1. Tab 1 system apps: displays system applications.

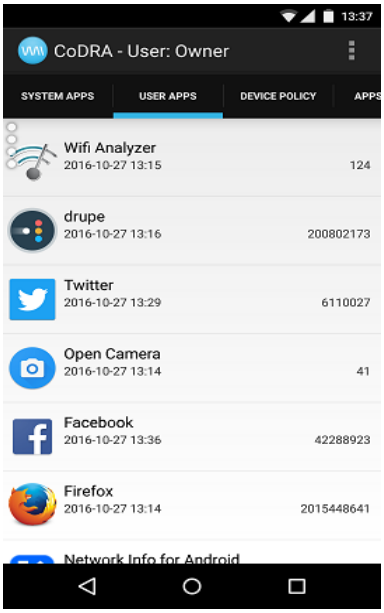

Figure 13: *CoDRA*: User Apps Tab

2. Tab 2 user apps: lists the installed applications and provides interfaces to configure application policies.
3. Tab 3 device policy: provides interfaces to configure system-wide enforcements.
4. Tab 4 apps policy: provides the template for application-wise enforcement that are used to create policies for an application during installation process.

The figures 11, 12 and 13 shows the screenshot of device policy, apps policy and user apps tabs, respectively. The figures 11, and 12 clearly demonstrate the simplicity in the policy administration, please note that these two figures do not show all the available policies. Upon the selection of "Configure Users" from the menu, the list of available users is displayed and the primary user can configure rules for the secondary device users. There are also provisions for whitelisting applications and networks in the menu. Using whitelist feature, a user, for instance, can associate device to a Wi-Fi network or install an application from any source without any restrictions.

To configure the status of policies for individual application, the specific application has to be selected from the User apps tab (see Figure 13). Upon

the selection of an application, the enforced policies will be listed and users can change the policy statuses according to their preferences. The restrictions of Firefox application is given in Figure 14

There are possibilities that a user can feel overwhelmed to configure policies per application. To ease this process, Apps policy (tab 4) will serve as a configuration template. During installation, the configured settings in tab 4 are used as a reference to create a policy set for new applications but it does not affect already existing applications. A user has to adjust policies for each application based on his/her needs and maintain a standard configuration in the template for new applications.

6. SYSTEM EVALUATION

CoDRA is implemented using the Android Open Source Project code base that support most of the Nexus variants and it is evaluated using Nexus 5 and 9 devices. During evaluation, several events were observed and some of them are described in subsection 6.4. The evaluation employs different test cases to prove the policy granularity, policy enforcement (system-wide and application-wise), and operational overhead.

6.1. Policy Granularity

One of the main objectives of *CoDRA* is to provide fine granular policies. As discussed in Section 5.1, *CoDRA* achieves finer degree of policy granularity due to the adoption of feature based context. To prove this claim, the Android permission model is compared against *CoDRA*.

The screenshots of Firefox application restriction in *CoDRA* and Android are given in Figure 14 and 15 respectively. Figure 14 shows partial list of the enforced policies for Firefox in *CoDRA*. From the figures, it is clear that *CoDRA* has finer granular policy to control Firefox behavior and also provides precise information on resource access. In the Android permission model, there is no provision to restrict resources based on their attributes and it fails to

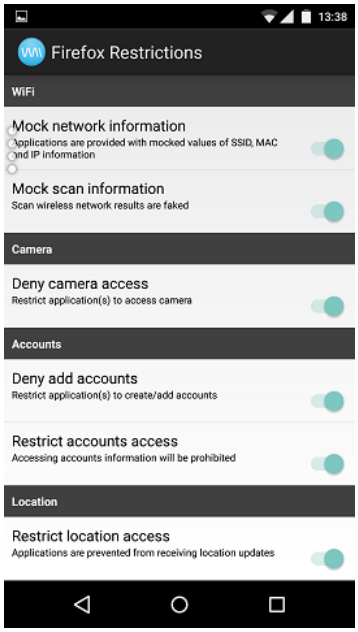

Figure 14: *CoDRA*: Firefox

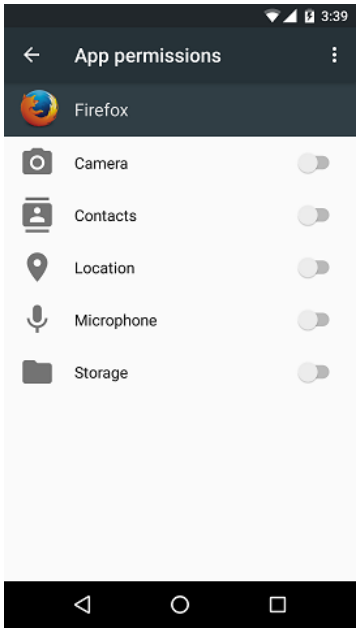

Figure 15: Permission model: Firefox

cover certain resources. Furthermore, there are no means to control device level activities.

6.2. Policy Enforcement

The policies that are created in *CoDRA* have to be evaluated to make sure that they satisfy its objectives. 55 popular applications, where each of these applications has several million downloads, were selected for evaluation. They were monitored closely and executed with full restriction. All of the applications request calls were logged and categorized into appropriate resource groups, such as Wi-Fi, contacts and broadcasts. Calls to the methods, such as GetNetworkOperatorName(subId), GetNetworkOperatorForPhone(phoneId) and GetNetworkCountryIsoForPhone(phoneId) were also recorded and grouped under device information as they reveal details about the user. Under broadcast category, every possible event triggers like package, power, USB and user were captured. It was noticed that none of the applications were crashed due to full restrictions. 41 applications along with the restriction on resource accessibility are given in Table 6 in Appendix. Among the evaluation results, five scenarios are discussed in detail: Wi-Fi association, application installation, device

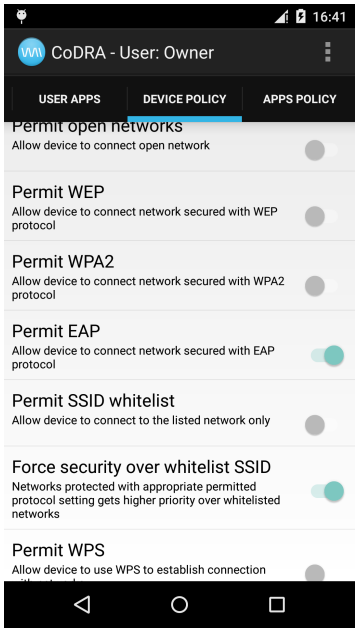

Figure 16: Wi-Fi Policy Tab

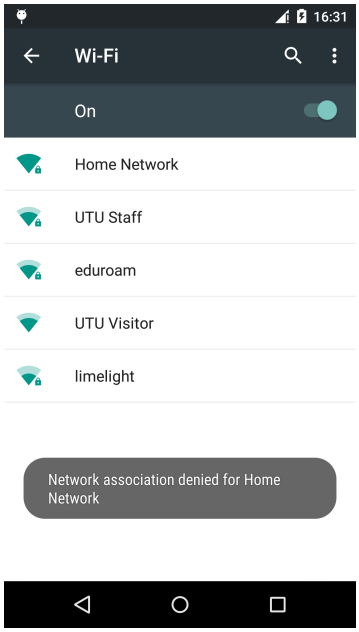

Figure 17: WPA2 Network Denial

fingerprinting, privacy policies, and hardware restrictions.

Wi-Fi Association: Wi-Fi networks exhibits certain features that allows mobile device to establish connection. With the help of control panel, one can force his/her device to connect with the network that has the desired security (open, WEP, WPA2, EAP) or passphrase strength (existence of uppercase, special characters, numbers, or all). The test device, where *CoDRA* is deployed, is configured to connect with the network which employs EAP for security and the configuration is shown in Figure 16. *CoDRA* prevented the device from connecting to a network called "Home Network" secured with WPA2 protocol (see Figure 17) as directed by the policies. *CoDRA* also controls the network association through WPS. If WPS restriction is active, all the WPS authentication means are disabled including APIs. The WPS unavailability in Wi-Fi settings is illustrated in Figure 18.

Application Installation: In Android, application can be installed in different ways and from various sources: Google Play, through IDE/terminal, download apk and on-click install. *CoDRA* controls every possible means of installation and the configuration used for this evaluation is presented in Figure 11. A

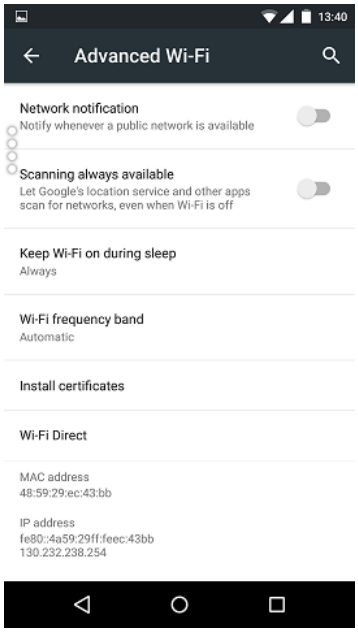

Figure 18: WPS unavailability

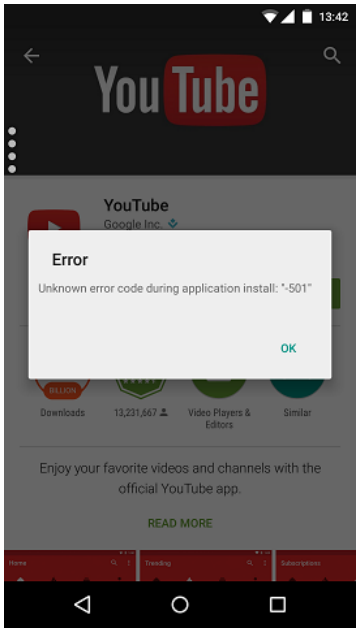

Figure 19: Google Play Install prevention

new error code, -125 (INSTALL_FAILED_INSTALLATION_NOT_ALLOWED), is defined to denote the failure of installation and will return this code if the installation is failed due to *CoDRA* restrictions. When user tries to install an application through IDE, it follows the same procedure as terminal installation in an automated manner as portrayed in Figure 20.

To verify other installation sources, Google Play is included in our customized Android build and the working of *CoDRA* during Google Play installation is given in Figure 19. Google Play returned different error code than -125. Since Google Play is a proprietary application, it is not possible to customize Google Play to understand the *CoDRA*'s error code. The other means of installation did returned the *CoDRA*'s error code (-125).

Device fingerprinting: The policies that protect user data, such as IMEI, Wi-Fi address, Bluetooth address, mobile numbers, service provider and ANDROID_ID along with several unprotected methods were verified under this case. From the logs, it is confirmed that the applications had received spurious information that prevented the application from gaining the device information.

Privacy policies: Any misuse of information like messages, call logs and con-

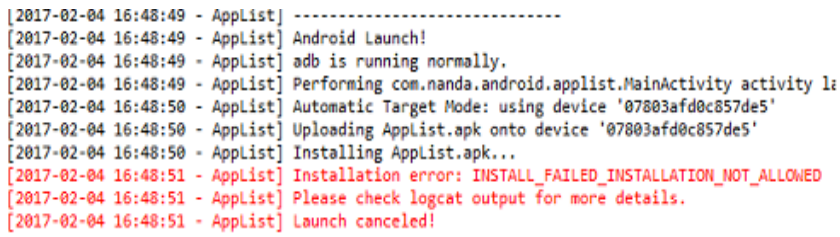

Figure 20: CoDRA prevents application installation through Eclipse IDE

tacts can lead to serious privacy threat. *CoDRA* applies rules on every aspect of privacy related resources. The policy configuration, allowed and restricted access to contacts of Drupe application are given Figure 21, 23 and 23, respectively.

Hardware restrictions: Similar to privacy and device fingerprinting, *CoDRA* enforces rules on access to the hardware resources like sensors and camera. The figures 24, 25 and 26 clearly demonstrates the dynamic policy restriction on camera, sensors and broadcast on AndroSensor and OpenCamera applications.

Permission verification abuse prevention: CoDRA works along with Android permission model to prevent the verification abuses. Instead of manipulating the permission model statuses, *CoDRA* manages its own data structures and depend on it for verification decisions. That is, user can grant requested permission in Android and then use *CoDRA* to enforce restrictions. During the evaluation of test cases, without Android permission certain applications did not work properly but policy enforcement through *CoDRA* worked successfully.

6.3. Operation Overhead

It is essential to measure the overhead and to prove that the customization made into the Android do not incur substantial resource consumption. System.nanoTime() method was used to quantify the additional execution period incurred by the Android. To calculate the overhead, the above method was added before and after the points of resource request calls. The execution time of resource calls related to Wi-Fi, Bluetooth, location, light and gyroscope

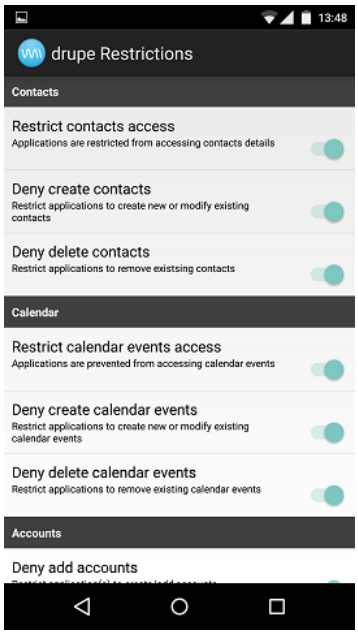

Figure 21: *CoDRA* - Drupe app restriction

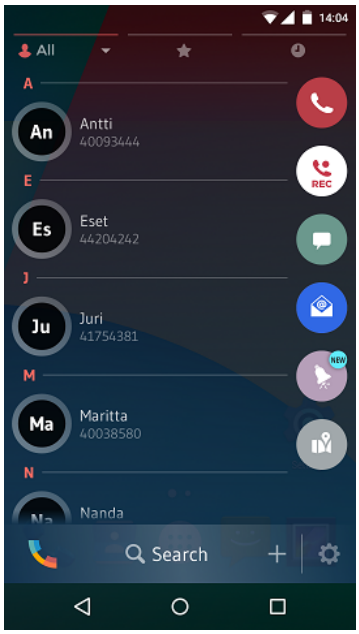

Figure 22: Drupe with contacts access

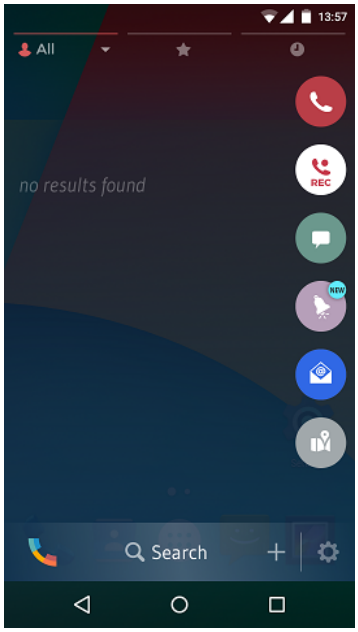

Figure 23: Drupe app without contacts access

Figure 24: AndroSensor app without sensor and broadcast access

Figure 25: OpenCamera app without camera access

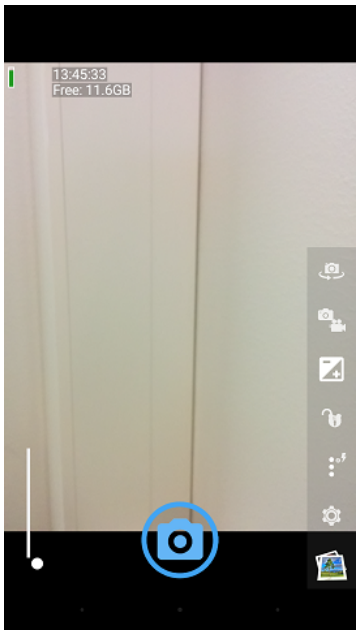

Figure 26: OpenCamera app with camera access

sensors, camera and contacts were evaluated. As a reference for comparison, executions of the same test cases in stock Android were also carried out. The comparison of execution time having an average of 100 invocations is given in Table 3. It is clear from the table that *CoDRA* incurs 1ms to 20ms additional execution time. The execution overhead of *CoDRA* is reasonably low and in real-time usage, the delay is hardly noticeable.

In terms of memory, *CoDRA* occupied around 800kB with initial contents such as application template and device policy. The memory size is directly proportional to the number of users and applications in the devices. It increases the memory by 5 to 9kB per application and an additional 5kB for every new user. The memory requirement of *CoDRA* is insignificant especially compared to mobile devices that have a minimum of 2GB internal storage and thus, proves that the newly developed *CoDRA* is usable in any mobile devices.

Unfortunately, the performance of *CoDRA* can not be compared with FlaskDroid [45], FireDroid [46], FlexDroid [47], Aurasium [48] and CRePE [21] since they

Table 3: Runtime execution overhead (average of 100 invocation)

Resources	**Stock (ms)**	**Policy Enforcement Classification**	
		Restrict (ms)	**Allow (ms)**
LastLocation	10.5635	11.19833	11.2263
Camera	114.8567	126.1546	129.587
Query Contacts	19.1654	19.6989	28.5018
Broadcast Register	2.0396	22.0018	22.1222
Light Sensor	7.1477	19.3518	23.0150
Wi-Fi Info	1.3018	9.5069	11.5942

used earlier versions of Android (2.2 - 4.4). Those version neither support multiuser environment nor runtime permission revocation. Since the functions of Android and execution time of its permission model differs in different versions, providing comparison on average overhead do not yield fair results.

6.4. Observed Events

During evaluation of *CoDRA*, interesting activities were noticed and few of them are described below.

Applications accessing protected resources without permission *Moments*, developed by Facebook, registered a listener successfully to monitor the phone state without having READ_PHONE_STATE permission. This activity allows *Moments* to obtain information about incoming and outgoing call though it has nothing to do with call activities. *CoDRA* successfully identified the activity and thwarted *Moments* from registering listener.

Application crashed due to logic error in program: Navigator [49] is the only application that crashed during our evaluation of 55 applications. The logs revealed that the application crashes when it calls *getPackageName()* in android.content.Component class which requires names of a package and appropriate class for instantiation. Upon further investigation, it is observed that *Navigator* is calling three navigation applications: *Google Maps, GPS Navigation [50] and GPS-Navigation & Maps Sygic [51]*, and by default, it tries to open

Google Maps. Since *Google Maps* was not installed in the test device at the time of executing the *Navigator*, it had crashed. The developers assumed that every Android powered device contains *Google Maps* and hardcoded to execute it. Therefore, the investigation confirmed that the application had crashed due to hardcoded logic and it was not related in anyway to the *CoDRA*'s restrictions.

Device fingerprinting & broadcast: Every tested application employs all available means to obtain details about the device and user including the applications that do not have appropriate permissions. In addition, they also gather information like frequency of device usage, installed or uninstalled applications and the active/inactive period by registering appropriate broadcasts as they do not require any permissions.

Phone State listeners: It is also noticed that gaming applications such as *Lep's World 2*, *Dead Trigger 2* and *Subway Surfer*, and *Stick Notes* [52] application were listening to phone status.

Difficulty in using certain applications With full restrictions, some applications failed to work properly as they were unable to access a resource that is necessary for its operation. For example, a gaming application, *Asphalt 8: Airborne*, requires sensors to change the direction of cars while playing the game and *Twitter* requires login every time as it was denied access to accounts.

7. Related Work

There are several research efforts to improve the security of Android [8], [9], [13], [14], [16], [17], [21], [48], [53], [22], and [54]. Since *CoDRA* is a context based access control system, most of the chosen related works are the ones which has some form of context awareness.

An access control system having context-related policy for Android (CRePE) is presented in [21]. It is based on environment context and relies on GPS, time and Wi-Fi information to carry out its operations. Using CRePE, one can allow/deny certain actions based on location or the nature of the operation. The policies are written in XML language having predefined commands

and keywords. The policy administration can be achieved locally using local administrator component and remotely through SMS, Bluetooth or QR-codes. CRePE relies on Android defined permissions for enforcing rules but there are resources that are not covered by Android permission. It follows only static policy enforcement procedure, and policy need to be written manually. The policy status modification can be done through the provided interfaces at runtime. The control of device activities is unclear (for instance, network association and application installations) and the degree of action in policy refinement is limited. In addition, the usability of local administrator component (we assumed that it is a system application) is not clear.

A Framework for context-aware access control system for mobile devices based on context classification (ConXsense) has been proposed in [22]. The framework uses probabilistic approach for context sensing and automatic classification of appropriate context through machine learning. It is implemented on top of FlaskDroid architecture [45] for fine-grained access control system. MOSES framework is a policy based framework for enforcing isolation of application and data depending on the context information [9]. MOSES framework employs the combination of TaintDroid [17] architecture and application activities restriction at Android middleware layer to create security profiles with the aim of achieving isolation and dynamic switching between the profiles. A security framework, TrustDroid, is presented to provide lightweight domain isolation using policies with limited context support (GPS and Wi-Fi)[8]. It achieves isolation by restricting inter-domain application communication at middleware layer and TOMOYO Linux as mandatory access control at kernel layer. In [14], the authors presented DeepDroid, dynamic security enforcement mechanism using location context for enterprise usage. It is implemented by placing monitors on certain critical process dynamically and applies restrictions on applications at runtime without making any changes in the platform. Root privilege is mandatory for DeepDroid operations. These works have limited context and policy granularity. Except DeepDroid, they all lack dynamic enforcement.

There are works where the enforcement is realized through security appli-

cations development. ASF [54] and ASM [13] belongs to this category. They both provide interface to implement security modules but ASF differ from ASM in two aspects: inline reference monitors and guaranteeing Androids existing security provisions. The security features of both frameworks provide APIs to implement resource access control and returning fake data. These works has two limitations. First, it does not implement any control by default even if it is available within the OS; applications have to be developed and installed in mobile devices to realize the resource restrictions. Second, individual users who do not have programming knowledge have to rely on the applications developed by third party sources in order to fulfill their security objectives and this may expose devices to privacy and security threats.

Aurasium [48] and Dr. Android and Mr. Hide [53] approaches rely on repacking the applications with reference monitors. The application repackaging process breaks the application signature. The management of signatures for each application incurs unnecessary operational overhead. The users can install applications from other sources without monitors from other sources may leads to abnormal application behavior. In addition, the security can be compromised if they do not monitor all APIs. Both of them fail to support restrictions based on context information.

Android has introduced several improvements in version 6.0 and one of them is runtime permission management that empowers users to restrict the application activity at later stages [55]. Though it has advantages, it covers only certain permissions that are classified as dangerous, the rest of the permissions can not be revoked, for instance, ACCESS_WIFI_STATE as it is categorized as normal. In [56], the authors concluded that 70% of the applications have failed to handle the permission revocation properly by using appropriate methods. Application developers may use verification methods (see 2.1.3) in order to gain access to unrelated resources. Another significant change in permission model is permission groups that allow applications to obtain additional permissions from the same group [57], though they are irrelevant for their operation. The permission groups are not formed well and lacks refinement. For instance, hav-

ing permission to read call logs makes the application to inherit all call based permissions. Finally, there is no permission to regulate network association on user basis and application installation based on origin.

Unlike *CoDRA*, most of the related works 1) have coarse granularity in policy, 2) lack dynamic enforcement, 3) fail to regulate device level activities, and 4) limited restriction for open resources access. The context based related works define mobile device or application as an entity. Whereas *CoDRA* relies on the activities and attributes of each resource in the device and defines resource as an entity rather than the whole mobile device. They target only application activities either to deny or mock after application installation. In our work, we took a step further and targets application prior to the installation based on the features such as mode of installation, application source and package name. In addition, they all carry out restriction at resource level while *CoDRA* enforce policies at resource feature level. In network restriction, existing researches have emphasized more on regulating internet. But, they have failed to address the security problems in the context of Wi-Fi networks as it becomes the entry point to the Internet in most devices. Some of the works failed to incorporate multiuser feature though multiuser support has been available in Android during their research period. For example, in [58], the authors proposed modification to Android architecture based on usage control model for application isolation, though it is possible to achieve application isolation by proper utilization of multiuser environment. In *CoDRA*, it is easily possible to have application isolation through separate user accounts.

8. Conclusion

A context-based dynamically reconfigurable access control system for Android has been presented. The novelty of *CoDRA* is its adoption of feature based context in policy design, resulting in better control on resource accessibility and policy granularity. The policy enforcement was enacted based on a combination of application behavior and resource features. Unlike most of

the previously proposed access control systems, which apply only static enforcements on application activities, *CoDRA* employs both static and dynamic restrictions. The enhanced permission revocation scheme of *CoDRA* made the policy changes remain unknown to applications. *CoDRA* is integrated into the Android operating system and capitalizes on multiuser feature for the enforcement of different policy sets on device individual accounts. An easily accessible graphical control panel is provided for dynamic policy configuration. The control panel enables easy policy administration, as it provides a switch interface to determine the policy statuses.

CoDRA was developed using the Android Open Source Project code base that supports a wide range of nexus variants devices. The developed system was deployed in Nexus 5 and Nexus 9, and both qualitative and quantitative analyses were performed by examining the execution of 55 popular applications. The test cases were modeled with the aim of verifying the achievement of the defined objectives. Evaluation results confirmed that *CoDRA* successfully imposed static and dynamic restrictions on application and system activities per device's user account. The outcome of the test cases clearly demonstrated that the adopted context for policy design assured finer granularity in policies. It has also been confirmed that permission verification abuse and forced user consent threats are no longer realizable, even with the usage of appropriate verification settings. This is due to the successful concealment of *CoDRA*'s policy statuses and the preservation of existing security execution flow. *CoDRA* occupied about 800kB memory for policy storage and 5kB of memory for every additional user context space. It also incurred additional 1-20ms execution time overhead for policy verification and enforcement of appropriate actions. Therefore, *CoDRA* is a very promising lightweight security solution for Android powered mobile devices with multiuser support.

The study on applications behavior, usability and evaluation of *CoDRA*'s ability to restrict malware activities are left for future work. In addition, the core operations of *CoDRA* will be extended to control internet activities and to fulfill BYOD requirements, as it already supports multiuser environment.

References

[1] Appbrain, Number of available android applications (2016).
URL http://www.appbrain.com/stats/number-of-android-appsl

[2] Statista, App stores: number of apps in leading app stores 2016 (2016).
URL https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores

[3] Gartner, Gartner says by 2018, more than 50 percent of users will use a tablet or smartphone first for all online activities (2014).
URL http://www.gartner.com/newsroom/id/2939217

[4] Android, Android dashboard (2016).
URL https://developer.android.com/about/dashboards/index.html

[5] Android, The android source code (2016).
URL https://source.android.com/source/index.html

[6] Y. Wang, Y. Chen, F. Ye, J. Yang, H. Liu, Towards understanding the advertiser's perspective of smartphone user privacy, in: Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on, IEEE, 2015, pp. 288–297.

[7] Android, Android developers-mainfest.permission (2016).
URL http://developer.android.com/reference/android/Manifest.permission.html

[8] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.-R. Sadeghi, 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.

[9] G. Russello, M. Conti, B. Crispo, E. Fernandes, Y. Zhauniarovich, Demonstrating the effectiveness of moses for separation of execution modes, in:

Proceedings of the 2012 ACM conference on Computer and communications security, ACM, 2012, pp. 998–1000.

[10] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, D. Wagner, Android permissions: User attention, comprehension, and behavior, in: Proceedings of the Eighth Symposium on Usable Privacy and Security, ACM, 2012, p. 3.

[11] J. Crussell, R. Stevens, H. Chen, Madfraud: Investigating ad fraud in android applications, in: Proceedings of the 12th annual international conference on Mobile systems, applications, and services, ACM, 2014, pp. 123–134.

[12] M. Mohamed, B. Shrestha, N. Saxena, Smashed: Sniffing and manipulating android sensor data, in: Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, ACM, 2016, pp. 152–159.

[13] S. Heuser, A. Nadkarni, W. Enck, A.-R. Sadeghi, Asm: A programmable interface for extending android security, in: Proc. 23rd USENIX Security Symposium (SEC14), 2014.

[14] X. Wang, K. Sun, Y. Wang, J. Jing, Deepdroid: Dynamically enforcing enterprise policy on android devices, in: Proc. 22nd Annual Network and Distributed System Security Symposium (NDSS15). The Internet Society, 2015.

[15] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, P. McDaniel, Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps, in: ACM SIGPLAN Notices, Vol. 49, ACM, 2014, pp. 259–269.

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

[17] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, A. N. Sheth, Taintdroid: an information-flow tracking system

for realtime privacy monitoring on smartphones, ACM Transactions on Computer Systems (TOCS) 32 (2) (2014) 5.

[18] K. Fawaz, K. G. Shin, Location privacy protection for smartphone users, in: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2014, pp. 239–250.

[19] F. Wei, S. Roy, X. Ou, et al., Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps, in: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2014, pp. 1329–1341.

[20] S. Smalley, R. Craig, Security enhanced (se) android: Bringing flexible mac to android., in: NDSS, Vol. 310, 2013, pp. 20–38.

[21] M. Conti, B. Crispo, E. Fernandes, Y. Zhauniarovich, Crêpe: A system for enforcing fine-grained context-related policies on android, IEEE Transactions on Information Forensics and Security 7 (5) (2012) 1426–1438.

[22] M. Miettinen, S. Heuser, W. Kronz, A.-R. Sadeghi, N. Asokan, Conxsense: automated context classification for context-aware access control, in: Proceedings of the 9th ACM symposium on Information, computer and communications security, ACM, 2014, pp. 293–304.

[23] J. Han, E. Owusu, L. T. Nguyen, A. Perrig, J. Zhang, Accomplice: Location inference using accelerometers on smartphones, in: 2012 Fourth International Conference on Communication Systems and Networks (COMSNETS 2012), IEEE, 2012, pp. 1–9.

[24] S. Narain, T. D. Vo-Huu, K. Block, G. Noubir, Inferring user routes and locations using zero-permission mobile sensors, in: 2016 IEEE Symposium on Security and Privacy (SP), IEEE, 2016, pp. 397–413.

[25] S. Nawaz, C. Mascolo, Mining users' significant driving routes with low-power sensors, in: Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems, ACM, 2014, pp. 236–250.

[26] X. Zhou, S. Demetriou, D. He, M. Naveed, X. Pan, X. Wang, C. A. Gunter, K. Nahrstedt, Identity, location, disease and more: Inferring your secrets from android public resources, in: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, ACM, 2013, pp. 1017–1028.

[27] A. Wang, G. Chen, J. Yang, S. Zhao, C.-Y. Chang, A comparative study on human activity recognition using inertial sensors in a smartphone, IEEE Sensors Journal 16 (11) (2016) 4566–4578.

[28] J. R. Kwapisz, G. M. Weiss, S. A. Moore, Activity recognition using cell phone accelerometers, ACM SigKDD Explorations Newsletter 12 (2) (2011) 74–82.

[29] S.-W. Lee, K. Mase, Activity and location recognition using wearable sensors, IEEE pervasive computing 1 (3) (2002) 24–32.

[30] B. P. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, I. Molloy, Android permissions: a perspective combining risks and benefits, in: Proceedings of the 17th ACM symposium on Access Control Models and Technologies, ACM, 2012, pp. 13–22.

[31] Y. Zhou, X. Jiang, Dissecting android malware: Characterization and evolution, in: Security and Privacy (SP), 2012 IEEE Symposium on, IEEE, 2012, pp. 95–109.

[32] Android, Requesting permissions — android developers (2017).
URL `https://developer.android.com/guide/topics/permissions/requesting.html`

[33] Android, permission— android developers (2017).
URL `https://developer.android.com/guide/topics/manifest/permission-element.html`

[34] Android, Android developers-mainfest.permission (2016).
URL https://developer.android.com/reference/android/Manifest.permission.html#ACCESS_WIFI_STATE

[35] G. Developers, The google maps geolocation api (2016).
URL https://developers.google.com/maps/documentation/geolocation/intro

[36] G. Developers, Google i/o 2015 - android m permissions (2017).
URL https://www.youtube.com/watch?v=f17qe9vZ8RM

[37] A. Documentation, Working with system permissions (2017).
URL https://developer.android.com/training/permissions/index.html

[38] F. Chiti, R. Fantacci, M. Loreti, R. Pugliese, Context-aware wireless mobile autonomic computing and communications: research trends and emerging applications, IEEE Wireless Communications 23 (2) (2016) 86–92.

[39] J. Wan, D. Zhang, S. Zhao, L. T. Yang, J. Lloret, Context-aware vehicular cyber-physical systems with cloud support: architecture, challenges, and solutions, IEEE Communications Magazine 52 (8) (2014) 106–113.

[40] X. Jiang, J. A. Landay, Modeling privacy control in context-aware systems, IEEE Pervasive computing 1 (3) (2002) 59–63.

[41] V. Arena, V. Catania, G. La Torre, S. Monteleone, F. Ricciato, Securedroid: An android security framework extension for context-aware policy enforcement, in: Privacy and Security in Mobile Systems (PRISMS), 2013 International Conference on, IEEE, 2013, pp. 1–8.

[42] A. Ahmed, N. Zhang, A context-risk-aware access control model for ubiquitous environments, in: Computer Science and Information Technology, 2008. IMCSIT 2008. International Multiconference on, IEEE, 2008, pp. 775–782.

[43] A. Corrad, R. Montanari, D. Tibaldi, Context-based access control management in ubiquitous environments, in: Network Computing and Applications, 2004.(NCA 2004). Proceedings. Third IEEE International Symposium on, IEEE, 2004, pp. 253–260.

[44] M. J. Covington, M. J. Moyer, M. Ahamad, Generalized role-based access control for securing future applications.

[45] S. Bugiel, S. Heuser, A.-R. Sadeghi, Flexible and fine-grained mandatory access control on android for diverse security and privacy policies., in: Usenix security, 2013, pp. 131–146.

[46] G. Russello, A. B. Jimenez, H. Naderi, W. van der Mark, Firedroid: hardening security in almost-stock android, in: Proceedings of the 29th Annual Computer Security Applications Conference, ACM, 2013, pp. 319–328.

[47] J. Seo, D. Kim, D. Cho, T. Kim, I. Shin, Flexdroid: Enforcing in-app privilege separation in android.

[48] R. Xu, H. Saïdi, R. Anderson, Aurasium: Practical policy enforcement for android applications, in: Presented as part of the 21st USENIX Security Symposium (USENIX Security 12), 2012, pp. 539–552.

[49] Navigation, Navigator (version 1.28)) (2017).
URL `Available:https://play.google.com/store/apps/details?id=com.navigation.navigator`

[50] B. S.R.O, Gps navigation (version 16.2.5) (2017).
URL `Available:https://play.google.com/store/apps/details?id=cz.aponia.bor3`

[51] S. A.S, Gps-navigation & maps sygic (version 16.4.10) (2017).
URL `Available:https://play.google.com/store/apps/details?id=com.sygic.aura`

[52] D. LLP, Sticky notes (version 15.5)) (2017).
URL Available:https://play.google.com/store/apps/details?id=com.gs.stickit

[53] J. Jeon, K. K. Micinski, J. A. Vaughan, A. Fogel, N. Reddy, J. S. Foster, T. Millstein, Dr. android and mr. hide: fine-grained permissions in android applications, in: Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices, ACM, 2012, pp. 3–14.

[54] M. Backes, S. Bugiel, S. Gerling, P. von Styp-Rekowsky, Android security framework: Extensible multi-layered access control on android, in: Proceedings of the 30th annual computer security applications conference, ACM, 2014, pp. 46–55.

[55] Android, Security enhancements in android 6.0 (2016).
URL https://source.android.com/security/enhancements/enhancements60.html

[56] Z. Fang, W. Han, D. Li, Z. Guo, D. Guo, X. S. Wang, Z. Qian, H. Chen, revdroid: Code analysis of the side effects after dynamic permission revocation of android apps, in: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, ACM, 2016, pp. 747–758.

[57] Android, System permissions in android 6.0 (2016).
URL https://developer.android.com/guide/topics/security/permissions.html

[58] F. Martinelli, P. Mori, A. Saracino, Enhancing android permission through usage control: a byod use-case, in: Proceedings of the 31st Annual ACM Symposium on Applied Computing, ACM, 2016, pp. 2049–2056.

9. Appendix

Table 4: Evaluation of CoDRA and performance of applications - I)

Downloads (millions)	**Application**	**Wi-Fi**	**Location**	**Call Log**	**Contacts**	**Messages**	**Sensors (Low-Power)**	**Boot**	**Camera**	**Device Info.**	**Broadcast**	**Accounts**	**Phone State**
50	LinkedIn		❍		❍					❍	❍	❍	
1000	Instagram	❍	❍		❍	●	❍	❍	❍	❍	❍	❍	❍
1000	Facebook	❍	❍		❍	❍		❍	❍	❍	❍	❍	❍
1000	WhatsApp	❍	❍	❍	❍	❍		❍	❍	❍	❍	❍	❍
100	Snapchat	❍	❍		❍	●	❍	❍	❍	❍	❍	❍	❍
500	Viber	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
50	Tumblr	❍			❍		❍		❍		❍	❍	
500	Twitter	❍	❍		❍	●		❍	❍	❍	❍	◆	❍
100	Pinterest	❍	❍		❍				❍		❍	❍	
1000	Messenger(FB)	❍	❍	❍	❍	❍		❍	❍	❍	❍	❍	❍
500	Skype	❍	❍		❍	❍		❍	❍	❍	❍	❍	❍
100	Telegram	❍	❍		❍	●		❍	❍	❍	❍	❍	❍
100	Spotify	❍								❍	❍	❍	❍
100	TuneIn Radio		❍					❍		❍	❍	❍	
100	Netflix	❍								❍	❍		❍
50	Moments(FB)	❍	❍				❍	❍	❍	❍	❍	❍	❑
500	Dropbox	❍			❍			❍	❍	❍	❍	❍	

❍	Access permission	Apps operated normally under full restriction
●	Access through broadcast	
◆	Require login for every run	
❑	Access withouth permission	
▲	Never execute without resource	Apps never executed if resource restricted

Table 5: Evaluation of CoDRA and performance of applications - II)

Downloads (millions)	**Application**	**Wi-Fi**	**Location**	**Call Log**	**Contacts**	**Messages**	**Sensors (Low-Power)**	**Boot**	**Camera**	**Device Info.**	**Broadcast**	**Accounts**	**Phone State**
10	WiFi Analyser	❍									❍		
500	Subway Surfer	❍								❍	❍	❍	❍
100	Temple Run 2	❍								❍	❍		❍
100	Ninja Fruit	❍								❍	❍	❍	❍
10	War Robots	❍								❍	❍	❍	
5	Open Camera		❍				❍		❍		❍		
100	Candy Camera	❍	❍				❍	❍	❍	❍	❍	❍	❍
10	Camera for Android	❍	❍		❍		❍	❍	❍	❍	❍	❍	
1	AndroSensor	❍	❍				❍			❍	❍		❍
1000	Google Maps	❍	❍		❍		❍	❍		❍	❍	❍	
10	Here We Go	❍	❍	❍	❍	❍	❍			❍	❍	❍	❍
500	Google Street View	❍	❍				❍		❍		❍	❍	
5	Navigator		❍				❍			❍	❍		❍
10	Compass		❍				❍			❍	❍		
1000	Google Hang-outs	❍	❍		❍	●	❍	❍	▲	❍	❍	▲	❍
5	Drupe	❍	❍		❍	❍		❍		❍	❍	❍	❍
100	True Caller	❍	❍		❍	●		❍		❍	❍	❍	❍
100	Asphalt 8:Air-borne	❍	❍				❍	❍		❍	❍	❍	

Table 6: Evaluation of CoDRA and performance of applications - III)

Downloads (millions)	**Application**	**Wi-Fi**	**Location**	**Call Log**	**Contacts**	**Messages**	**Sensors (Low-Power)**	**Boot**	**Camera**	**Device Info.**	**Broadcast**	**Accounts**	**Phone State**
100	Shadow Fight 2	❍					❍			❍	❍	❍	
10	Dead Trigger 2	❍								❍	❍		❍
1000	Chrome	❍	❍					❍	❍	❍	❍	❍	
100	Firefox	❍	❍				❍	❍	❍	❍	❍	❍	
100	Opera	❍	❍				❍	❍	❍	❍	❍		❍
100	UCBrowser	❍	❍				❍	❍	❍	❍	❍	❍	❍
10	Math Tricks							❍		❍	❍		
10	Camera (Google)	❍	❍		❍		❍	❍	❍		❍	❍	
50	Musical.ly	❍			❍			❍	❍	❍	❍	❍	❍
1000	Youtube	❍					❍	❍	❍	❍	❍	❍	
100	MS Excel	❍			❍			❍		❍	❍	▲	
100	Imo	❍	❍	❍	❍	❍		❍	❍	❍	❍	❍	❍
1000	Gmail				❍			❍			❍	❍	
50	QR Code Reader	❍	❍						❍		❍		
10	QR & Barcode Scanner	❍							❍	❍	❍		
50	Lep's World	❍								❍	❍		❍