Advances In Multidisciplinary & Scientific Research
A Multidisciplinary & Interdisciplinary Journal

Vol. 2, No. 2, June, 2016

An Enhanced Access Control Framework for External Memory Storage of Android Devices

A.B. Adeyemo (PhD)
Department of Computer Science
University of Ibadan,
Ibadan, Nigeria
ab.adeyemo@mail.ui.edu.ng

Afe O. F., Ibitola A.G. & Oyekunle V.B
Department of Computer Science,
Lead City University,
Ibadan, Nigeria
afeseyi@yahoo.com, ibitolaayobami@yahoo.com& bolanleoyekunle@yahoo.com

ABSTRACT

The enhanced access control framework (eACF) was designed such that upon installing an application, the user is made aware of the permissions that the application requests through a pop-up dialog box which prompts the user by asking to what level the application can be granted access. The framework, through user interaction, can therefore allow users to provide access control and hence improve security of their device, rather than automating the decisions through all-or-nothing permission approach. This was implemented by integrating the framework with the operating system architecture such that a user can give permissions to applications immediately after installation or at application runtime. The experimental results indicate that at application runtime, the user is prompted, via pop-up box, on an attempt by an application to access a particular resource category in the external storage card of the device, thereby enabling the user to control its access. The performance of the framework was evaluated to 64.71%. From this paper, it can be concluded that the framework improves significantly over the existing access control mechanism by being user-centric. Notably, the framework also gives good performance on access restriction to the use of sensitive resources stored in the external storage by applications and keeping of the log details to monitor resources usage.

Keywords: Access, Control, Framework, External Memory, Storage & Android Devices.

Aims Research Journal Reference Format:
A.B. Adeyemo, Afe O. F., Ibitola A.G. & Oyekunle V.B. (2016): An Enhanced Access Control Framework for External Memory Storage of Android Devices. Advances in Multidisciplinary Research Journal. Vol 2, No.2 Pp 89-110

1. INTRODUCTION

Mobile devices such as notebooks, personal digital assistants, smartphones and USB storage drives have become ubiquitous, and for an increasing number of employees, their jobs would be challenging without the mobility provided by these devices. According to Liebergeld and Lange (2013), people use their smartphones, aside from calling and texting, for connecting with their digital life – email, social networking, instant messaging, photo sharing; storing valuable personal information such as login credentials, photos, emails, contact information, and more. Research shows that about 70% of all smartphone-owning professionals are now using their personal devices to access corporate data, yet almost 80% of that activity remains inadequately managed by Internet Technology departments. The potential benefits of mobile devices can include improved productivity and reduced costs. As defined by Deepak and Pradeep, 2012, the term "Mobile Computing" is used to describe the use of computing devices, which usually interacts, in some fashion, with a central information system while away from the normal, fixed workplace.

Mobile computing technology enables the mobile worker to create, access, process, store and communicate information without being constrained to a single location. This could include working from home or on the road, at an airport or hotel. For the purpose of this research, we have limited our scope of mobile devices to Android smartphones. The decision was based, in part, on the current market demand, the widespread use of this type of device, and the observed security/privacy leakage in the Android mobile platform. The consumer-centric nature of the device was also a major factor for choosing smartphones, that is, they are widely used by consumers and are easily being introduced into corporate environments.

The smartphone market is growing exponentially, but there is one Operating System (OS) in particular that has been enjoying particularly rapid development recently; Google's Android OS. Android is a young platform amongst the various mobile platforms, and the development is very rapid, in that, new major releases come out every few months. The platform was created by Android Inc. which was bought by Google and released as the Android Open Source Project (AOSP) in 2007. A group of 78 different companies formed the Open Handset Alliance (OHA) that is dedicated to develop and distribute Android. The open nature of Android and its large user base have made it an attractive and profitable platform to attack. As the user base has grown so too has the risks associated with the OS. The reason for the rise of Android is connected to the fact that consumers are able to choose from a broad range of manufacturers and price levels.

Mobile devices nowadays store a plethora of sensitive information about users – both private and business-related. Usually, this information can be accessed in predefined locations, such as address books or photo folders, and is thus easily locatable by an attacker. Most of these locations, however, lack comprehensive access control and protection mechanisms, creating low barriers of entry for application developers which then increases the security risk for end users. Risks of unauthorized use, disclosure, modification or destruction of information and information resources are increased with mobile computing. The fact that Google does not make hardware and therefore does not control the eco-system increases the vulnerability of Android mobile devices. There are numerous manufacturers out there making smartphones and tablets which run the Android operating system because it is an open source, and regarding applications (Apps), the fact is that one can get an application from anywhere and install it on one's phone without any restrictions from Google.

1.1 Statement of the Problem

Smartphones have become a part of people's way of life and many information about the users are being stored on them. Applications that provide sensitive services such as banking, SMS, or location services may save their data on the phone (especially the external storage card); and usually, this information can be accessed in predefined locations, and is thus easily locatable by an attacker. Most of these locations, however, lack comprehensive access control and protection mechanisms, creating low barriers of entry for application developers which then increases the security risk for end users. It is therefore imperative to prevent applications from directly accessing resources in the data storage areas of the Android Smartphone, and also to ensure that access to sensitive files can be controlled and /or protected to hinder malicious applications from directly accessing these resources, especially when there is shared privilege permission. Therefore, since sensitive user's data are saved on storage card, is there a security mechanism that protects the data? If all applications can access storage areas, can access control be applied over the storage area?

Research Aim and Objectives

The aim of this research work is to develop an access control framework that improves the security of sensitive information stored in the memory storage of an Android mobile Operating System.

Objectives

i. To analyse the vulnerabilities within Android mobile OS.
ii. To design a system that improves the security of information stored in a memory storage of an Android smartphone through access control.
iii. To implement the system and prevent resource abuse by ensuring that permissions for requested access are granted only to allowed applications.

2. LITERATURE REVIEW

2.1 Information Security in Mobile Computing

With the widespread use of smartphones both in private and work related areas, securing these devices has become of paramount importance. Owners use their smartphones to perform tasks ranging from everyday communication with friends and family to the management of banking accounts and accessing sensitive work related data. These factors, combined with limitations in administrative device control through owners and security critical applications, make Android-based smartphones a very attractive target for attackers and malware authors of any kind and motivation (Fedler, 2012).

2.2 Mobile Risks

Platform vendors have taken measures to keep malicious code off their devices, but malware developers find ways around these controls, and devices are becoming even more enticing targets. The current trend for malware development has been to follow and target the most widely used operating system or platform.
The various mobile risks include:

- Lost or stolen devices
- Mobile malware
- Advanced threats
- Software vulnerabilities
- End-user behaviour

2.3 Threats to Mobile Computing

In July 2012, the Cloud Security Alliance and the Mobile Working Group surveyed 210 security practitioners from 26 countries. Respondents were approximately 80% "experts in the field of information security," which includes security administrators, and consultants, who all ranked mobile top threats. After considering over 40 different top threats to the mobile landscape, the top eight threats include:

1. Data Loss from lost, stolen, or decommissioned devices
2. Information stealing mobile malware
3. Data Loss and data leakage through poorly written third-party applications
4. Vulnerabilities within devices, OS, design, and third-party applications
5. Unsecured Wi-Fi, network access, and rogue access points
6. Unsecured or rogue marketplaces
7. Insufficient management tools, capabilities, and access to APIs
8. NFC and proximity-based hacking

2.4 The Mobile Platforms & their Security Issues

There are various mobile platforms in use today which include: Google Android, Apple iOS, RIM's Blackberry, and Microsoft Windows Mobile. The level of adoption of these mobile platforms into the Marketplace varies, as well as their vulnerabilities.
The latest mobile platforms were designed with security in mind with an attempt to build security features directly into the operating system to limit attacks from the outset.
While mobile platforms are typically built with a number of safeguards, including the ability to isolate applications from critical system processes via a feature called sandboxing, some cybercriminals have found ways to bypass such restrictions using application vulnerabilities.

2.5 Apple iOS

The designers of iOS and Android based their security implementations, to varying degrees, upon five distinct pillars which are: traditional access control, application provenance, encryption, isolation, and permission-based access control. According to Schwartz (2011), Apple iOS offers full protection against malware attacks, fully vets application provenance, offers good encryption and access-control capabilities, but is only moderately good at isolating applications, enforcing permission-based access control, and preventing resource abuse. Apple iOS provides built-in security from the moment the device is turned on. Low level hardware and firmware features are designed to protect against malware and viruses, while high-level OS features help to secure access to personal information and corporate data. Also, it really controls the overall ecosystem.

Though iOS leverages on all the five security pillars mentioned above, its security model is primarily based on four of them, and the secondary reliance is on permission-based access control. The first four security pillars are the primary pillars while the fifth is the secondary security pillar (Nachenberg, 2011):

i. *Traditional access control*: iOS's traditional access control options include password configuration option and account lookout option.
ii. *Application provenance*: Apple employs verification of provenance and authenticity with their iOS developer and iOS developer enterprise programs. The software developers must go through a registration process and pay an annual licensing fee before software can be released to users. Then they must digitally sign each application with an Apple-issued digital certificate before its release. This is done for 2 reasons: first, to guarantee that the application author is an Apple-approved developer; second, to ensure that the application's logic cannot be tampered with after its creation by the author.
iii. *Encryption*: iOS's encryption system provides strong protection of emails and email attachments, and enables device wipe, but thus far has provided less protection against a physical device compromise by a determined attacker. It protects data by encrypting information in three separate areas: in transmission, at rest on the device, and when backed up to iTunes.
iv. *Isolation*: iOS's isolation model totally prevents traditional types of computer viruses and worms, and limits the data that spyware can access. It also limits most network-based attacks, such as buffer overflows, from taking control of the device. However, it does not necessarily prevent all classes of data loss attacks, resource abuse attacks, or data integrity attacks.
v. *Permission-based access control*: iOS's permission model ensures that applications cannot obtain the device's location, send SMS messages, or initiate phone calls without the owner's permission.

3. ANDROID SYSTEM ARCHITECTURE

Running applications is a major goal of operating systems and Android provides several means on different layers to compose, execute and manage applications. Android is a marriage of the Linux operating system and Java-based platform called Dalvik, which is an offshoot of the popular Java platform (Nachenberg, 2011). Zygote is a prototype process that gets forked into a new process whenever a (Java) application is launched. Each application is executed within a Dalvik Virtual Machine (DVM) running under a unique UNIX user and group ID which makes it a sandbox (Liebergeld and Lange, 2013). At the kernel level, access control is enforced via the standard UNIX permission mechanism based on the user id (and group id) of the application. As it is in any Unix/Linux-like operating system, basic access control is implemented through a three-class permission model. It distinguishes between the *owner* of a file system resource, the owner's *group* and *others*. For each of these three entities, distinct permissions can be set to read, write or execute. This system provides a means of controlling access to files and resources. For example, only a file's owner may write (alter) a document, while members of the owner's group may read it and others may not even view it at all. At the middleware level (Application framework), each application is sandboxed, i.e., it is running in its own instance of Dalvik virtual machine, and communication and sharing between applications are allowed only through an inter-process communication (IPC) mechanism (Gunadi and Tiu, 2014).

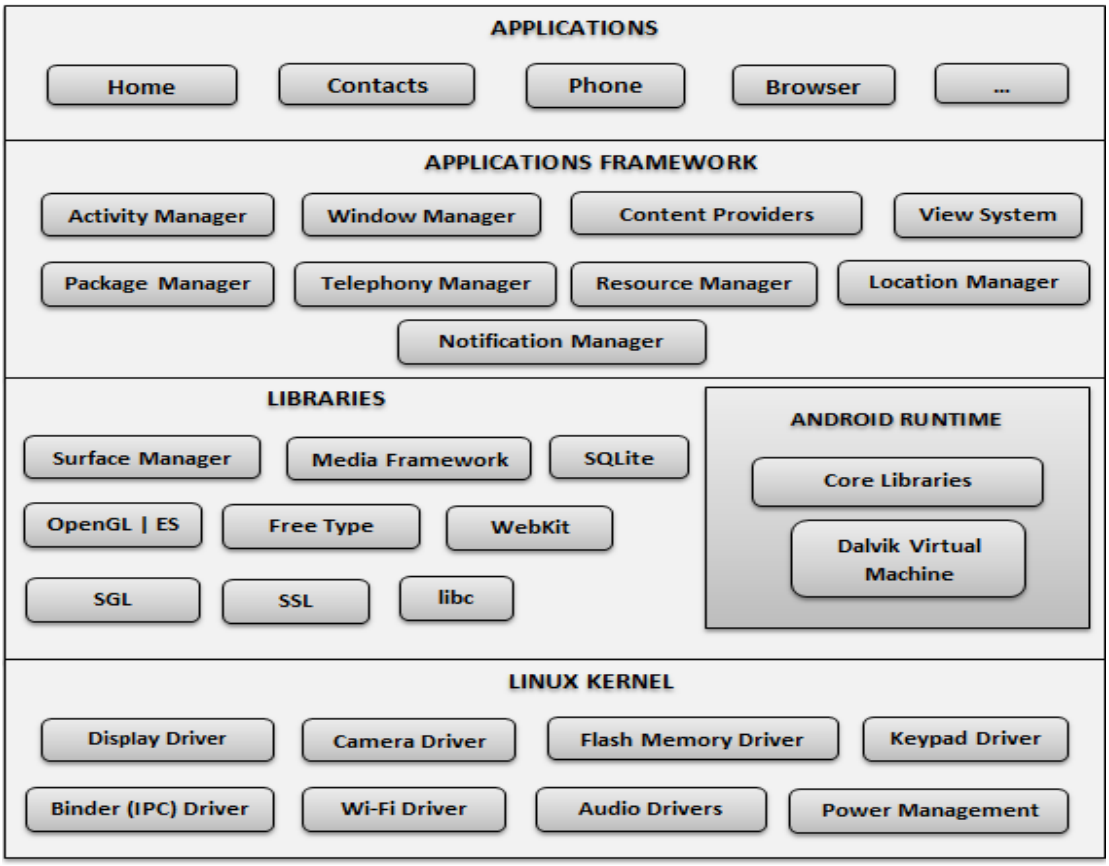

Figure 2.1: Android's System Architecture (Braehler, 2010)

The phone comes pre-installed with a selection of *system applications*, e.g., phone dialer, address book, SMS client app, Web browser, and Contact manager. An application's only interface to the phone is through Application Programming Interfaces (APIs). Application API and the Binder middleware run on top of Linux. *Linux Kernel* The kernel in use is a Linux 2.6 series kernel, modified for special needs in power management, memory management and the runtime environment (Braehler, 2010). It is located at the bottom of the system delegating calls to hardware resources.

3.1 Libraries

This layer is the Android's native libraries. It is the layer that enables the device to handle different types of data. As Android is supposed to run on devices with little main memory and low powered CPUs, the libraries for CPU and GPU intensive tasks are compiled to device optimized native code.

3.2 Android Runtime

Android Runtime consists of Dalvik Virtual machine and Core Java libraries. Android employs Dalvik Virtual Machine (DVM) instead of Java Virtual Machine (JVM) for the reason that DVM is more lightweight and has low memory requirements.

i. Dalvik Virtual Machine: The Dalvik virtual machine (DVM) was developed for resource restricted devices such as mobile phones. It was developed by Dan Bornstein of Google. Unlike JVM, DVM does not run .class files instead, it runs .dex files which are built from .class file at the time of compilation and provides higher efficiency in low resource environments.
ii. Core Java Libraries: These are Java libraries and are available for all devices.

3.3 Application Framework

These are the blocks that the applications directly interact with. These programs manage the basic functions of phone like resource management, voice call management etc. This application framework has robust implementations of common security functionality such as cryptography, permissions, and secure IPC.

The frameworks in this layer are written in Java and provide abstractions of the underlying native libraries and Dalvik capabilities to applications (Braehler, 2010). It is possible for the managers provided by the Application framework for different purposes like power management, system wide notification, resource handling, and window management, to enforce application permissions through the means of the sandbox permission system and ensure that an application is allowed before doing any activity, for example, initiate a phone call, or send data over the network.

3.4 Important blocks of Application framework are:

i. **Activity Manager**: Manages the activity life cycle of applications
ii. **Content Providers**: Manage the data sharing between applications
iii. **Telephony Manager**: Manages all voice calls. We use telephony manager if we want to access voice calls in our application.
iv. **Location Manager**: Location management, using GPS or cell tower
v. **Resource Manager**: Manage the various types of resources we use in our Application

3.5 Components of Android Application

An Android application consists of different parts, called, components. According to its task, an application has one of four basic component types. These are;

i. *Activities* which are the presentation layer of an application, allowing a user to interact with the application. They define a specific user interface (e.g., a dialog window).
ii. *Services* which represent background processes without a user interface.
iii. *Content providers* which are data stores that allow developers to share databases across application boundaries i.e. provide an SQL-like interface for storing data and sharing them between applications, and finally
iv. *Broadcast receivers* which are components that receive and react to broadcast messages, for example, the Android OS itself sends such a broadcast message if the battery is low.

Each component of an application runs as a separate task, making an Android device, to a large extent, a distributed system even if all processes are running on the same device (Berger et al, 2011). Activities, services, and broadcast receivers communicate via asynchronous messages called *intents*. If a recipient of an intent is not instantiated, the OS will create a new instance. An *intent* is an abstract description of an operation to be performed on the platform.

Like all operating systems, Android enables applications to make use of the hardware features, like Global Positioning System (GPS) receivers, cameras, light and orientation sensors, WiFi and UMTS (3G telephony) connectivity and a touchscreen, through abstraction and provide a defined environment for applications (Braehler, 2010).

Security of Android platform has been studied by many researchers. There are three main trends according to Hei, 2013:

i. Static analysis
ii. Security scheme assessment
iii. Malware and virus detection

The major concern of this work is on the security scheme assessment, that is, to study the access control and permission schemes of Android.

3.6 Access Control

Access control is a security aspect whose importance increases with technology advancement as it forms the core of any security system. Looking at access control, there are four basic tasks one might want to carry out: allowing access, denying access, limiting access, and revoking access. Access control can be applied at the Operating System (OS) level, middle-ware level, or the application level. It is responsible for preventing or limiting unauthorized entities from accessing resources, while ensuring access for authorized entities. As discussed earlier, the goals of access control are: 1) Confidentiality i.e. preventing unauthorized disclosure of information 2) Integrity i.e. preventing improper malicious modification 3) Availability i.e. ensuring accessibility of resources to authorized entities.

To provide confidentiality for data stored on an Android phone, data stored by one app must not be read by another app. For data integrity they may not be edited by another app. This is achieved using access control mechanisms. Access control ensures, that a process can only access data that he owns a clearing for. Access control can be discretionary, mandatory or role-based.

4. ANDROID SECURITY ISSUES

The Android security lies both in the Linux kernel and in the application framework. The security inherited from Linux is the user ID. Along with this, at the framework level, a mandatory access control mechanism is enforced by Android permissions to control access between components. By requesting the proper permissions, a malicious app could launch resource abuse attacks (for example, sending large volumes of spam or launching distributed denial of service attacks), data loss attacks (stealing data from the device's calendar, contact list, etc.), and perform data availability/integrity attacks (by modifying/deleting data in calendar, contact list, or on the SD card). So, while Android implements a robust permission system, its dependence on the user and its leniency in offering access to most of the device's sub-systems compromise its effectiveness and have already opened up Android devices to attack.

4.1 Android Security Mechanisms

Android's security model is primarily based on two security mechanisms: isolation, and a permission-based security model.

4.2 Isolation (Sandboxing)

Android has two types of resources that need to be protected. One is the application resources including the processes and files, the other is the system resources, such as camera, network, radio, GPS, and various sensors. Android employs a strong isolation system to ensure that apps only access approved system resources. This isolation system not only isolates each app from other apps on the system, but also prevents apps from accessing or modifying the operating system kernel, ensuring that a malicious application cannot gain administrator-level control over a device.

4.3 Permissions-based Access Control

Permissions are the core concepts in the Android security. By default, most Android applications can do very little without explicitly requesting permission up-front from the user to access personal information and phone features (Grace et al, 2012). For example, if an application wants to communicate over the Internet, it must explicitly request permission from the user to do this; otherwise the default isolation policy blocks it from initiating direct network communications. This, thus infers that a user is expected to gauge the application's capability and determine whether or not to install it in the first place. Once granted, permissions are not revocable. The Android system can thus no longer prevent an application from misusing permissions and violating the user's privacy (Zhang and Yin, 2013).
Android offers no middle ground (i.e. allowing some permissions, but rejecting others).

Nachenberg (2011) opined that the problem with this approach is that the model ultimately relies upon the user to make all policy decisions whether or not the combination of permissions is safe or not. But unfortunately, users are not technically equipped to make these security decisions in majority of cases, where a malicious app simply requests the set of permissions it needs to operate and users happily grant these permissions. These eventually cause damage without having to bypass the permission system. In contrast, Apple's iOS platform simply denies access, under all circumstances, to many of the device's more sensitive subsystems. This increases the security of iOS-based devices since it removes the user from the security decision-making process. However, this also constrains each application's functionality, potentially limiting the utility of certain classes of iOS applications.

4.4 Application Provenance by Google Play Market

In Android, application developers can generate their own signing certificates, as often as they like, without any oversight. In fact, the software developer can place any company name and contact information that they like in their certificate. Google does not appear to perform a rigorous security analysis of applications posted onto its Android Marketplace. As indicated by Fedler et al. (2012), the majority of all infections is conducted through free illegitimate copies of repackaged apps as shown in figure 2.2. Repacking is a process of altering the logic of an app code in order to deliver malicious code.

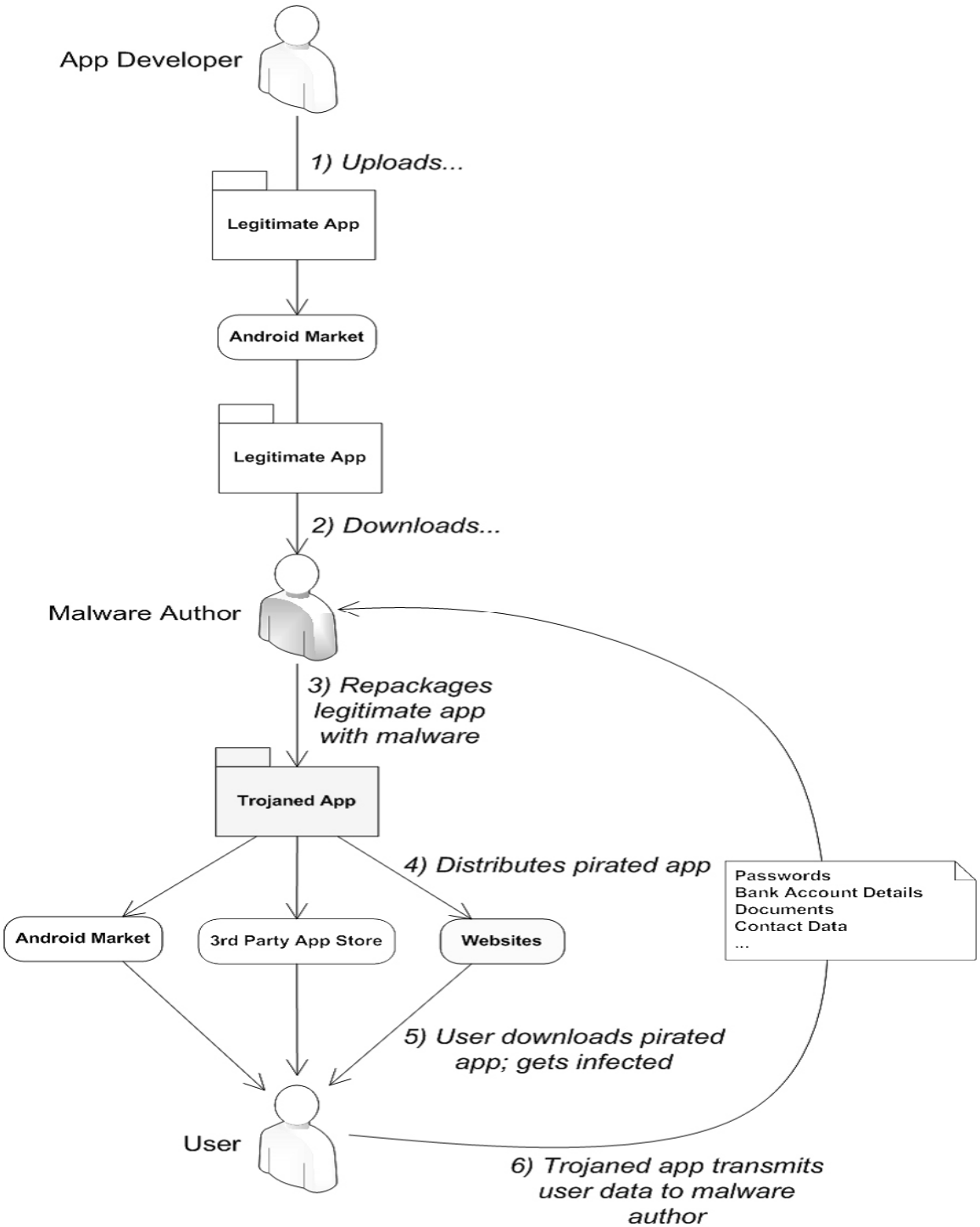

Figure 2.2: Repackaging Process (Source: Fedler et al., 2012)

This means that malware authors can distribute their apps through this distribution channel with less likelihood of being discovered. Also, Android application developers can distribute their apps from virtually any website on the Internet, and users are allowed to "side-load" apps from any source on the Internet. According to Hei et al (2013), the central principle of the Android security architecture is that no application, by default, has permission to perform any operation that would adversely impact other applications, the operating system, or the user. To meet this principle, Android sandboxes each application by combining Virtual Machines together with the Linux access control. Nachenberg (2011), revealed that each Android application runs within its own virtual machine, and each virtual machine is isolated in its own Linux process. This model ensures that no process can access the resources of any other process (unless the device is jailbroken). All protection is enforced directly by the Linux-based Android operating system.

5. ANDROID DATA STORAGE

Android data storage recognises two storage points –internal and external, the definition of each storage point depends on the Original Equipment Manufacturer (OEM).

5.1 Internal storage

Applications can store and retrieve their data in various ways. If an application wants to make sure that no other application or even the user can access saved data, it can write this data to the internal storage. By default, files saved to internal storage are private to an application, and other applications cannot access them. When the user uninstalls the application, these files are removed. Files saved to the internal storage can be set up to allow others to access the files. Google refers to the user storage as "external" storage even though it is internal to the device and not on the SD card. That is why the mount point is /mnt/sdcard and the actual SD card mount point is /mnt/extSdCard (though this varies from manufacturer to manufacturer).

5.2 External Storage

External storage of an Android OS is a location that one can use to save files. This can be a removable storage edia (such as an external SD card) or an internal (non-removable) storage situated in the device. External SD cards are world readable i.e. they are always publicly visible and accessible. Due to insufficient memory space in the internal storage of the smartphone, users store their data in SD Cards, and applications can directly access resources in this memory storage, especially when there is a shared privilege permission. Files that are supposed to be shared and/or user visible, can be stored to the external storage. Files in this storage area are always publicly visible and accessible. Android allows the external storage to be turned into a USB mass storage with full access to all files in this storage area (Braehler, 2010). That is, App A could write App B's external storage, which could be dangerous if App B was malicious.

This implies that any application that has access to read from or write to data on an SD Card can read from or write to any other application's data on the SD Card. So, the primary area of concern is the protection of information stored, transmitted and processed when users are using mobile computing devices. As it is in any Unix/Linux-like operating system, basic access control is implemented through a three-class permission model. It distinguishes between the *owner* of a file system resource, the owner's *group,* and *others*. For each of these three entities, distinct permissions can be set to read, write or execute. This system provides a means of controlling access to files and resources. For example, only a file's owner may write (alter) a document, while members of the owner's group may read it and others may not even view it at all.

6. ANDROID APPLICATION PERMISSIONS

Android's API provides access to phones' cameras, microphones, GPS, text messages, WiFi, Bluetooth, etc. Most device access is controlled by permissions, although large parts of the overall API are not protected by permissions. Applications can define their own extra permissions, but we only consider permissions defined by the Android OS. According to Felt et al (2011), there are 134 permissions in Android 2.2. Permissions are categorized into threat levels:

- ***Normal permissions:*** These are permissions that are not especially dangerous to have, and they are as well automatically granted to the application without the user's approval before/during installation process. API calls with annoying but not harmful consequences are protected with Normal permissions. Examples include accessing information about available WiFi networks, vibrating the phone, and setting the wallpaper.
- ***Dangerous permissions:*** API calls with potentially harmful consequences are protected with Dangerous permissions. These permissions are more dangerous than normal and are not normally needed by applications. They include actions that could cost the user money or leak private information. Example permissions are the ones used to protect opening a network socket, recording audio, and using the camera.
- ***Signature permissions:*** The most sensitive operations are protected with Signature permissions. These permissions are only granted to applications that have been signed with the device manufacturer's certificate, or that have the same signature as the one declaring the permission (Ismail et al, 2014). Market applications are only eligible for Signature permissions if they are updates to applications that were pre-installed and signed by the device manufacturer. Requests for Signature permissions by other applications will be ignored. An example is the ability to inject user events.
- ***SignatureOrSystem permissions:*** This category includes signed applications and applications that are installed into the /system/app folder. Typically, this only includes pre-installed applications. They are not available to third party applications. Advanced users who have rooted their phones can manually install applications into this folder, but the official Market installation process will not do so. Requests for SignatureOrSystem permissions by other applications will be ignored. For example, these permissions protect the ability to turn off the phone.

6.1 Android API Permission Model and Manifest File

On installation, the user is presented with a dialog listing all permissions requested by the app to be installed. These permission requests are defined in the Manifest File AndroidManifest.xml, which is obligatory for every Android app and the permission requests are such that either they are all granted or none is granted (Fedler et al., 2012). Although an application might request a suspicious permission among many seemingly legitimate permissions, many users will still confirm the installation. This suggests the fact that users, often, cannot judge the appropriateness of permissions for such an app in question. Another flaw in this model is circumvention. Functionality, which is supposed to be executable only given the appropriate permissions, can still be accessed with fewer permissions or even with none at all.

6.2 Manifest

Applications in Android use install-time manifests to request access to user-owned resources. Once the user agrees, the installed application has permanent access to the requested resources. This violates least-privilege, as installed applications may access these resources at any time, not just when needed to provide intended functionality. Studies indicate that many applications ask for more permissions than needed and recent Android malware outbreaks suggest that users install applications that ask for excessive permissions (Roesner et al., 2011).

6.3 Privilege Escalation

While Android implements a robust permission system, its dependence on the user and its leniency in offering access to most of the device's sub-systems compromise its effectiveness and have already opened up Android devices to attacks. In other words, with respect to accessing B, a system with unprivileged component A should behave the same as a system without A. The only exception is if additional policy explicitly allows A to affect B. Without such exceptions, this property would likely be too restrictive. The principle of least privilege demands that an application be given only those privileges needed to carry out its duties. Privilege escalation on Android occurs if an application can read the files stored by another application. This privilege escalation attack is successful if an application can exploit system resources that it is not allowed to use, but which other applications might have the permissions to use.

6.4 Android Permission Logical Model

This model covers the semantics of application and its components, permissions, and uses-permissions.

Definition 1:
A permission declares the requirements posed by a component for accessing it.
Let A_p be the permission function defined as
$A_p: C \rightarrow P$ *associates each component to a single permission.*

Where C is an application containing components and P is the set of permissions required by C (Nauman et al, 2010).

Definition 2: (Optional Permission)
We define a function of optional permission granting as
$A_{op}: C \rightarrow P_o$

Where $P_o \subseteq P$
$P_o = \{P_1, P_2, P_3, \ldots, P_n \mid P_i \in P, i = 1, 2, 3, \ldots, n\}$
So that each application C, and its associated components can be granted one or more permissions in the set of permissions P.

Definition 3: (Privilege Escalation)
Given any component B protected by permission P, and any component A that does not have that permission, if S_{AB} is a system that contains A and B (and other components), and S_B is the same system without A, then a call chain ending with B exists in S_{AB} $\leftrightarrow$ it exists in S_B. Additional call chains ending with B may exist in S_{AB} if explicitly allowed by policy (Fragkaki E. et al, 2012).

7. Android Security, Pitfalls and Lessons Learned

Liebergeld and Lange, 2013 analysed some of the security weaknesses of Android as follows:

1. Security updates are delayed or never deployed to the user's device.
2. Office Equipment Manufacturers (OEMs) weaken the security architecture of standard Android with their custom modifications.
3. The Android permission model is defective.
4. The Google Play market poses a very low barrier to malware.

In their work, they investigated the security of Android mobile OS, described its problems, and derived a set of lessons learned which will help to avoid pitfalls for future OS developers. The set of lessons are:

- ✓ *Timely security updates*: This is essentially important for open source systems where the code is public and bugs are easy to spot. They think that the key lies in clear abstractions and a modular system.
- ✓ *Control platform diversity*: Operating system designers should enforce that third party modifications to the operating system do not introduce security breaches by design.
- ✓ *Design permissions with user and developer in mind*: In order to avoid over-privileged applications by the developer, and for the user to make an educated decision when granting permissions, a permission system should be designed such that the permissions it implements are well understood by both the user and the developer. Also, since granting of permissions at installation time is either the user grants **all** or none, it makes permission granting problematic. They suggested a better solution would be to ask user to grant permissions on demand.
- ✓ *Ensure that the application does not distribute malware*: People have trust in the App Markets, and have no chance to determine the quality of an application by themselves. This, therefore, makes the App Market the most important distribution place for applications. So, they suggested that aside from having a mandatory admission process, an App Market should also scan for repackaged applications.

8. RESEARCH METHODOLOGY

In order to control access to the memory storage, the threat model must be analysed and an extension to the repackaging process of an application developed.

8.1 Threat Model

To achieve the objectives of this research paper, the threat model must be considered. The ultimate goal of digitally signing an application is twofold: one, to ensure that the application's logic is not tampered with, and two, to allow a user of the application to determine the identity of the application's author. But this is not so for Google because application developers can generate their own signing certificates, as often as they like, without any oversight. This approach presents two problems. First, it makes it much easier for malware authors to create and distribute malicious applications since these applications cannot be tracked back to their source. Second, this approach makes it easier for attackers to add Trojan horses to existing legitimate applications. For example, as depicted in Figure 3.1, an attacker can obtain a legitimate application, add some malicious logic to the application, and then re-sign the updated version with an anonymous certificate and post it onto the Internet. While the newly signed application will lose its original digital signature, Android will certify and install the newly signed malicious application with its anonymous digital signature. Thus, Android's model does not realistically prevent tampering.

8.2 The enhanced Access Control Framework (eACF)

Different mechanisms of access control try to mitigate privilege escalation attacks on Android. One of the defects of Android permission security model is Combo permission that is, applications from the same author can run with the same UID, running under a shared permission. Take for instance two applications from the same author, one has access to the SMS database because it provides full text search for a user's SMS, and the other application, e.g. a game, has access to the internet because it needs to load ads from an ad server. These two applications can communicate through Android's inter-process communication (IPC) mechanism and essentially leak the user's SMS database into the Internet. This is a problem that can be solved by making the user grant permission only on demand which led to the development of this project. The enhanced Access Control Framework (eACF) will use the Android permission scheme, as shown in Figure 3.1 (a modification to Figure 2.2 with the inclusion of eACF) to prevent access by any illegitimate application to secure folders/files in a memory storage, even if it has a shared privilege with a legitimate application through Combo permissions. The eACF is a protection mechanism that ensures that only allowed Android applications are able to access specific folder/file. It defines a permission *android.permission. WRITE_EXTERNAL_STORAGE or android.permission. READ_EXTERNAL_STORAGE* that a client application must request in its manifest in order to obtain access to the API.

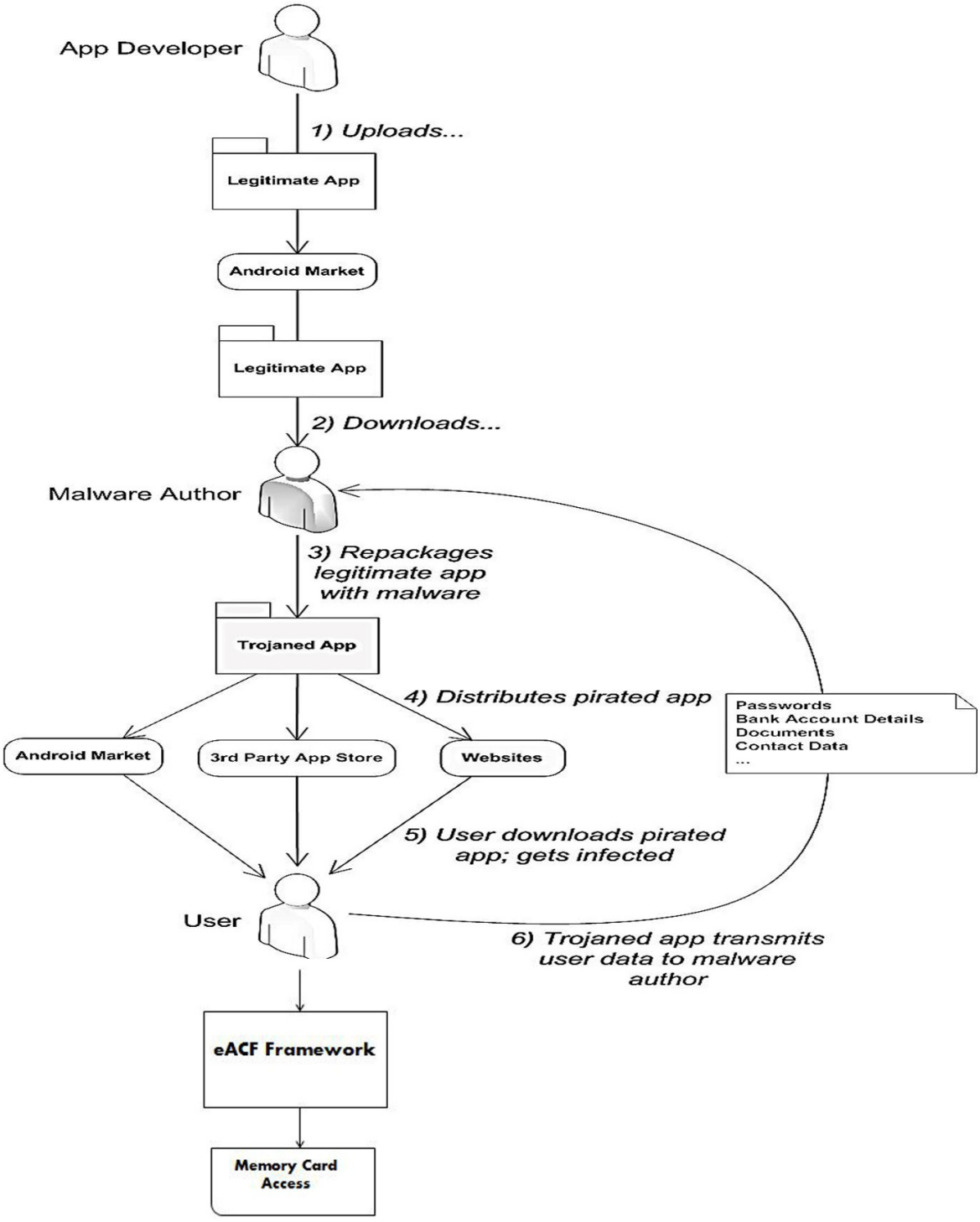

Figure 3.1: The process diagram for Repackaging Process with eACF

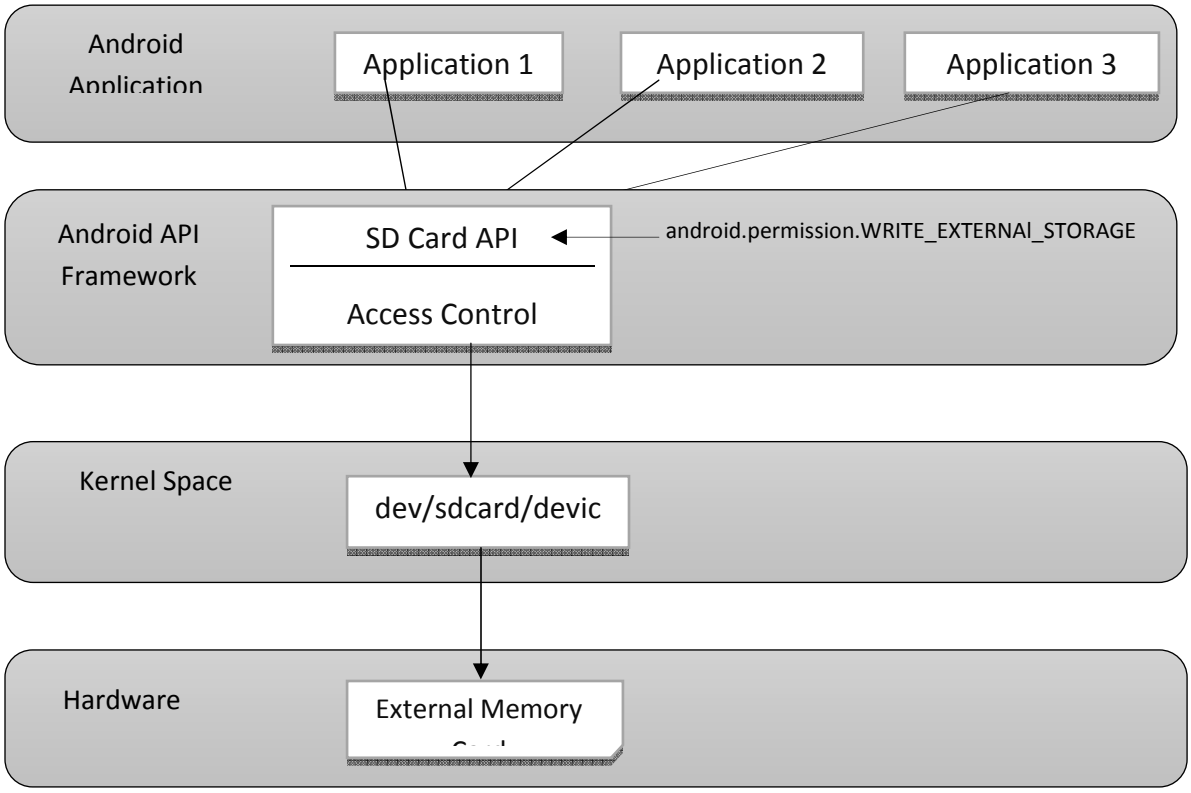

Figure 3.5: System architecture of eACF

With this framework, all newly installed applications, by default, cannot access any data category. This prevents a new application from leaking sensitive data right after installing it. After installing a new application, a dialogue box will prompt the user, asking to what level the application can be granted access. For example, if an application's access to the Internet and to external storage (typically an SD card) are restricted by denying access, and if perhaps, some applications still try to access the Internet or the external storage, it potentially results in crashes or error messages.

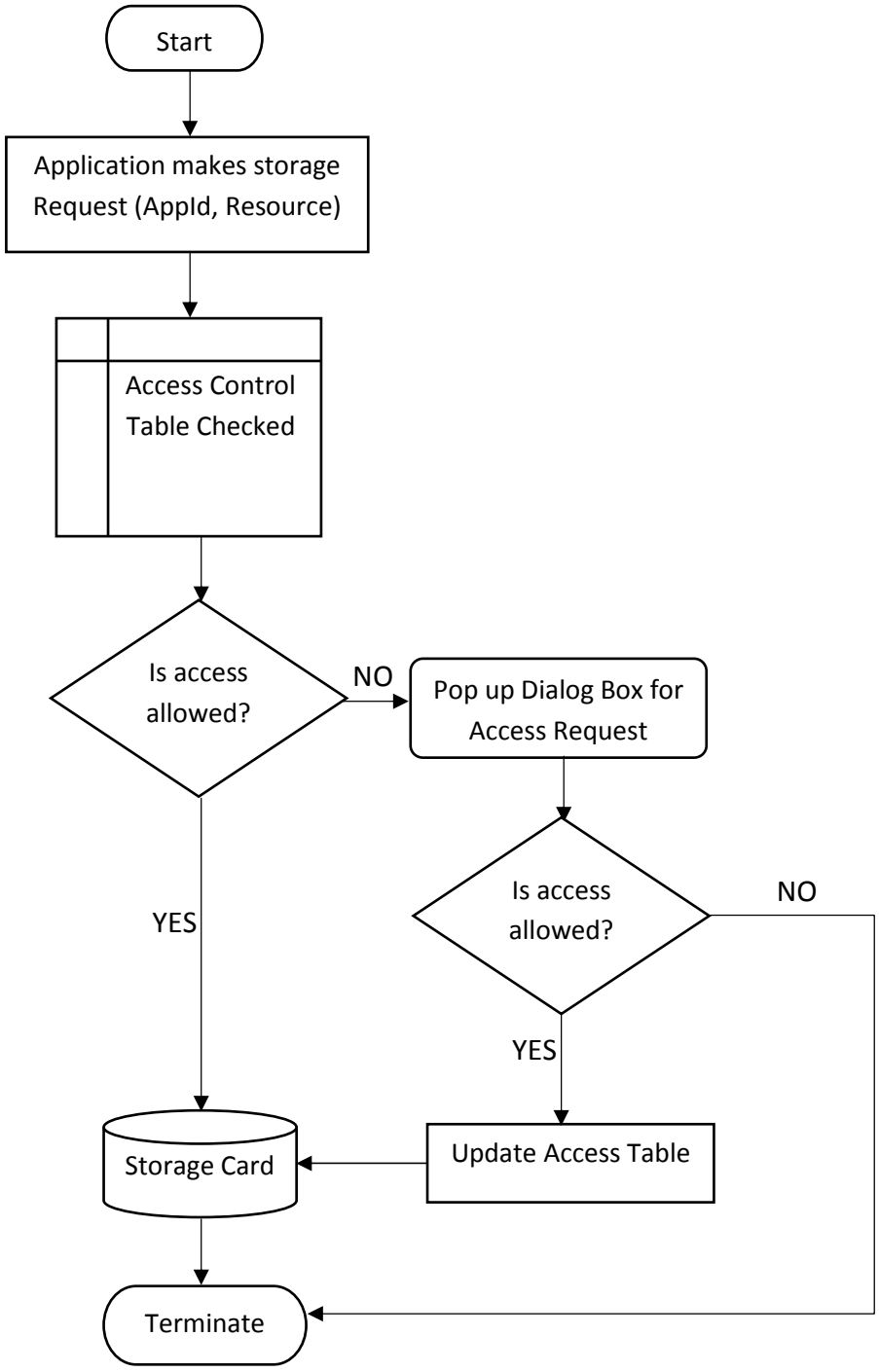

Figure 3.3: Operational flow chart for eACF

If, for example, a client Application X wants to access a specific folder created by another application, the Access Control Framework (eACF) reads the Access Control Table (ACT) for the specific folder's AppID and the application's access permission mode, as shown in Table 3.1. The user then either grants access or denies access to the client Application X according to the permission mode stated on the ACT.

Table 3.1: Access Control Table (ACT)

Application ID	Resource	Access
App1	dev/sdcard	Allow
App2	dev/sdcard	Deny
App2	dev/sdcard	Deny

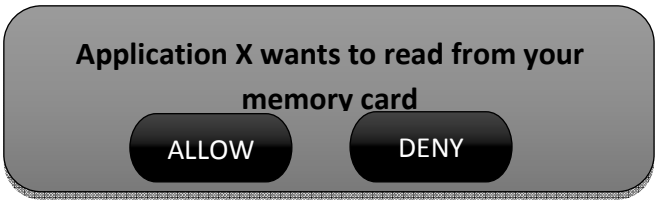

Figure 3.4: An interface for an application requesting permission

8.3 System Requirement

The required mobile platform for this work is Android OS Versions from 2.3 to 4.2.2.
Linux (and Android) has a very strict permission policy which restricts the usage of administrative functions. Therefore, the smartphones on which eACF was implemented were rooted because superuser access (i.e. rooting) is required in order to gain administrative control on the smartphone. All Android OEMs prevent users from gaining administrative privilege which may eventually hinder the implementation of the framework if not rooted. The handset brand that was used for the implementation is Samsung. Why Samsung? The reason for choosing Samsung as the choice of testing device is because Samsung is an Android-based well-established OEM which provides the advantage of testing with both internal memory space and external memory slot.

9. RESULTS AND DISCUSSION

The first selection box, when selected, restricts an application while the second box, when selected, displays an on-demand pop up box, as shown in Figure 4.1. In the app interface shown in Figure 4.2 with marked selection boxes, the first group of the selection boxes, if marked, depicts that if the user fails to decide whether to "allow" or "deny" before the progress bar times out during the prompting by the system, it automatically observes the already-declared constraint to restrict the application from accessing the checked permissions. The second group of selection boxes, with question mark, means that the marked resource(s) will be accessed only on demand with an on-demand pop-up box to alert the user.

Figure 4.1: The main app interface displaying all apps

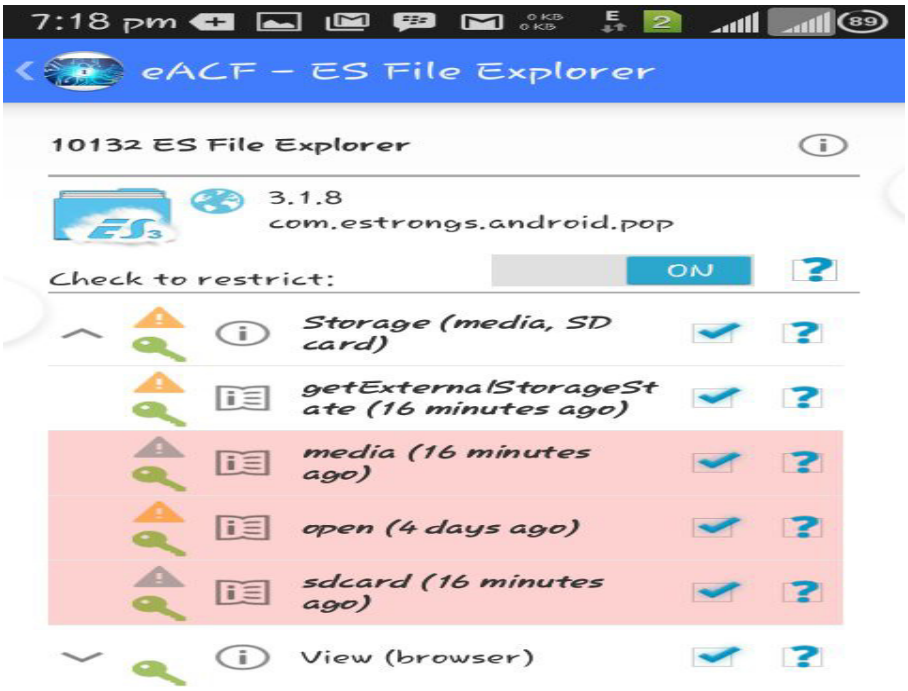

Figure 4.2: The App interface showing marked selection boxes

10. TESTING AND RESULTS

At the installation time of any mobile application as shown in figures 4.3, the application declares a list of permissions requesting access to some resources in the device as shown in the figure below.

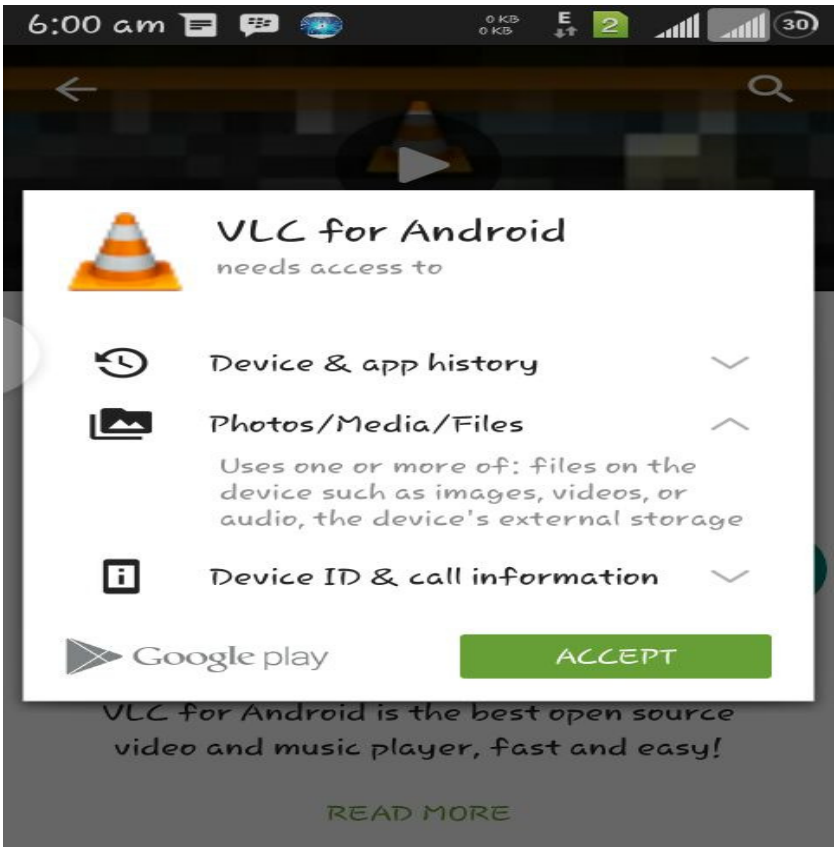

Figure 4.3: A video player app with the permissions required at install time

A download manager application (LoaderDroid) for Android shown in Figure 4.4 depicts an on-demand pop-up box for LoaderDroid. Here, the application is requesting permission to view a file. The selection box "Apply to entire category" means that the category of the file to be accessed is made open to the app such that the app will require no permission request again to access the particular resource next time it attempts to access. This is useful when a new application that is trusted is attempting to access that category after installation. This shows that there will be no pop-up for a trusted new app next time. If the selection box "Once for 15 seconds" is marked, this is when an app can be allowed for just a period of time, based on trust and necessity. This is useful when an app that has its resource on the storage card needs its resource at runtime. For example, a not-totally trusted app can be given permission to read storage card at runtime for just a while (say 15 seconds) if the resource needed for the app to run at that particular point in time is located in the storage card.
The yellow-coloured bar (i.e. the progress bar) counts down till the user clicks on either "Allow" or "Deny", else, the app blocks if time runs out.

The "whitelist" selection box shown on the interface, if marked, means that a folder/file is whitelisted. Whitelisting a folder/file implies that the app is trusted to access that particular folder/file anytime without requesting another permission from the user. But this privilege does not cover any other folder/file except this whitelisted folder/file. If the app attempts to access another folder/file (apart from the whitelisted one), then the app must request for a new permission, if not, it will be restricted as shown in Figure 4.5.
The search application in Figure 4.5 attempts to use the privilege of a whitelisted file to view a file that was not whitelisted but was restricted by eACF.

Figure 4.4: An on-demand pop-up box showing for LoaderDroid

Figure 4.5: A search application being restricted by eACF

A permission to access external storage request was made by a game application at start up, when access was denied, the application crashed, because it was not able to access an important file needed for its smooth running due to the denial.

11. EVALUATION OF EACF

eACF also monitors all application's attempts to access sensitive data in order to identify potential data leaks.
The evaluation of the framework was carried out by measuring the efficiency through calculating the on-demand count (i.e. pop up count) versus the log count. The pop up count is the number of times a particular application requests access to a resource through a pop up message that was designed to alert the user. The log count is the number of times an attempt was made to access the resource.

Efficiency = ((work output/ work input) * 100) %
Efficiency is low if: 0 ≤ efficiency ≤ 60
Efficiency is good if: 60 ≤ efficiency ≤ 100

Table 4.1: Log count vs Pop up count

App Name	Pop up Count	Log Count
UC Web	2	3
WhatsApp	3	4
AppLock	2	4
iReader	1	2
Opera Mini	1	2
MxPlayer	1	1
PhotoGrid	2	3
Es-xplorer	1	1
G-Map	3	4
BBM	2	3
Facebook	1	2
Dropbox	2	3
LoaderDroid	1	2

Taking log count as work input, the total log count for the tested application is 34.

Therefore:
Work input = 34 units

Also taking pop-up count as work done,

Work output = 22 units

Hence, Efficiency = ((22/34) * 100) % = 64.71%

Although, the performance of eACF could be higher, the performance reduction was as a result of displaying a single pop-up for both SDcard access and media access. If both count are regarded as one in the log count, then

Work input = 22 units

Then,

Efficiency = ((22/22) * 100) %
= 100%

12. CONCLUSION

From this research work, it can be concluded that the framework improves significantly over the existing access control mechanism. Notably, the framework also gives good performance on access restriction to the use of resources stored in the external storage by applications and keeping of the log details to monitor resources usage. The performance of our enhanced access control framework, eACF, was demonstrated on several third-party applications. It can then be concluded that this framework is significantly different from related works (Ismail et al., 2014) in that it allows users to make decisions about permissions on their device at the time of use rather than employing the all-or-nothing permissions granting policy of Android.

REFERENCES

1. Android OS http://www.android.com Accessed on 24th May, 2014.
2. Becher, M., Freiling, F. C., Hoffman, J., Holz, T., Uellenbeck, S., and Wolf, C. (2011). MobileSecurity Catching Up? Revealing the Nuts and Bolts of the Security of Mobile Devices,IEEE Symposium on Security and Privacy.
3. Berger, B. J., Bunke, M., and Sohr, K. (2011). An Android Security Case Study with Bauhaus,Working Conference on Reverse Engineering, pp 179-183.
4. Braehler, S. (2010). Analysis of the Android Architecture. Thesis. Fakultat fur Informatik, Karlsruher Institut fur Technologie (KIT).
5. Carlson, T. (2011). Information Security Management: Understanding ISO 17799.http://www.gta.ufrj.br/ensino/cpe728/03_ins_info_security_iso_17799_1101.pdf
6. Cloud Security Alliance (2012). Security Guidance for Critical Areas of Mobile Computing, CSA Mobile Working Group v1.0.
7. Dar, M., Parvez, J. (2013). Evaluating Smartphone Application Security: A case study on Android. Global Journal of Computer Science & Technology (GJCST-E classification), Vol. XIII, Issue XII, version 1.0.
8. Deepak, G. and Pradeep, B. S. (2012). Challenging Issues and Limitations of Mobile Computing, International Journal on Computer Technology & Applications (IJCTA), Vol. III (1): pp. 177-181.
9. Enck, W., Octeau, D., McDaniel, P., and Chaudhuri, S. (2011). A study of Android Application Security, Proceedings of the 20th USENIX Conference on Security SEC'11, USENIX Association, pp. 21-21.
10. Fedler, R., Banse, C., Krauss, C., and Fusenig, V. (2012). Android OS Security: Risks and Limitations A Practical Evaluation, Fraunhofer Research Institution for Applied and Integrated Security (AISEC).
11. Felt, A. P., Greenwood, K., And Wagner, D. (2011). The Effectiveness of Application Permissions, USENIX Conference on Web Application Development.
12. Fragkaki, E., Bauer, L., Jia, L. and Swasey D. (2012). Modeling and Enhancing Android's Permission System, Proceedings of 17th ESORICS, Pisa, Italy. Vol. 7459, pp 1-18.
13. Grace, M., Zhou, Y., Wang, Z., Jiang, X. (2012). Systematic Detection of Capability Leaks in Stock Android Smartphones, Proceedings of the 19th Annual Network & Distributed System Security Symposium (NDSS).
14. Gunadi, H. and Tiu, A. (2014). Efficient Runtime Monitoring with Metric Temporal Logic: A Case Study in the Android Operating System, Proceedings of 19th International Symposium, Singapore. Vol. 8442, pp 296-311.
15. Hao, H., Singh, V., and Du, W. (2013). On the Effectiveness of API-level Access Control Using Bytecode Rewriting in Android, Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer & Communications Security (ASIA CCS'13), pp 25-36.
16. Hei, X., Du, X., and Lin, S. (2013). Two Vulnerabilities in Android OS Kernel, Proceedings of IEEE International Conference on Communications (ICC'13), pp 6123-6127.
17. Ismail, M., Aboelseoud, H., Senousy, M. (2014). An Investigation into Access Control in Various Types of Operating Systems, International Journal of Computer Applications (0975-8887) Vol. 98, No.10.
18. Liebergeld, S. and Lange, M. (2013). Android Security, Pitfalls and Lessons Learned, Proceedings of the 28th International Symposium on Computer & Information Sciences (ISCIS'13) Vol. 264, pp 409-417.
19. Mohini, T., Kumar, S. A., and Nitesh, G. (2013). Review on Android and Smartphone Security, Research Journal of Computer and Information Technology Sciences, vol. I (6).
20. Nauman, M., Khan, S., and Zhang, X. (2010). Apex: Extending android permission model and enforcement with user-defined runtime constraints, Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security. ACM, pp. 328–332.
21. Pesante, L. (2008). Introduction to Information Security Carnegie Mellon University.
22. Roesner, F., Kohno, T., Moshchuk, A., Parno, B., Wang, H. J. and Cowan, C. (2012). User-Driven Access Control: Rethinking Permission Granting in Modern Operating Systems, Proceedings of the IEEE Symposium on Security & Privacy, IEEE Computer Society, Washington DC, USA, pp 224-238.
23. Security Tips http://developer.android.com/training/articles/security-tips.html. Accessed on 23rd May, 2014.
24. Symantec (2011). A Window into Mobile Device Security, Symantec Security Technology and Response.
25. Xu, R., Saidi, H., Anderson, R. (2012). Aurasium: Practical Policy Enforcement for Android Applications, Proceedings of the 21st USENIX Conference on Security Symposium.
26. Yang, F., Zhou, X., and Hu, D. (2013). A general Mandatory Access Control Framework in Distributed Environments, International Journal of Computer, Information, Systems and Control Eng Vol. 7, No. 10.
27. Zhang, M. and Yin, H. (2013). Transforming and Taming Privacy-Breaching Android Applications, Proceedings of the Network & Distributed System Security Symposium (NDSS).