1-1-2015

A Software Development Model for Building Security into Applications for the Android Platform

Christopher Patrick Ivancic

A software development model for building security into applications for the Android platform

By

Christopher Patrick Ivancic

A Dissertation
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

August 2015

A software development model for building security

into applications for the Android platform

By

Christopher Patrick Ivancic

Approved:

David A. Dampier
(Major Professor)

Donna S. Reese
(Committee Member)

Alfred Christopher Bogen
(Committee Member)

Robert Wesley McGrew
(Committee Member)

T. J. Jankun-Kelly
(Graduate Coordinator)

Jason M. Keith
Dean
Bagley College of Engineering

Name: Christopher Patrick Ivancic

Date of Degree: August 14, 2015

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. David A. Dampier

Title of Study: A software development model for building security into applications for the Android platform

Pages of Study: 106

Candidate for Degree of Doctor of Philosophy

The popularity of smart phones has risen throughout the years since first introduced. With the popularity of the devices growing so too has the number of malicious applications flooding the devices' marketplaces. With more usage there becomes a larger target for malware and exploitation creation. As threats to these devices continue to grow there is a constant need for security to safeguard against these threats. Some attempts to protect smart phones involve building software to analyze applications running on the devices. This attempt has cut back on the amount of malicious software on the marketplace. These attempts however only catch malicious applications after they have been running.

This dissertation presents the Secure Android Development Model. The goal of this model is to contribute to security of these devices by having a development model that implicitly builds security into applications. The model ensures a minimal amount of open permissions thus limiting the number of attack vectors that malicious software can make

on the devices. By following the model, developers will have all information available during development to make appropriate security decisions in their applications.

Key words: smartphone, software life cycle, Android, permission, malware

DEDICATION

I dedicate this dissertation to my wife Nicole Ivancic and my sons Henry and Jacen. The constant support has made my entire time in graduate school possible.

ACKNOWLEDGEMENTS

The author would like to express his gratitude to many individuals who provided support through this research work. First I would like to thank my wife Nicole Ivancic for all of her patience and support through my dissertation process. Without it I would have had a much harder time staying focused. I would like to thank my family for their support as well. Dr. David A. Dampier, I would like to thank you for all of the guidance and help given to me throughout my research and supporting me along the way. To Dr.s Bogen, Reese, and McGrew I would like to thank for serving on my committee and contributing to the process. I would like to thank Dr. Nan Niu who gave help and guidance with his expertise in the field of Requirements Engineering. I would like to thank the administration at the Computer Science and Engineering department and Bagely College of Engineering at Mississippi State University. I also would like to thank the Software Engineering Senior Design team for helping test my research and going through the process with me, and to all CSE students and industry developers who helped test the model I thank you as well.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Smartphones have become more widely used with more than half (56%) of US citizens using smartphones[27]. Standard cellular phones (cell phones) were used for making phone calls and sending Short Message Service (SMS) messages. As such, standard cell phones provided little multimedia capabilities. Smartphones are designed as multimedia devices with the ability to read email, read and write documents, and browse the Internet. The emphasis on phones being used for multimedia purposes has made smartphones a more central part of people's days.

Smartphones are used more and more by individuals and companies as ways to stay connected to each other. Smartphones have grown in popularity making application development more mainstream. Increased application development leads to apps for the devices that make storing, retrieving, and sharing information easy to accomplish. With the ease of information sharing, smartphones make it that much easier to stay connected for business and personal use.[27]

The growth in usage has led to growth in malicious software being written to target the increased user base. Security of these devices is a growing concern and many attempts at securing them have been made[2, 4, 7, 8, 15, 23, 29, 32]. Most attempts focus on fixing a

problem with the operating system (OS) or finding running malware and removing it. This dissertation proposes a model for software development for the Android operating system that will implicitly build security into new applications developed attempting to minimize the risk of these applications being exploited by malicious software on the devices. The goal is to build the protection into the applications to prevent misuse, rather than detect the misuse and remove it.

This chapter will define what smartphones and mobile devices are and will discuss how malware is written for these devices. It will then introduce the hypothesis and contribution of this dissertation.

1.1 Smartphones and Mobile Devices

Smartphones differ from standard cell phones in that they provide additional functionality. A cell phone's primary functionality is to send and receive phone calls and text messages. "Smartphone" is the term given to a device that is still a cell phone, but more emphasis is placed on the device's additional functionality. Smartphones are designed to be portable media centers in addition to being a phone.

Internet usage and email reading were early marketing points for smartphones. With Internet capabilities, these devices became more like personal computers. These devices are easy to use and can store a user's personal documents to be read and used at any time. The convenience that smartphones allow has led to growth in their usage over the years and smartphones are now used by more than half of American adults[27].

Attempts at building these Swiss Army Knife devices began as early as 1993 with the IBM Simon[26]. In addition to being a cell phone, the device also acted as a Personal Digital Assistant (PDA) and as a mobile fax machine (it could send digital documents to other fax machines). This phone was expensive at $899 and was extremely cumbersome.

Palm Inc. introduced the Palm Pilot in 1996. The Palm Pilot was not a mobile phone but it was a mobile PDA. This device became commonly used to store documents and sync calendars. The device was light and inexpensive, thus it caught on more quickly than other devices of the time.

Additional progress was halted until around 2002 when Research In Motion developed the Black Berry. This device was the first commercial success of combining a Palm Pilot with a mobile phone. This device had Internet and email capabilities in addition to being a phone. The Blackberry was mostly used for businesses and large enterprises. This made the device reside solely within a niche market.

The first success of non business acceptance of smartphones came in 2007 when Apple released the iPhone. Apple combined all of their multi-media features present in their iPods with a mobile phone. These devices could browse the Internet, check email, play movies and music, read and write documents, take video and pictures, and play games. These devices were marketed for personal use compared to Blackberry's market of business use. Shortly after release of the iPhone, Google released its new mobile device operating system Android. This gave Apple its competition in the same personal use market for smartphones. These two companies successfully marketed the idea of a smartphone to home users and use of these devices has been increasing ever since.

Today, users have access to many different models of iPhone, many different models running Android OS, and now Microsoft has a mobile Windows OS for smartphones in the market as well. Smartphones have become so ubiquitous that these companies have even started taking the idea of a smartphone and putting it on larger devices giving us our mobile tablets. Apple has several versions of the iPad and many different Android based tablets exist. E-readers such as Nook Color and the Kindle Fire run their own version of Android. In addition to smartphones, these other mobile devices have many of the same features. This also means that they have many of the same flaws. All of these devices have security issues and the more widespread use they gain, the more they become the targets of malicious software.

The growing threats to these devices warrant an increased effort in protecting them. Security models need to be developed and updated to react to the growing problems facing these devices.

1.2 Built in SmartPhone Security

One way to secure a computer is to limit the privileges of the user. When a user acquires a new smartphone, the policy in place is to not allow the user to have root level access. User level access is granted by the OS to prevent the user of the device from having complete control of it. This is done as a form of security by having the user require permission from the OS to install new applications instead of being able to install anything the user wishes.

Several techniques exist to give system level access of the smartphone to the user. These techniques are referred to as jailbreaking or rooting the device. This technique is

accomplished by exploiting different bugs in the system to flash a hacked version of the OS onto the device. This new version of the OS does not have the permission blocks that prevent root access to the user. The majority of users do not ”jailbreak” their phones and leave the normal OS installed.

Companies, such as Google and Apple, rely on the restricted access to achieve a level of security. By rooting the device, the user voids any support and security these companies are building into the system and are left on their own. For non-rooted devices, companies maintain control over their App stores and marketplaces. Whenever a new application is built it is submitted to the marketplaces for distribution. Upon arrival to the marketplaces a team analyzes the app for potential problems before it is released. The degree of scrutiny differs between the companies. Apple spends more time in the beginning checking Apps for known problems before being released but offers little information about the app itself upon installation. Google allows the community to decide if an app has flaws by giving permissions the app needs upon installation and will remove Apps the community determines unsafe.

Marketplaces are designed to minimize the threats of malicious software from being installed on a user's device. This does make it harder for malicious software to spread but many techniques are still used to infect devices using normal user level privileges [14, 21, 22, 17, 9]. When apps like this are discovered, they are eventually removed from the marketplace but by then the damage has been done. The current security policies of these OSs, like iOS and Android, attempt to limit these actions but still need improvement

and many attempts [15, 29, 4, 3, 23, 6] to improve security and attempts to educate users are still being designed and implemented.

1.3 Malicious Software Design

Developers have different reasons for designing malware (financial gain, blackmail, novelty, etc.), but any malicious software needs users to install it. It is for this reason that malware authors design their malware to target operating systems used by the most people.[14] A malware developer will develop malicious software for Windows over other operating systems simply because of how much more widely used it is. As other OSs for personal computers have grown more popular, so too has the interest in creating malware for these operating systems.

Smartphones face the same trend in malicious software. When the technology was first introduced, businesses were the primary consumer. Blackberrys were the dominent device and the market was small and specific. With such a small target for developers, there was little malware written for these devices[14]. As mentioned in section 1.1, smarthpones became more widely used by casual consumers. It became possible for people to use the devices for more personal reasons from entertainment to keeping financial records close by. As more smartphones were being sold, the app markets for these devices received more app submissions. With more apps on the market, the desire to use these devices for personal use continued to grow causing the smartphones to be more widely used[14].

The increased usage of smartphones also increased the reliance on the devices allowing for more targets of malicious software. No longer would the malware be used to target a

smaller niche group, but now it would be used to target a more diverse group of individuals giving more opportunity for their malware to be installed. What the world saw after 2007 was an increase in malware developed specifically for smartphones appearing on the markets[14].

Malware authors had two problems to face with these new targets: They had to get around the markets and they had to deal with user level access to the device. Unlike normal desktops, the smartphones only give the user non-root access to the device. This makes it harder to write code to access system level functionality and install the code. Third party apps have a limited access to the phones forcing malware development to conform to the same user level restrictions. The user cannot install an application that requests system level permissions. Malware has to be able to perform its functions at a user level in order to work on these devices.

The other problem facing malware developers is the app market. Android allows the ability to install "Unknown Sources" meaning that the app did not specifically come from the Android market but from a third-party programmer. This can get around the market but have to make the app easily available and is hard to advertise your app without being on the market. To hit the largest number of people, a developer needs to place his or her app on the market. With the app on the market, teams working for these companies have the power to remove any app if found to be malicious either by their own security team or as a response from the users. To get the app on the market, it is first tested for vulnerabilities before being released. This means that the developers have to give their malware to the companies and hope the malware passes inspection and is placed on the app stores.

The malware designed for these devices attempts to hide its activities. An app that is taking personal information is easy to spot and will quickly be removed. An app that can hide its activities by going through other apps on the device is much harder to track[21, 9]. While attempts have been made to scan for these types of attacks, if normal applications did not have vulnerabilities, these types of attacks would not be feasible. This dissertation looks at the approach of removing vulnerabilities from non-malicious apps in development.

1.4 Contribution and Hypothesis

When dealing with malicious software on smartphones, users can attempt to find the malware and remove it. Attempts at detecting malicious activity have been made and require the application to be installed and running on the device[15, 4, 2, 23]. When this happens, the malware is installed and running until it can get detected. This method of protection does not prevent the malicious software from running; rather, it attempts to find and remove the malware in a reactionary way.

Educating users of the risk with installing applications will help reduce the amount of malware installed on their devices. The difficulty of this approach is preventing the malware from being installed. Attempts have been made to educate users on the effects of malware on devices and give policies to the users that will minimize their risk from malicious applications[22, 3, 8, 19]. A policy based approach helps give guidelines to the user of the device to help protect it. This policy forms the rules the user must follow for his or her device to maintain an accepted level of security. The ultimate security of the

device is still in the hands of the user. While education helps the user with decisions, the decisions are still made by the end user.

Android gives the user a policy for using the device and gives a set of tools designed to scan for installed and running malware. This ultimately puts the security in the user's hands. As mentioned before [21, 9], most malware for these devices runs in user space and relies on vulnerabilities in the permissions of currently installed apps to accomplish their goals. This malware uses currently installed apps to retrieve any information it needs from the system while hiding its intention. Instead of relying on the user making the best decision, most permission-based vulnerabilities can be prevented during the development of the application.

Software development models have been made with the purpose of giving a process to developers for designing and building software. The goal of these models is to have the developers plan every step in the building of the software with the end result being software with fewer bugs and shorter maintenance phases. The better the software is built in the beginning, the less time is spent fixing errors in the final product.[13] This is accomplished by deciding complete functionality before the development begins.

Preventing permission misuse on mobile devices is a big step in preventing malware from misusing the device. When a developer begins building an app, it is easy to leave permissions open for later use or for functionality that the developer may want later on in development.[11] By planning before building, a developer can establish all permissions needed ahead of development. This will lead to permissions only being set that the developer needs to be set. This dissertation proposes a model of software development that is

tailored specifically to designing permissions of an Android application in an attempt to minimize the permissions used. With fewer permissions being open, an app has a much lower chance of being misused by other apps. Much like a software development model is designed to map out functionality before programming, the author's model is designed to map out permissions and functionality before development and includes a checking process in the last phase to ensure minimal permission setting.

The hypothesis of this dissertation is as follows:

> *The use of the Secure Android Development Model will allow developers to determine the minimal set of permissions needed for the application to function, resulting in an app that has a minimal set of permissions.*

The researcher defines minimal permissions as the set of permissions of an application such that the removal of any one of the permissions will prevent the application from meeting its functional requirements. An application should not have any more permissions than is absolutely necessary to function. Every permission opened is another vulnerability that can be exploited and this dissertation attempts to solve the problem by preventing permission misuse at the developer level, not the user level.

The rest of the dissertation is structured as follows: Chapter 2 will discuss all previous work done with mobile device security and the Software Development Life Cycle. Chapter 3 discusses the Secure Android Development Model. Chapter 4 details the testing procedure for the model. Chapter 5 will discuss the analysis of the tests from chapter 4. Chapter 6 will detail the conclusions and future work of the dissertation.

CHAPTER 2

PREVIOUS WORK

There has been an increase in malicious software flooding the app markets for smartphones. This is a result of the growing usage and growing number of different types of smartphones. With the increased threat of the devices, the need for securing them is a growing concern. This chapter outlines the work previously done in this field.

2.1 Threats to Mobile Devices

With the increased usage in smartphone devices, there has been an increase in instances of malicious software making its way to the devices. With more users of these devices, there is more incentive for developers to get malicious software installed on these devices. Adrienne Felt et al. compiled a survey of malware that was found out on the marketplaces between the 2009 and 2011.

Figure 2.1, taken from [14], displays the timeline of malicious applications. From this timeline, the researchers showed that these devices are susceptible to malicious attacks like other digital devices. The amount of malware on Android started growing faster towards the end of the timeline. This is around the time Android started gathering a larger market share.[14] This section outlines the kinds of malicious applications and what their attack goals are on these devices.

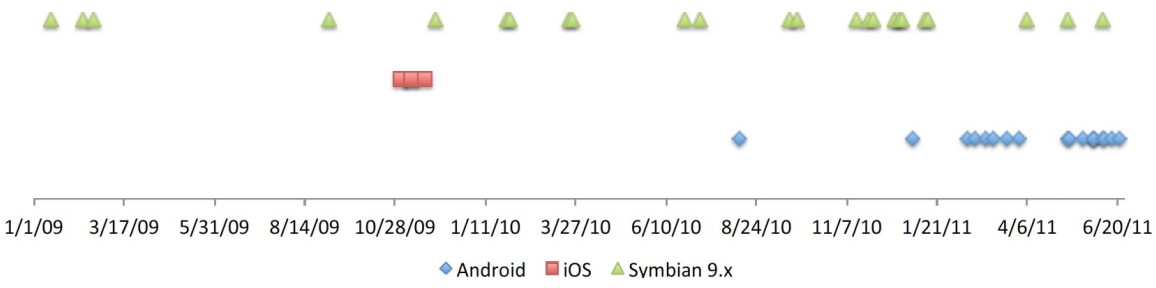

Figure 2.1

Timeline of Malicious Applications Between 2009 and 2011

2.1.1 Malicious Application Types

Malicious applications can be installed and run on the device like any other application. These can be downloaded from websites and side loaded (a process of installing an application from an unknown source) or they can be downloaded from the market place. While most market places do build in checking to filter out applications that could be malicious, there are some that sneak by [14, 33]. These applications can all have a different reason for existing but they can be classified into three main categories according to [14]: Malware, Personal Spyware, and Grayware.

Malware is an application that gains access to the device for the purpose of stealing data, damaging data, or harassing the user. The malware can reside on the marketplace with hidden usage or can be directly given to the user by the attacker. Once installed, the malware allows the user to connect remotely to retrieve data. Trojans, worms, botnets, and viruses all fall under malware by this classification and all can exist on mobile devices.

Personal spyware has some similarities to malware. The overall goal is to install software on the device that can monitor the usage of the device. The user of the software

knowingly installs it on the device with the intent of monitoring how the device is used. This is done to spy on someone close by whose device the user has physical access to, or simply to monitor his or her own personal phone usage. Then end result is similar in that the software can monitor data usage, phone calls, text messages, etc. and allows a particular user to retrieve this information as his or her will.

Grayware is the term given to applications that collect user data for the purposes of companies to gain targeted marketing information. These applications are not defrauding the user because the companies will disclose the purpose of the application and the purpose of the data collection. Companies can use it by giving ambiguous descriptions of the purpose of the software. Usage of this software is legal.

While each application that can be developed as a threat has its own specific purpose on a case-to-case base, the application can be classified in one of the three above threat types. Malware is the one where the user of the device is tricked in some way into installing a piece of software whose overall goal is to compromise the security of the user's device. With the other two types, the user is aware of the applications intent upon installation. Even though a user intends for an application to have access to some of the personal data of the device does not remove it as a threat. These applications are still a threat to the security on the device because there is a user level application (as opposed to the OS level applications built into the devices) that can access personal information of the device. With permission settings used to place some security into the OS, another user level application that is malware can still compromise one that is not and now the malware has access to the information as well[9, 12, 33].

2.1.2 Attack Goals of Malware on Mobile Devices

In their research of mobile malware in the wild [14], Adrienne Felt et al. collected 46 individual pieces of malware and classified it by its behavior. Table 2.1 is taken from [14] and summarizes the behavior of the malware observed.

Table 2.1

Mobile Malware Behavior Count

Behavior	**Count**
Exfiltrates user inforamation	28
Premium calls of SMS	24
Sends SMS advertisement spam	8
Novelty and amusement	6
Exfiltrates user credentials	4
Search engine optimization	1
Ransom	1

One common goal of malware developers is to gain information (personal or other) about the user of the system infected. This can be anything from passwords, to emails, to documents with account information. The same is true for malware running on mobile devices. These devices are becoming more ubiquitous and contain more and more of users' personal information [5]. Malware exists on these devices for stealing user information and user credentials from the devices. Typical user credentials stolen are passwords and bank account information. These devices allow for a user to check banking information while away from their personal computers. This malware intercepts the keyboard presses and reads the information presented back to the user. The user information stolen can be

information such as location, contacts, installed applications, text messages, and browser history. This is all information that is used for marketing. The malware takes this information and the creator sells this information to advertising and marketing companies. A large portion of malware in existence is designed to harvest user data for these purposes.

Malware can also be responsible for taking control of the mobile device and using the device to make phone calls and send SMS to premium numbers (1-900 numbers). This malware is capable of sending these messages and making silent phone calls to these numbers without having the device even display the number pad [14, 33]. The malware is allowing charges to be made by these numbers to the phone companies. The malware is capable of doing this by having permission of the messaging and phone call systems. The device will be in a user's pocket and dial out to this number without any sort of indication from the phone that the call is being made.

Along the same lines as dialing out to a premium number, some malware is capable of using the SMS capabilities of one device to send out a group message to the contact list of the user's device with advertisement messages. This is no different from a company sending out a spam SMS from their computers but is illegal in most countries.[14] The malware is designed to get around this problem by masking the company doing the spamming by getting non-commercial devices to spam the messages for them. By having these devices send the spam message themselves, it does not directly point back to the original source and has to be traced back through all the compromised devices.[14]

Some malware exists for the sole purpose of existing. This type of malware is created out of the novelty of having it out "in the wild". Rarely does the creator of the malware

intend on using the application for any real malicious use. This malware exists because the creators want to test or prove something. When a new update to an iPhone comes out and a developer notices a bug in the update, the developer can create an app to exploit this bug. The developer does not intend to hurt people's devices but wants to test to see if the malware will work. Some companies offer marketplaces to try and keep the malware from being placed on the marketplace. This causes another test for developers to try and take on. A developer can look at the marketplace and attempt to try and get an application placed on the market with malware packaged in with it. These pieces of software are not attempting to steal the user information from the devices. This software is designed for the amusement of the developer but still has compromised access to the mobile device it is installed on and this can lead to exploitation from other malware on these devices[14].

Malware can also be used for blackmail. This type of malware is intended to take control of the mobile device and restrict usage and access from the user. The malware developers will lock a device out of use and only unlock it after being paid. A Dutch worm (no official name, the hacker who made it was Dutch) would lock the screens of iPhones and demand that the users pay 5 euros to have their screens unlocked[14]. A desktop Trojan Kenzero existed to steal the browser history of the infected computer[14]. This information was then published publicly on the Internet with the user's name associated with it. For 1500 yen, the information would be taken down. As mentioned before, malware exists on these devices that can read a person's Internet browsing history. The same idea exists on malware of using this information in attempt to extort money from the user[14].

Just like using malware to send SMS and phone calls to premium numbers, malware can also use these devices to send SMS and web searches to certain websites. The idea is the same as the premium number spamming. The malware uses the device to generate traffic to these sites. This is not done to charge money to the phone companies like calling a premium number but is done to increase traffic to websites. Search engines prioritize results from searches based on the number of traffic hits certain sites receive. By having different devices constantly search and go to a site, it inflates the traffic the site is receiving helping make them a larger priority by the search engines[14, 33]. Researchers found this to be a low priority malware but this type of malware has grown more common. Developers making malware to generate network traffic has caused the growth in this type of attack. Advertisement on these sites pay for hits and traffic flow of the sites. By having mobile devices contact these sites, it generates more money from the advertisers[7, 33].

A problem with the malware on these devices is how hard it is to track down. Just like advertisers using malware to have other devices spam SMS advertisements makes it hard to track down the original source of the spam, malware can hide its activities. It can be done by going through other apps [9] or it can be responsible for downloading the apps actually performing the malicious behavior.[33] The malware DroidDreamLight would not be responsible for the information stealing of the devices. Instead, it would connect to an outside repository and download other applications. When the user realized there was malware on the device, and tracked it down to these new apps, he or should could remove them. When this happens, DroidDreamLight would re-download them and the process started over again.

2.1.3 Built In Security

The vendors of smartphone operating systems have included some built in security for prevention of misuse of the devices. They offer a marketplace for any apps for the devices and include some form of permission-based access for the software once installed.

2.1.3.1 Application Markets

The smartphone OS vendor encourage the users of their devices to have all of their purchases and downloads of various apps through a centralized market. This market gives the vendors more control over what applications they will allow users to install on the users' devices. With the control over the applications placed on the markets, the vendors have an easier time in finding possible malware and keeping it off of the market. Users who keep all of their apps updated through the market have the vendor's protection against malicious software[33].

Apple iOS has strict control over its market. The way they do this is by not allowing users to install applications from alternate sources. Users of iOS devices must use the market for any application they want. Short of a jailbreak on the device, the users have no option in installing applications that do not exist on the market. This allows Apple to have a strict review process. When a developer finishes an app, he or she posts it to the market. The app must then go through a strict review process. The goal of the process is to prevent as much malware as possible from being placed on the marketplace. The review policy does not allow for personal spyware to be on the market, but can allow grayware to

appear on the market. Apple will remove any grayware that it finds after the process and any spyware and malware that managed to slip past the review as soon as possible[14].

Android provides a market for their apps as well, although the review process is much more lenient when compared to Apple's. Google allows for users to install apps from alternate sources by enabling the Unknown Sources option. Because of this capability, Google's policy on reviewing apps prior to listing is minimal and they are only reviewed after they have been uploaded to the market. Google relies more on user input for tracking down malicious software. The Android security team does review applications based on user feedback and will remove malware after review. The security team will also remove any known malware remotely from a user's device[14].

2.1.3.2 Application Permissions

In addition to having a centralized market for applications to exist, the vendors attempt to keep interaction of the apps among each other limited. The operating systems use a permission-based policy when determining what the applications are allowed to access. Whereas Apple has stricter policies regarding the review process for their market, Android has stricter policies for permission enforcement.

Apple iOS attempts to control the applications by having them run through their market. As a result the applications are installed and allowed to access whatever they were programmed to access without consent from the user (the consent comes in the fact that it passed the review process and is downloaded by the user). Any extra permissions that

iOS needs approval for, such as location and notifications, require user acceptance during runtime[1, 14].

Google's policy for Android is that each application must request permission to access any other part of the device or any other application running on the device. When an application is installed on an Android device, the user is prompted with all the permissions the app is requesting. The user has to approve these permissions, which are set when the app is installed cannot be changed during run-time[14].

The permissions are designed to let the user know ahead of time what controls the application needs. The user needs to approve the apps usage of these permissions before the app is installed. This falls in line with Google's philosophy of having the security be more user oriented. Google expects the users to report problems to their security teams. Apple's approach corresponds to having the market control all apps installed on the device with little user control. While having users agree to let apps have certain permissions does give more control, too many permissions can make certain apps unsafe even if it was unintentional[33].

2.2 How Malware Attacks Mobile Devices

Mobile devices are just as susceptible to vulnerabilities as any computer. Malware not only exists but continues to grow with more mobile devices being used[14]. Knowing that this malicious software exists is not enough to protect the devices, but knowing how these devices are attacked and exploited will help in building prevention from these attacks. Built in security such as markets and permission policies, are good to filter out most malware yet

is not sufficient to protect from all. Permissions help cut down on malware, but overused permissions can actually allow for more exploits to happen than intended[5, 9, 10]. This section outlines how these attacks are possible.

2.2.1 Application Collusion

With the focus on permission based security on mobile devices the user has a need to know what permissions are being used by the application at install time. With Android, when the application is installed, a list of permissions is given and the user needs to accept them. If the user does not accept the permissions then the application is not installed preventing any harmful application from running. This policy is based on the user being the one who determines which permissions are acceptable for an application to have. The user acceptance of the policy also means the user must predict what harm can be done based on the permissions. Google offers the ability for Android phones to be able to sync contacts between the user's phone and his or her Google account. To do this, the application has permission to the contact list of the device as well as to network communication. This is not necessarily bad but a user who sees that an application requires both could be attempting to steal his or her information[5, 21].

Collusion attacks capitalize on the user having to make guesses as to the intent of the application based on permissions alone. With Apple iOS not having the same permission policy as Google, it is up to the market security team to be able to detect these attacks. Google's policy once again resides with the user checking the permissions on install. Either

way, the attack works the same way with the end result being a different analyzer trying to find the malicious apps.

The idea of a collusion attack is to have two or more applications that independently have very few permissions and access to the device. However, when these applications work together they act as one application with all of the permissions together[21].

Figure 2.2 shows a basic collusion attack with two independent apps. Application A is a contact list manager application. This app only needs access to the Personal Information permission. It is this permission that houses the contacts list for the device. Application B is an app that requires some outside network access, such as a weather app. App B needs permissions set for Network Access. The user of the device downloads both of these applications independently of each other, sees that limited access each one is requesting, and agrees to install them. The collusion happens when both apps work with each other combining their permissions together. Once application A and B are installed, they are then allowed to communicate with each other so that it is possible for A to send B the contacts list, then have B send the contacts list out to a remote site.

With Android, each application resides in its own Virtual Machine (VM) called a sandbox. These sandboxes separate the application process from each other. This means each process can be completely isolated but may still need to communicate. The Android Middleware is a middle layer which handles this communication by enforcing Mandatory Access Control[9]. In general an application needs to have appropriate permission to access different parts of the OS. Each application is made up of components and these components can be public or private. Private means that only other components within the same

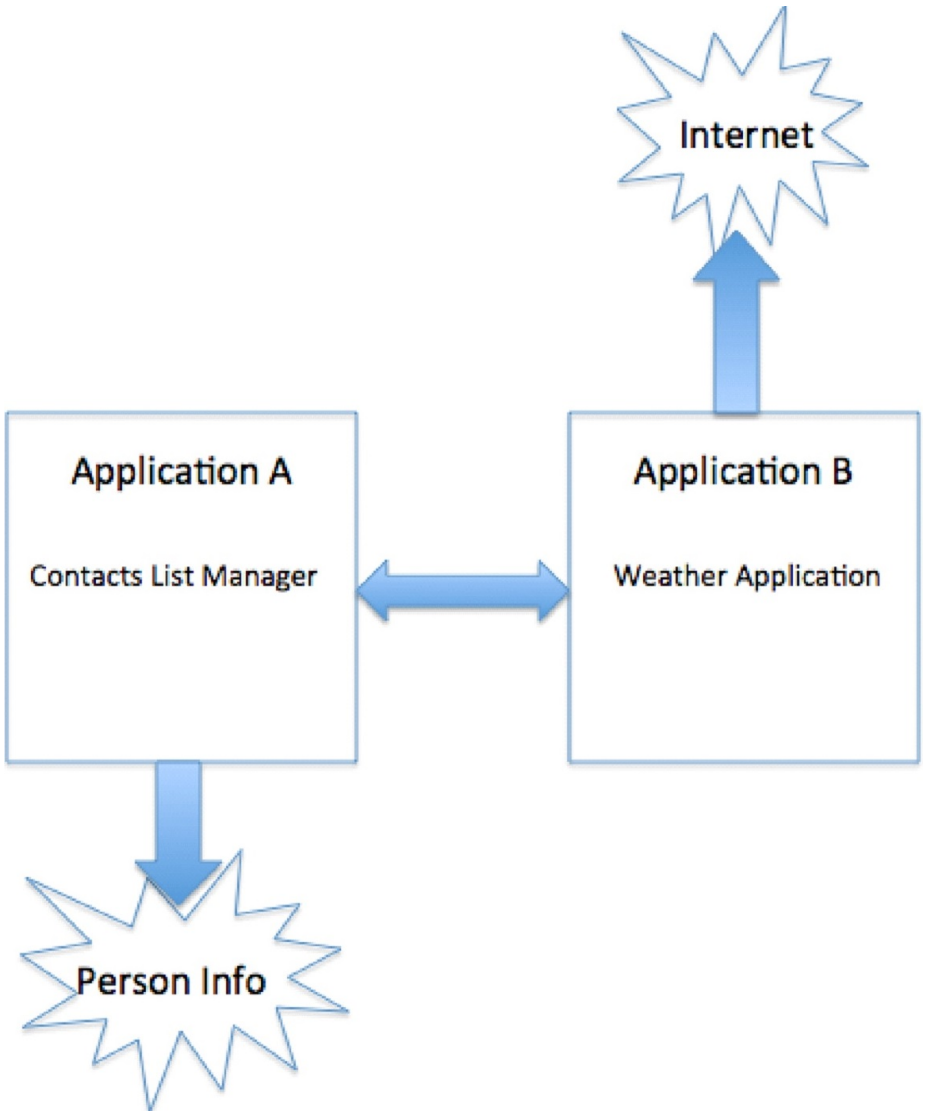

Figure 2.2

Two App Collusion Attack

application can access that component. Public meaning components of outside application can access these components. While permissions need to be explicitly set, the components access points (public or private) are not explicitly enforced. As such the OS defaults to public so that applications will not be completely shut off from each other[11].

Application A, contact manager, has public components to send information to other applications. Application B, weather, has public components to receive outside information from other apps. With this channel now open, the two applications work together to retrieve a user's contact list and send it out to some outside source. This attack is hard to track because individually each application have very limited access to the phone and did not seem like a threat.

2.2.2 Privilege Escalation

The privilege escalation attack is similar to a collusion attack. This still relies on a user installing malicious software based on limited permissions, yet this kind of attack is designed to work independently and exploit other applications with too many permissions.

The goal of a privilege escalation attack is to mask what the malicious app is doing by using other applications to do its work. As mentioned above, permission to access device and OS components needs to be explicitly made. This is because of the way the components are designed. A user made application can set how he or she wants components to be accessed by making the components public or private. It is very easy to leave different components of a developer's application as public, especially if the developer is unsure at

first what activities he or she wants the application to be capable of. A privilege escalation attack exploits an open application that has access to some component of the system.

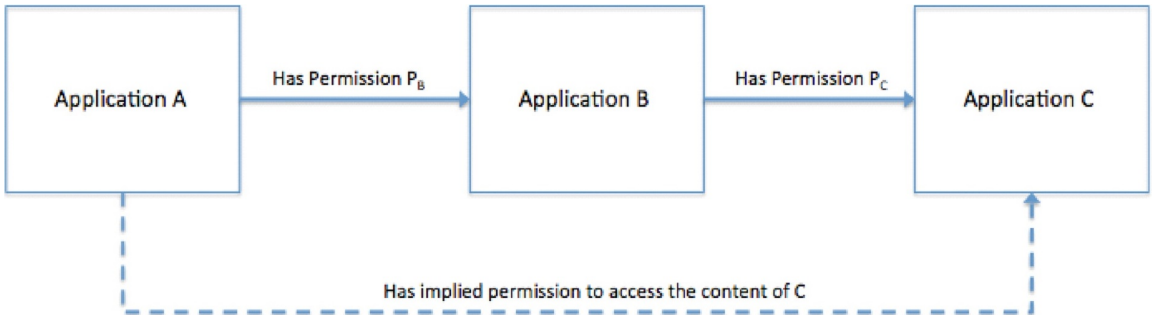

Figure 2.3

Privilege Escalation Attack

In this kind of attack, there are three applications installed on the device. Application A is the malicious app, which will be doing the attack. Application B is a non-malicious app that has components inside of it set as public, which are open to other applications. Application C is a component of the OS that has restricted personal information, such as the contact list. In the example application C has granted permission only to B while A does not have it. This means that A cannot communicate directly with C. App B, on the other hand, has open permissions to all of its components and so A can now communicate with it. Because B has open permissions and has access to the information contained in C, application A indirectly has the same information as well without needing the appropriate permissions.

This kind of attack masks the software performing the attack by giving it limited permissions, yet still using the permissions of other applications to achieve its goal. In the

above example app C is the contacts list of the device. App A is not given permission to access the contacts list. This is limiting what the application actually can do, but it is able to communicate with app B that has all the information of the contacts list. This bypasses the need for A to actually have explicit permission to get the lists.

This attack and the collusion attack thrive on the misuse of permissions by applications and on the fact that users may not fully understand or read the permissions when the applications are installed. The permissions are put in place by the OS to ensure some security between the applications installed; yet, if not used properly can still allow for malware to not just harm the device and information but hide what it is doing making it that much harder to track down and prevent.

2.3 Current Security Attempts

As usage of smartphone devices grows, so do the attempts in exploitation of the devices. The increase in malware in these systems has resulted in different attempts to protect the devices from the new malware on the devices. These solutions are designed to protect the end user of the device. There are different ways to protect the user. The user can be educated as to understand the permissions of applications, and a device can have its processes monitored from currently running malware. The work done in this area takes on the role of protection after the discovery of vulnerabilities and does not focus on prevention. This section outlines the security policies in development.

2.3.1 Behavior Based Analysis and Security

Malicious applications run on the devices the same way as any other application. The application needs to have permissions set to access components on the device, runs in its own sandbox, and communicates through the Android Middleware like other applications. It is once these applications are installed and running that the device becomes compromised. Once the application is running analysis of its process can reveal the data being collected by the application. Through the collection of permissions of the application and specific data being collected a security team can determine if the application seems to be performing un-documented procedures. This is done with the collection of data during install time and during runtime.[15, 29, 2]

Banuri et al. researched the idea of a runtime policy enforcer. This framework built is designed to analyze an application during runtime and enforce the permission constraints placed on the application. It starts by having the researches develop policies they call a harmful sequence of permissions. An example of this is an application has the Internet Access permission and later, through the interaction of other apps, gains the Record Audio permission.[2] This sequence of permissions is labeled as a harmful sequence and a policy is developed to handle it. The framework takes these policies and stores them in a Policy Repository for reference. When an application is installed, the framework stores a copy of the permissions from the Manifest.xml file in a permission repository. As the application runs, the permission usage is kept track of and is recorded along side the permissions in the permission repository.

With both repositories filled, the framework begins analysis of the application during runtime. As already specified, the application will make requests based on its permissions and this total is incremented in the permission repository. The framework monitors permission usage to check against any of the harmful permission policies in place. Every time a chain of permissions is established the framework must check this new chain against one of its policies. The checking is done during runtime and when an application breaks any of the framework's policies the application can be isolated from the processes with an evaluation returned to the user. The ultimate goal is for the framework to enforce permission constraints.

Other attempts at mobile security focus on standard malware detection techniques for desktops.[29] These solutions attempt to handle the malicious behavior detection and classification that [2] could use to relieve the user made policy analysis. Shabtai et al. attempts to solve the problem of classifying mobile malware with a framework they call Andromaly. The framework is designed to monitor application processes and attempt to classify malicious behavior

Andromaly is a behavior-based detection framework. The framework uses machine-learning techniques [29] to monitor process communication and decide what is malicious and what is not. The researchers monitor the processes at runtime looking at resource consumption such as battery usage, packets sent / received, CPU consumption, and number of running processes. By monitoring these resources the framework can determine which applications are safe and which are not.

During runtime, Andromaly monitors each application's sandbox for the resource consumption. Over time it begins to build metrics for each application on the resource usage. By using machine learning algorithms, the framework is able to determine the normal and abnormal usage of certain resources. Andromaly will learn what is normal battery usage and what is high battery usage. Each metric is sent to what they call a Processor, which is the main analysis unit. This unit contains rule-based anomaly detection and makes a classification of the particular application. Once the processor is done analyzing the app, it is then sent to the Threat Weighting Unit, which gives the device an infection level. This infection level is compared to the minimum and maximum thresholds for an infected system. The framework keeps the infection level stored and as more application data comes in, the additional data along with the infection level are sent back to the processor to refine the infection level.

Andromaly is useful in helping to classify the applications as malware by removing a large part of human analysis. The whole process takes time and running on a single device can be resource heavy. The framework monitors individual devices with no automatic way of corroborating the information. A similar behavior-based malware detection framework is called Crowdroid developed by Burguera et al[7].

Crowdroid monitors the application behavior at runtime as well. It monitors the Linux Kernel system calls and sends all the calls out to a central server. Various devices run the framework locally. When the framework detects the Linux Kernel calls, it sends the data out to a centralized server. This allows many different devices to be running at the same

time to give the central server a large dataset. The framework is crowdsourcing the work to get large amounts of data in a short amount of time[7].

The researchers [7] use clustering techniques to partition the raw data into different call vectors. The call vector is a string of numbers that represents a count of each specific kernel call being made. The central server parses out the data and compares it to the system calls and is able to classify the application based on the system calls. By taking samples of applications from many different devices, the framework is able to determine which ones are making suspicious calls and are expressing suspicious behavior. From there the central server can classify each application as benign or malware. By analyzing malware on the device by monitoring the behavior during runtime, a collection of malware can be established to allow for removal from the market. The more runtime analysis used, the more malicious calls can be classified to judge newer applications being developed.

2.3.2 Policy Based Security

Malware detection is useful when attempting to find problems with newer applications as they appear on the market. By knowing how these applications attack the system, security teams are able to pinpoint the exploits in other applications. The heart of the security teams for Apple's iOS market, and Google's Android market rely on these techniques in finding existing malware to remove from the market. These pieces of malware keep showing up on the markets because of how easy it is for users to install them.

A problem with mobile device security is based on the users to protect their devices. The end user of the device does not have the education or experience to know if an ap-

plication seems suspicious. With the market it is easy for a user to access the app he or she wants and download it to the device. With Android's policy of having the permissions shown to the user before installation, and having the user agree to them, the assumption is made that the user understands what these permissions are and how the applications will use them. Most users can see no problem in these open permissions and will install that application anyway[5].

While the inclusion of a strict permission policy builds in security for the device, a user who grants an app access to these permissions has agreed to any exploit the application may be using. One way to fix this problem is to educate users to the dangers of applications on the devices. Researchers have developed policies with guidelines built for users to follow to ensure more secure mobile devices.

The National Institute of Standards and Technology (NIST) published a document in 2008 called Guidelines on Cell Phone and PDA Security[19]. The goal of the document is to outline a set of guidelines that users of mobile devices can use to greatly increase their security. NIST determined that people tend to violate these guidelines because they do not think about them on a big scale. The give a list of User-Oriented Measures:

- Maintain Physical Control
- Enable User Authentication
- Backup Data
- Reduce Data Exposure
- Shun Questionable Actions
- Curb Wireless Interfaces
- Deactivate Compromised Devices

- Minimize Functionality
- Add Prevention and Detection Software

As with any computer, the security of the device is gone as soon as an attacker has physical access to the device. Maintaining physical control at all times is important for any device. Loss of the device or theft not only makes the user lose the money invested in the device, but now all of the personal data of the device is in someone else's hands. In the event of lending the device to another user, the owner should never leave the device unattended because then the owner will not know what could have been placed on the device[19].

Enabling user authentication allows for the device to be locked while not in use. When a device is locked, it ensures that only the person who has the appropriate credentials, such as a password, can unlock the device. In addition to being able to lock the device, user authentication is crucial for installation of applications[19]. With Android, the user must agree to the permissions before installation. But if the device has been Rooted at any point, other applications can do installations without the user's consent.

Backup all data on the device. Using the device as the only repository for any of the user's data is unsafe. The device can be lost, stolen, or damaged. If any of these events take place then the user's data will be lost and not recoverable[19].

Users can reduce data exposure by avoiding using the device as a storage place for sensitive information such as personal contact information or financial records. While authentication mechanisms can help protect the device from being accessed, as soon as the

authentication breaks all the information is exposed to the malicious user. If any sensitive information does exist on the device it should be encrypted[19].

Users should shun questionable actions by only allowing processes to occur that the user is expecting. The user should not respond to unknown text messages nor click any links sent to him or her. This could be a way of getting the user to download a malicious app remotely. The user should also be aware of network connections and not allow any network or Bluetooth connections to happen that the user was not expecting[19].

Users should curb wireless interfaces by disabling all network connections when not in use. In the case of a smartphone, all wifi, Bluetooth, and mobile network access should be turned off when not specifically using them. Staying offline reduces the chances of malicious connections from reaching the device[19].

Users should deactivate all compromised devices as soon as they have been compromised. This is usually done if the device is lost. By deactivating the lost device, anyone who finds it will be unable to use the device for personal gain. The device can also be compromised while still in the user's possession. It should still be deactivated immediately until the malicious software can be removed. While compromised any communication the device has can cause harm[19].

Users should minimize functionality by limiting the features of the device. The more features and capabilities given to the device, the greater the risk in exploitation. Similarly to deactivating all network connections, the device should not be using any capabilities unless specifically being used at the time. More capabilities means more components to attack[19].

All devices should have prevention and detection software. Device encryption, antivirus software, firewalls, etc. all can be used to limit the chance of getting malware installed on the device to keep it safe[19].

The document is intended to give a list of rules for users to follow for protecting their mobile devices. NIST then goes on to encourage companies to use the rules to build company-wide policies for employees to use. Alan Goode performed a study in which he would ask companies questions about their mobile device policies. He discovered that 65% of companies allow employee-owned devices to be used for business purposes as well as personal. In addition, 46% of respondents do not have a document outlining security policy for mobile devices[16].

The idea of using one's own mobile device to conduct business remotely is a growing one. With the increase in usage of these devices for business use companies need to educate their employees to the dangers around exploitation of the devices. Goode proposes that companies need to deploy policies their employees can follow to help reduce the vulnerability of their devices. He suggests forming policies around encryption, backup, and authentication mechanisms[16].

The goal of policy driven security is to prevent the attacks from happening by giving the user the tools and education he or she needs. Since most exploits come from user misuse of the device, educating the users, can help prevent the malware from being installed on the device before anything else can happen.

2.3.3 Permission Based Security

The Android OS relies on a strict permission policy to secure their applications. The whole concept revolves around the application needing to request explicit permission in order to access components of the device.[1] If an application needs Internet connection, it needs permission to the Network Connections privilege. Yet it is the permissions that can be exploited to gain access to device components without privilege access[9, 30].

Bugiel et al. developed POSTER, a framework designed to monitor the communication of applications against their permissions. This framework works on runtime like most of the behavior-based security solutions. While POSTER is monitoring the communications of the applications, it is doing so in context of the permissions of the application.

The framework monitors the Inter-Process Communication (IPC) and intercepts calls to the reference monitor. The request is analyzed and compiled into a system security policy. This policy contains the permissions needed for the communication to happen. The IPC is continuously monitored and the information sent back is also recorded in a system security policy. The system policies are checked against the manifest.xml file of the application to check permissions. The permissions are checked to see if they will grant the access the application is looking for. Upon determination that the application is going beyond its permissions, the communication can be terminated and the application removed[24].

The goal of the framework is to prevent privilege escalation attacks by monitoring the permissions of an application. Developers have a tendency to leave permissions open as well as add permissions as they develop. As a developer programs his or her application for Android, he or she has a tendency to make calls to APIs and instantiating new permissions

each time. This creates a duplication of permissions in apps.[32] An application can have a permission secured and not left open; however, if accidently duplicated, the duplicate will not have the same restrictions placed on it. Table 2.2 displays the summary of applications with duplicate permissions. The table was taken from [32].

Vidas et al. developed a plugin for the Eclipse IDE to help check the permissions of developers during development. As the developer is coding the applications, the plugin will monitor the usage of the permissions. Any time the plugin finds a duplicate permission being called it notifies the developer to the duplicates and allows the developer to make the appropriate change[32].

The problem with extraneous permissions is that it can leave holes open in the application. If the application has a duplicate Network Access permission, then the OS will default to the one that has fewer restrictions on it. This permission may not be the one used, but it is still opened. The opened permission can allow for a malicious user to exploit the vulnerability with another application. This is how the device becomes vulnerable to privilege escalation attacks.

The permissions set by the Android OS creates a level of security in the applications. The permissions must be explicitly set to allow an application to communicate outside of its own sandbox. The more permissions an application has set, the more open the application is. Misuse of the permissions allows for the software to become vulnerable to outside attacks. The goal of permission security is to reinforce the idea of application security by using only the permissions the application needs.

Table 2.2

Applications with Duplicate Permissions by Market Category

Market Category	Total Apps	With Duplicates
Arcade and Action	1344	17
Books and Reference	1452	24
Brain and Puzzle	1352	14
Business	1092	38
Cards and Casino	842	85
Casual	966	4
Comics	836	11
Communication	1311	77
Education	1305	21
Entertainment	1522	40
Finance	1354	44
Health and Fitness	1258	36
Libraries and Demo	1156	21
Lifestyle	1489	48
Live Wallpaper	537	14
Media and Video	1360	49
Medical	527	2
Music and Video	1124	89
News and Magazine	1419	63
Personalization	1342	54
Photography	1165	283
Productivity	1319	54
Racing	216	76
Shopping	1155	46
Social	1296	41
Sports	1433	23
Sports Games	365	71
Tools	690	27
Transportation	454	8
Travel and Local	1473	35
Weather	342	4
Widgets	1395	64

2.4 Software Development Life Cycle

The concept of the Software Development Life Cycle (SDLC) was developed to streamline the designing and building of large software packages deployed by organizations. As the use of computers for business grew, so did the complexity of the software running the businesses. With less complex systems, it is possible for the developers to begin implementation, guiding the evolution of the system design. When the systems become more complex it becomes harder to build without a solid design[13, 25, 31, 28, 18].

With the SDLC organizations break the software development into blocks or phases. These phases are structured guidelines that are designed to make software development easier and more cost efficient. Each phase is a piece of the overall system being built and is taken one at a time. The ultimate goal is that if each phase is completed properly then there should be little to no rebuilding of the system in the end.

2.4.1 Traditional SDLC

The software life cycle is used to model the phases that software should go through during development. Having a detailed list of goals in mind before the actually programming begins makes it easier to know what functionality to build into the system. Each phase is designed to build a particular part of the overall system and must be completed before moving on to the next phase. Figure 2.4, taken from [25], shows the general life cycle.

Ragunath et. al [25] show a general model for a software life cycle as shown in figure 2.4. This model shows the general flow the developers will follow when building a new software system. Each phase works on sequentially until completion. The first step is to

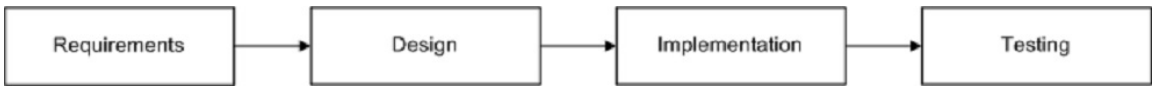

Figure 2.4

General Life Cycle

work within the first phase of the life cycle and nothing else for the system is worked on until this phase is completed. Once the first phase is complete, the developers move on to the second phase. The above model is separated into four phases: Requirements, Design, Implementation, and Testing.

The Requirements phase is where the developers of the system meet with the client requesting the system being built and decide what functionality the system requires. During this time, the functionality is determined ahead of time and not as needed during implantation.

Next is Design. Once the functionality is determined, the developers move to a phase where the software is designed before any code is written. It is this phase where the functionality is translated to components of the system and the messages the components will use to communicate. This phase is used so that the high level design is used to guide the implantation, not the other way.

Once design is complete, the developers move to Implementation. This phase is when the software is actually programmed. All of the decisions made from the design are translated to code and the system is built.

The last phase is Testing. Once the system is completely built, the software moves into a testing phase where any errors and bugs are discovered and fixed. This phase polishes the product before final delivery.

At each one of these phases, the developers are focused only on the phase they are currently working. When in the Requirements phase, no time should be spent designing the system. The ultimate goal of this process is to use each of the phases to guide the completion of the next phases. If the developers are successful during the requirements phase in gathering all requirements, these requirements can then be used to guide the design of the overall system. With the requirements finalized, the developers have no reason to constantly get a requirement and alter the design as newer requirements arrive. At each new phase, all previous phases are completed and should have no need to be revisited. When the developers are finished designing the system, then all of the requirements will be accounted for in the design and implementation would have no need build off of the requirements. This process continues until the system is completed.

The general model has been used as the basis of developing other software life cycle models. The process is simple as it clearly defines a phase and the progression through the phases. The general model is usually represented a second way known as the Waterfall Model[25].

The concept of the Waterfall model, shown in Figure 2.5 taken from [25], is to show the different phases of the life cycle and at each step the model ”steps down” falling like a waterfall. The idea behind the Waterfall is the same idea that the general model was displaying. During the software development process the developers go through different

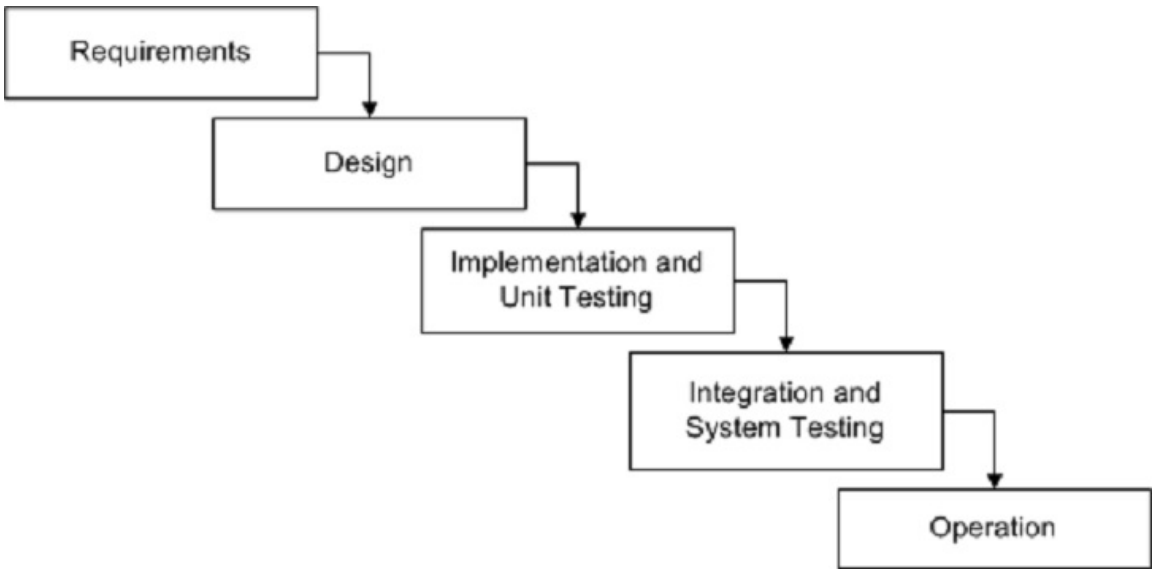

Figure 2.5

Waterfall Lifecycle Model

phases. Once each phase is completed, the developers step into the next phase of development. While the phases can changed based on a particular client's needs, the idea of how to use the phases in the Waterfall remains the same.

Although considered a flawed model [25], the Waterfall model is just an extension of the general approach to SDLC and is simple to understand. The simplicity of the model is why it is used to teach the concept of SDLCs. Once the developers understand this concept changes can be made to the model to fit their particular needs. Even though flawed, this general approach is useful in guiding the way developers go about building a system and extensions have been made to help improve on this model[28].

The flaw with the Waterfall and the general model of software development is that it does not allow for any backtracking through the phases if requirements or design decisions

change. Some adaptations have been made to account for this with the Iterative Feedback Waterfall model. This model does allow for the developers in a phase to step back to the previous phase. This still only allows for one step at a time and must be completed before stepping back again, or moving forward. The rigid flow of these general models makes changes and adaptations harder to implement and has led to the design of more flexible models.

2.4.2 Flexible SDLC

It is common for decisions to change during the development cycle. If during development the customer decides to add some additional requirements, then the development team must be able to adapt the design without starting over. The traditional approach to SDLC had the developers complete a phase and move onto the next phase with no easy way to go back. That approach works if the system does not need to be changed in the middle of any particular phase. Flexible models of SDLC do not require an entire phase to be 100% completed before moving onto the next step. They allow for the developers to reevaluate the process in each phase and determine ahead of time if changes will need to be made.

The look-ahead component of these models is what allows them to be flexible to changes in the system during development. The Spiral Model developed in 1986 by Boehm[28] gives developers a guideline to developing the system with reevaluation at each phase.

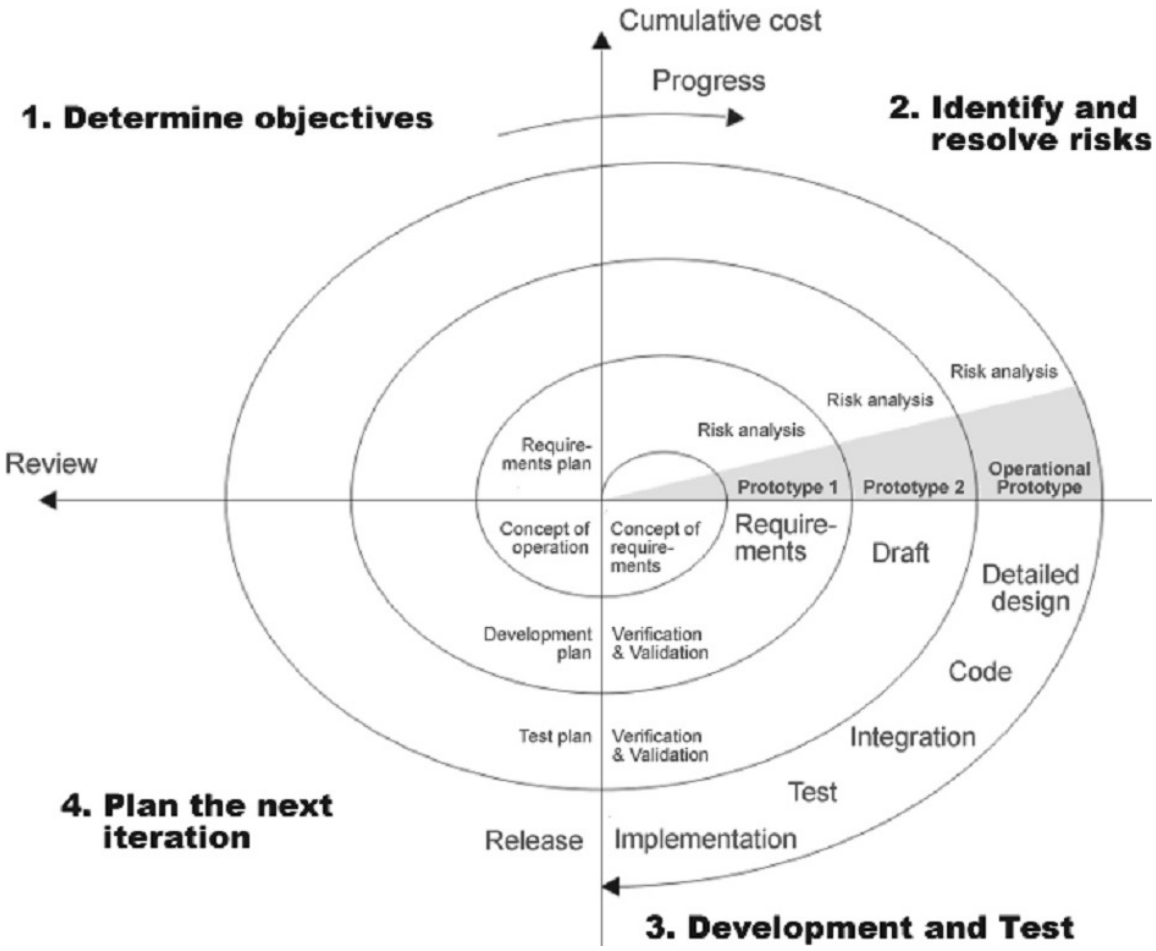

Figure 2.6

Spiral Model

The Spiral model was designed to allow for greater risk assessment during development. Like the traditional model the Spiral model still breaks development into phases, but it is not concerned with completion of a phase before moving on with no real plan for going backwards. The Spiral model allows for the developers to retrieve feedback while in a phase to allow for adjustments to be made during the lifecycle.

In figure 2.6 taken from [28], the starting point is requirements planning like the traditional model. The requirements are gathered then are submitted to a risk analysis subphase. The risk analysis is designed to look at the requirements and begin prototyping a design to match them. This prototype begins the design phase while still gathering all of the requirements. The prototype is then used to elicit feedback from the customer to insure that the integrity of the requirements is being maintained in the design. If any changes need to be made, the process loops back around again gradually moving into the other phases of the lifecycle.

With the Spiral model, the ability to combine multiple phases (meaning what the general model would call phases) together gives the developers a way to look ahead and determine if changes need to be made before they happen. At each stage the developers prototype the current incarnation of the system that is used to gain feedback from the customer. This prototyping gives greater flexibility in the overall designing and building of a system. This concept can be used to modify existing models by adding in a circular transition between phases during development.

The Rapid Application Development (RAD) model is a methodology designed by James Martin in 1991. [28] As seen in figure 2.7, the RAD model takes the general model

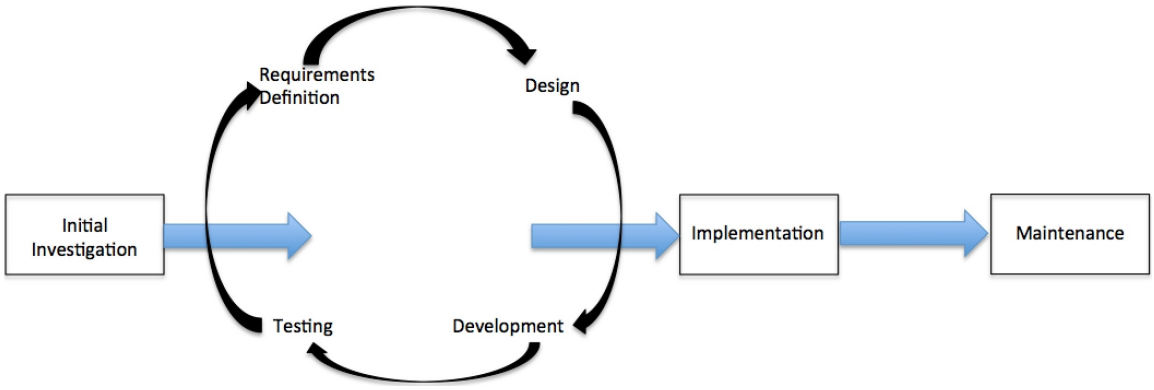

Figure 2.7

Rapid Application Development

from traditional development and adds a cyclical transition between phases that allows for incremental development. In RAD the requirements are defined, a design is made, the design is developed and created, and the system is tested. The results of the test are used to judge the system and, going back to requirements definition, makes adjustments. The key for this model is that each phase is not iterated through only once. Small prototypes are made and used to redesign parts of the system. With each pass through the cycle, more components of the system are added. By prototyping each component, the developers are able to know that these components work when it comes to adding more to it.

The goal of having SDLC models is to reduce the overall cost by having all parts of the system designed before implementation of the system. By not having to backtrack through development, the developers are able to save time making the system. The key to having a flexible model is the ability to cycle back to other phases without losing work. These flexible models make great use of prototyping to gain feedback. The prototypes are built and given back to the customer for review. Once the prototype is approved it can then be

used to build upon. The constant building on earlier prototypes is how models like the Spiral Model are able to keep development time low while still allowing the flexibility of change during development.

2.4.3 Summary of Reviewed Works

Many attempts at Android (and all smartphone OSs in general) security relies on a tool or process to protect the device after applications have been installed. Malicious software is written to exploit vulnerabilities in poorly design applications. Current research in Android security deal with tools designed to protect from known vulnerabilities by isolating or completely removing vulnerable applications from the device.

Software development life cycles exist to assist in the design and building of software applications. These models and methodologies result in less time spent redesigning code during and after testing. With current methods of security focusing on protecting devices after installation, research in preventing poor design from the beginning is sparse. The proposed model utilizes SDLC methodologies to prevent poor design and open permissions from final Android applications as detailed in chapter 3.

CHAPTER 3

SECURE ANDROID DEVELOPMENT MODEL

The Secure Android Development Model is designed to give the steps and tools necessary for developers to implicitly build security into their applications. This chapter outlines the procedures leading to the development of the model as well as the model itself. The chapter is organized as follows into three sections. 3.1 Inconsistent Permission Usage details how permissions work and why they can offer security holes. 3.2 Permissions Defining explains how developers can get a list of permissions and organize them into priorities. 3.3 Secure Android Development Model shows the model itself with a detailed explanation of each step.

3.1 Inconsistent Permission Usage

When a developer makes an application, part of the design involves knowing what actions the application will be making that require permission to access. The application needs to be explicitly permitted to access the components within the system. The same security principle applies to a user-made app trying to access other user-made apps. Each application is made up of smaller components. The components are smaller pieces of software that interact with each other for functionality (object-oriented style of development). For an application to access components from another application there needs to exist per-

missions for this interaction. The developer of the application needs to decide whether the components need to be public or private. A public component is open to other applications while a private component is only available to other components within the same application.

Public components are reachable by outside applications and will require permission to access. The developer designs a user-made permission that other applications must know about and request in order to communicate with it. In the event that a developer assigns a component as public without specifying a permission, the OS allows any application to access the component. A developer can often overlook this feature in the quick development of the application[11]. If a component is not made public or private, the OS can decide during compilation what the component should be. This can lead to components that should be private being public without any access permissions assigned to them.

A developer can also request too many permissions from the OS during development if unsure what the end result of the application will be. By opening up many different permissions, the application is exposing itself to more access points. For example, an application can be given access to the Bluetooth capability of the device. As soon as a malicious developer discovers an exploit in the Bluetooth application native to the device, any application that connects to it is exposed. This is unavoidable if the application relies on the Bluetooth of the device. It becomes a problem if an application is developed with this permission given and not being used. Developers can leave permissions open even if there is no appropriate API calls being made[32].

Development can be a quick process where decisions are made spontaneously. When the decisions are made spontaneously there is no revision made to ensure that every component left open is being used by teh system. Any open connection leaves the application vulnerable to other applications. The best practice is to have the minimum permissions needed to run the application being called.

3.2 Permissions Defining

The first process to software development is to define the requirements. The requirements of the system are the actions that the software must perform and capabilities the software must have. Accessing an online database as well has having a simple GUI are both examples of requirements. With Android applications, the actions performed by the software still need to access permissions from the system. The first step in developing an android application is defining what actions the software will take and what permissions need to be requested for these actions.

During the development process, it is easy to add more permissions to the Manifest.xml file while making API calls in the source code. As mentioned earlier developing in this style can lead to permissions being opened and left opened after the associated API call is no longer used. It also leads to duplicate permissions being opened without firmly controlling both. The developer only maintains control of one of the duplicates at a time[32].

The Android API defines two classifications of permissions for the manifest: Manifest.permission and Manifest.permission_group. The idea of having a permission-group is to wrap similar permissions together into one. An example of this is the permission-group

BLUETOOTH_NETWORK. This permission is in reality a collection of other permissions. In this example BLUETOOTH_NETWORK contains the following permissions: BLUETOOTH, BLUETOOTH_ADMIN, and BLUETOOTH_PRIVILEGED. The use of the group is to ease application building for the developer. The developer has to only make one permission call rather than three in his or her manifest file. The disadvantage to using groups is that all permissions in the associated group are open, even if they are not being used. The first way to prevent overuse of permissions is to not utilize permission-groups.

In addition to adding system permissions as needed, the developer can choose to not specify any permissions associated with the application's components. Any component left open as public must have user made permissions associated with it or else any application can access the component. The developer needs to plan ahead of time what actions the application will take when working with other applications and set its own permissions before development starts.

To determine what permissions the application needs the developer must decide the requirements of the system. From the requirements the developer prioritizes them based on preference[20]. When determining preference the developer asks himself or herself, "What requirements are necessary for the application to function, and what requirements are nice to have but not crucial?" The requirements can then be broken down into the permissions needed for it to run.

The requirement in figure 3.1 is the application needs to use Bluetooth capabilities. From here we find three permissions associated with Bluetooth. BLUETOOTH is used to allow applications to connect to and communicate with paired devices. BLUETOOTH_AD-

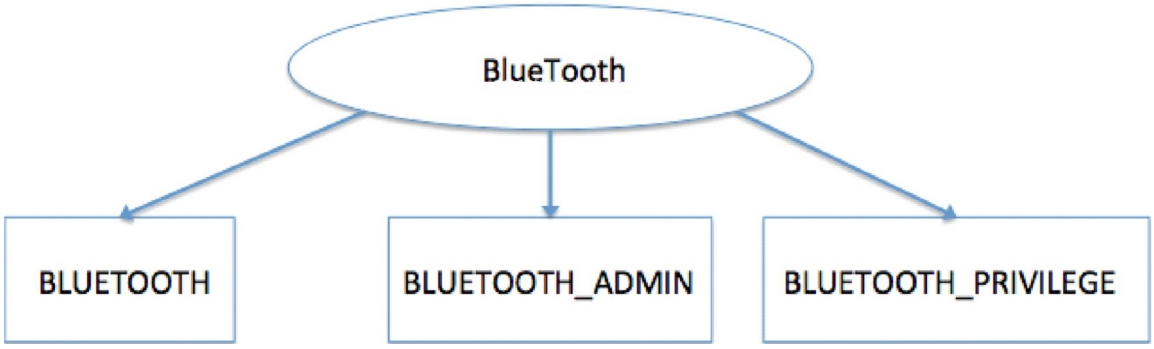

Figure 3.1

Breakdown of Bluetooth Access Requirement

MIN allows applications to discover and pair with Bluetooth devices. BLUETOOTH_PRIVILEGED is used to allow applications to pair with devices without the user interacting with them. All of these permissions are used for a Bluetooth connection, but not all of them are needed. The application can function with just the BLUETOOTH privilege and the user can manually discover and pair devices outside of the application. Because the user can use the built-in functionality to pair with other Bluetooth devices, the only one that is needed for the app to work is the BLUETOOTH permission. The others are nice to have but not crucial.

As mentioned earlier, there exists a permission-group that encapsulates all three of these Bluetooth permissions. A developer can call the group permission and not have to call three separate individual ones. Using the permission-group can be useful if the developer needs to use all three of the Bluetooth permissions. Unless it all three of these permissions are needed, using the permission-group for Bluetooth will result in open permissions not being utilized by the application.

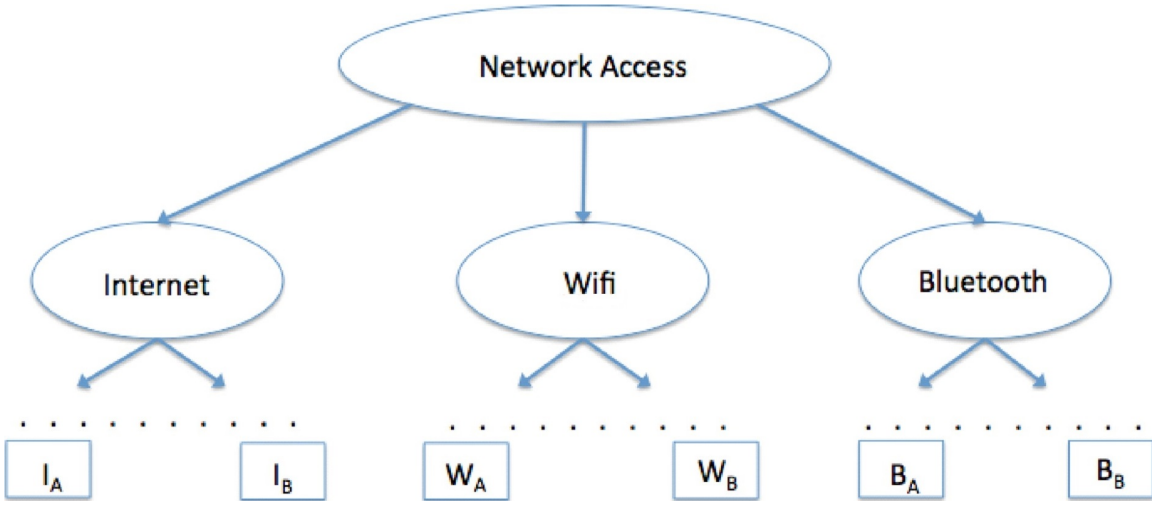

Figure 3.2

Breakdown of More General Requirement

A requirement can also be a more generalized task without specific permissions tied to it. Figure 3.2 shows the requirement of the device needing network access to remote sources. This is broken down into three other tasks specifically as needing Internet access, wifi information, or Bluetooth connectivity. Each requirement is further broken down into sub components and on until a list of permissions is established. The goal of this process is to establish the permissions a developer will need for the application.

3.3 Secure Android Development Model

With the problems permissions can present to a developer, having a process to follow to ensure minimum permission usage (both system permission request, and user made permission) will result in cleaner coding with fewer outside openings. The more activities a single application performs, the more vectors of attack are open to it from privilege escalation and collusion attacks. The Secure Android Development Model (SADM) is a

development lifecycle model that focuses on creating an application that achieves all the intended functionality with minimal permissions.

The SADM consists of 4 main development phases and a delivery phase. The phases all contain steps for developing the permissions needed for the application to work. Each phase with the exception of the Build Application phase all contain sub-phases to further isolate the activity into smaller but still important steps. The model follows a standard Spiral Model of software development. This model focuses mostly on generating the Manifest.xml file for the permission setting. Figure 3.3 illustrates the steps of SADM.

3.3.1 Permission Gathering

The Permission Gathering phase is the first phase in the process. This phase is similar to a requirements gathering phase as the goal is to determine before implementation what kinds of permissions are going to be needed. The phase is broken into three sub-phases: System Requirements, Component Requirements, and Permission Listing.

In the System Requirements sub-phase the developer is attempting to determine all of the permissions the application will need to request from the operating system. The developer plans for every interaction with the system the application will be wanting and listing these as requirements for the system. Once the requirements are determined the developer follows the process of decomposing the individual requirements into the permissions needed. The decomposition will break each requirement into sub-requirements, which helps in further refining the requirements, and will yield the permissions. Figure 3.4 illustrates the process of going from a requirement to a set of permissions.

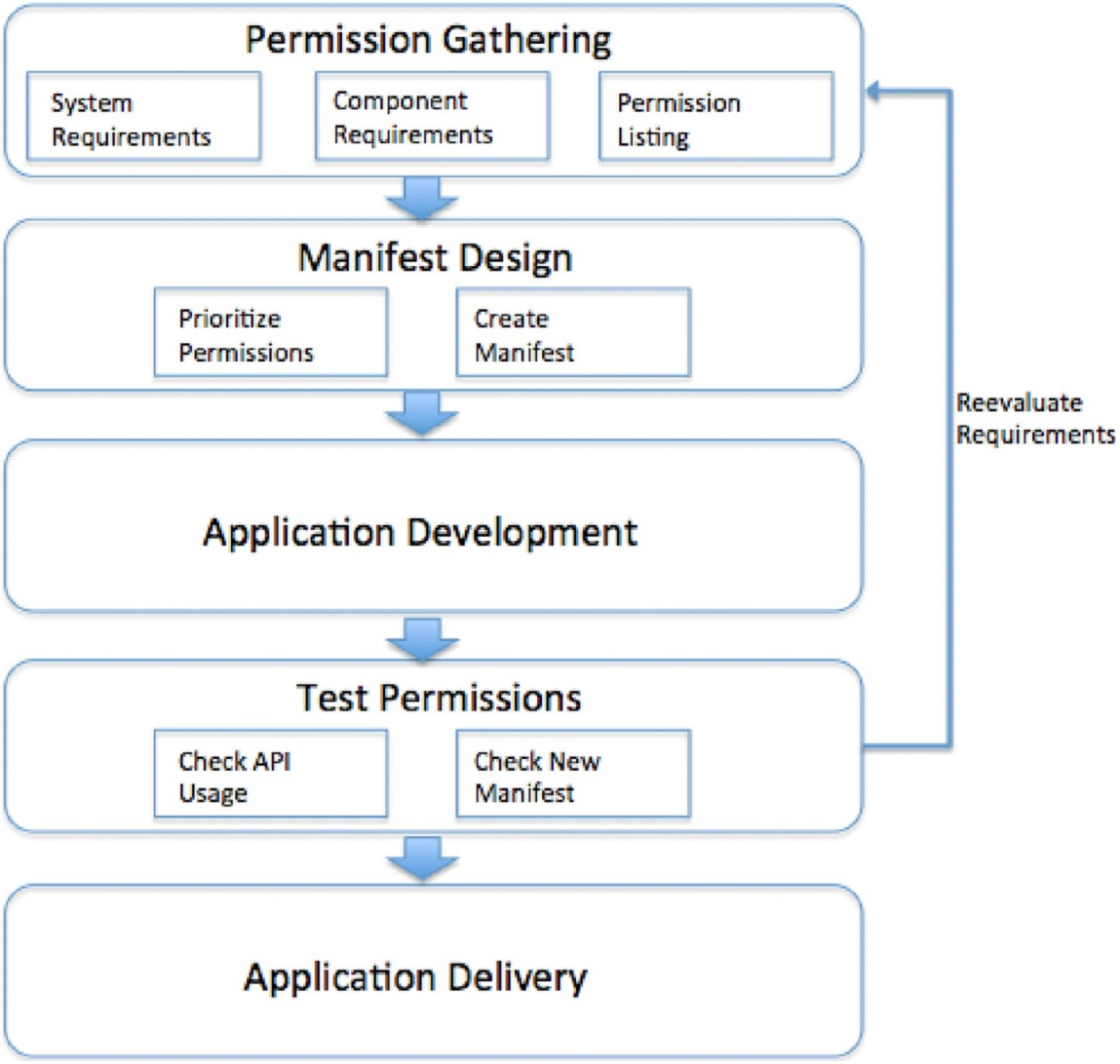

Figure 3.3

Secure Android Development Model

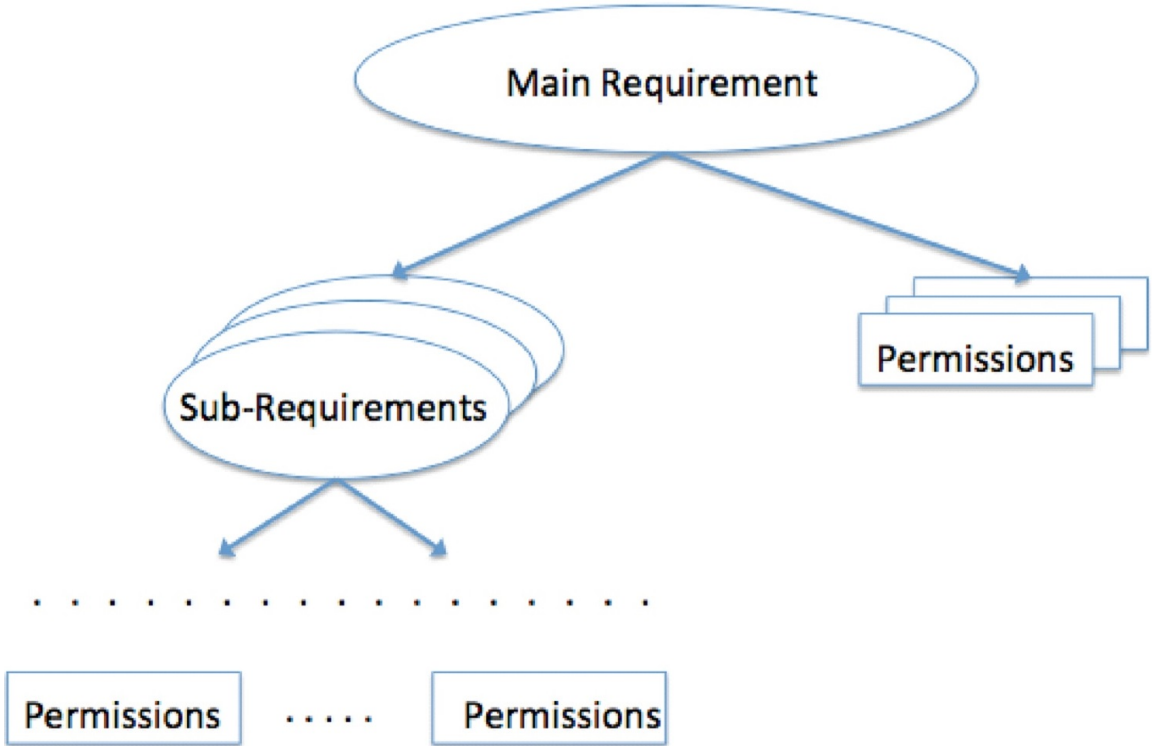

Figure 3.4

Decomposition of Requirements to Permissions

The Component Requirements sub-phase follows the same idea as the previous sub-phase except that it is for any actions the applications will make regarding receiving and sending information to other applications. The components of each application are what run it. For each application, if there are components that need to send data to other applications, then the component needs to be made public and a user made permission needs to be implemented. If the application needs to receive data from other application it needs to know what the user defined permission is for the app and request it. The end result will be a list of components needing to be public with any of the user defined permissions the application will require for others as well as any extra permissions the application needs from other applications.

The last sub-phase is the determining of permissions. Once all of the requirements have been determined and the decomposition leading to the list of permissions has been made, the developer then organizes the list of permissions. This final step is to ensure that any permission that meets the needs of the requirements is included to be prioritized.

3.3.2 Manifest Design

After the requirements for permissions are established, the developer then moves to the Manifest Design phase. Before any implementation is started, the Manifest.xml needs to be constructed to enforce the permissions determined before development. This enforces the idea of not adding any new permissions during development and sticks to maintaining what was determined during the initial design. This phase is broken into two sub-phases: Prioritize Permissions and Create Manifest.

By the Prioritize Permissions sub-phase the developer has taken his or her initial requirements and extracted a list of permissions. The list of permissions is still unrefined and needs to be prioritized. The prioritization is to establish what permissions are necessary to function and which permissions are simply to add more functionality to the application. The developer progresses through each of the requirements, looks at every permission, and determines the ones vital for the application to function and which ones make using the application easier. The goal is to get a minimal list of permissions set.

Table 3.1

Statements About a Permission to Determine Its Priority

Statement Template	**Meaning of Statement**
need X	Permission X must be used to satisfy requirement.
X would be nice	Permission X makes functionality easier.
Y needs X	Permission Y needs Permission X to function.
never X	Permission X never needs to be used.
Y and X	Functionality requires both permissions Y and X.

The Create Manifest sub-phase is started once the permissions are prioritized. The developer takes the determined permissions and creates the Manifest.xml file for the application before development. The file is created and all of the permissions are put into it ahead of time. With the permissions being preset into the Manifest.xml file, the developer can use it to maintain integrity on the API calls for the permissions during development.

3.3.3 Application Development

Once the Manifest.xml file is created and all of the permissions determined, the user then can start programming the actual application. This phase is only concerned with the coding. The pre-generated Manifest.xml will be used to guide the development of the application by giving the developer the idea of what API calls will need to be made for the various permissions. No API call should be made that does not have a corresponding permission set in the file. Likewise, there should not exist a permission left open that does not also have a corresponding API call.

3.3.4 Test Permissions

After the application is built, it should be tested for conforming to the integrity of the model. Testing is needed before any application is released to the market. For this model, testing refers to the testing of the permissions against the original Manifest.xml. This phase will perform the checking with the goal of checking to see if any new permissions were added or changed during development. The phase is divided into two sub-phases: Check API Usage and Check New Manifest Against Old.

The Check API Usage sub-phase is designed to find inconsistencies between the API calls of the software with the Manifest.xml file. During development, decisions can be changed with respect to functionality and API calls made by the application. These changes result in permissions being set in the Manifest.xml but never used. This violates the minimum permission constraint the model enforces. This sub-phase is a manual process forcing the developer to check for an appropriate permission set for every API call.

The next sub-phase, Check New Manifest Against Old, is designed to check for integrity between the premade Manifest.xml and the resulting one. Similarly to changing decisions with API calls, permissions can be added or removed as design changes happen during development. This sub-phase checks the permissions listed in the new Manifest.xml file with the permissions set in the premade one.

The next step is to determine what process to take if there is inconsistency between the predetermined permissions and the ones set at the end. In the event that either of the two sub-phases result in inconsistent permissions, the priorities and requirements need to be reevaluated. The process loops back to the beginning Permission Gathering phase and repeats itself. This loop continues until a final permission set is equivalent to the one made before development.

3.3.5 Application Delivery

After the process has been completed (even if it took more than one iteration), the final step is to deliver the application. The result from following the previous phases is an application that contains a minimal number of permissions both set and requested in order for the application to function properly. While more permissions could make use of the application easier at times, by keeping the interaction of the app to a minimum the developer has reduced the risk of having his or her application compromised by other malicious software.

CHAPTER 4

EXPERIMENTAL DESIGN

In Chapter 3 a methodology was outlined for secure software development on the Android system. This chapter describes the experimental design and reports on the results of experimentation by the participants. The participants were given a set of requirements for an Android application to build. The participants used the SADM methodology to build the application and completed a survey upon completion. The results of the survey and the final permission list are used in conjunction to analyze the effectiveness of using the methodology for development. This chapter describes this experiment and presents the results.

4.1 Testing Plan

The hypothesis proposed in this dissertation is that using the proposed SADM methodology results in minimal usage of permissions for an application. As stated earlier, this dissertation defines minimal as the fewest number of permissions used to satisfy the requirements. To test this, the methodology needs to be used by software developers and have an application built. This section presents the application built by participants as part of the experiment testing the methodology.

4.1.1 Model Verification and Test Case Design

Verification of the model was performed by the researcher building an application based on existing apps. The application was built to allow for two devices to connect to each other allowing for message transmission and file sharing. One user needed to pick a file from his or her device and send it to a connect device. In addition to file sharing, the two devices will be able to send short text based messages to one another. Prior to building the application, the researcher began a survey of applications with similar capabilities from the app market. A total of 20 different applications were analyzed. These applications were a combination of file transferring and messaging applications and the distribution is showing in Table 4.1.

Table 4.1

Pre-Test Analysis Application Type

Application type	Number of Applications
File Transfer only	5
Messaging only	5
Messaging and File Transfer	10

The permissions of each application were checked against the usability of the application itself. If a permission was set by the application then the user should be able to utilize it. For example, a file transfer application the application needs to set the android.permission.READ_EXTERNAL_STORAGE and the android.permission.WRITE_EXTERNAL_STORAGE permissions. These permissions can be seen by the user in the form of finding a file to transfer and downloading a file from another user respectively.

Any permission set by the application that did not have some usable functionality by the user was recorded. Table 4.2 shows a list of unused permissions and how many applications had these permissions set.

Table 4.2

Permission Usage

Permission	File Transfer Only	Messaging Only	Combination
	Unused Permissions		
ACCOUNTS	4	5	9
MICROPHONE	3	2	10
PERSONAL_INFO	5	5	10
SOCIAL_INFO	3	5	7
	Used Permissions		
DISPLAY	4	5	10
NETWORK	5	5	10
SCREENLOCK	5	4	10
STORAGE	5	5	10

Using this information, a test application was developed that utilized the common functions of these applications. The permissions used for the application were the same permissions presented in table 4.2. The application built by the researcher resulted in many different permissions opened with no functionality utilizing them. The permissions set were deconstructed from the group permissions to individual ones. Through three different rounds of requirements mapping, the specific requirements were determined and a final list of permissions was generated.

Table 4.3 lists the starting permissions used for initial test application. From this, a set of requirements was built and a list of permissions was determined for the requirements. Table 4.3 shows the requirements and the related permissions for them.

This application verified the steps in the SADM process. The researcher built an application that achieved all of the functional requirements with the fewest permissions possible. It was determined that if any of the permissions were removed, then the application would not meet all of the requirements.

Validation of the model began after the initial tests with a group of volunteers. The application built by the researcher is used as a control to compare with the volunteers' final product. The case study relied on each participant building the same application built by the researcher during verification.

4.1.2 Test Environment

Validation of the model was performed by a group of participants recreating the application built during verification. Participants with backgrounds in software development were chosen and given the application requirements for developing. The validation group consisted of a mixture of students and industry developers. Each participant was given the requirements for the verified application. The participants used the SADM to build the same application as the researcher and had their final permission list compared to that predetermined before testing began.

Table 4.3

Requirements and Associated Permissions

Functional Requirement	**Permission Associated with Requirement**	**Notes**
Encrypt File/Message	None	No permission used to encrypt files and messages before sending to device.
Decrypt File/Message	None	No permission used to decrypt files and messages before sending to device.
Connect to other device	None	Two devices need to be able to communicate with each other.
Connect to the internet	INTERNET	Allows for sockets to be opened between devices over the internet.
Only transfer over wifi	ACCESS_WIFI_STATE	Needs to access the wifi hardware on the device for internet usage.
	CHANGE_WIFI_STATE	Connects to wifi hardware of device.

Table 4.3

(continued)

Functional Requirement	**Permission Associated with Requirement**	**Notes**
Send Messages to devices	None	No special permission is needed to send text over network sockets.
Send Files to devices	READ_EXTERNAL_STORAGE	Read files from a device in order to send over network.
	WRITE_EXTERNAL_STORAGE	Files received over network needs to be written to device.

4.1.2.1 Test Group

The SADM is a software development model. Since it is a software development model the targets for participation are those familiar with software development on some level. A mix of students and industry workers were used for this test. Four of the participants were Computer Science and Software Engineering students at Mississippi State University while seven were software developers working in industry in various areas in and around Mississippi.

As shown in Table 4.4, all of the participants have experience with programming applications. Only one reported his experience as being medium with a majority of participants rating themselves as high. When asked about their experience with Android application

Table 4.4

Participant Demographic

Subject	Gender	Programming Experience*	Android Development Experience*	Use Smartphone Everyday
1	M	High	None	Yes
2	M	High	Low	Yes
3	M	Medium	None	Yes
4	M	High	Low	No
5	M	High	Low	Yes
6	F	High	Medium	Yes
7	M	High	None	Yes
8	F	High	Low	Yes
9	M	Expert	High	Yes
10	M	Expert	Medium	Yes
11	M	High	Low	Yes

* None(0 years), Low(1-2 years), Medium(2-3 years), High(3-4 years), Expert(4+ years)

development, the participants have not had as much experience with that platform. Most participants had low to no experience developing applications on Android, yet one reported having high experience. While the participants are experienced programmers, there was more variety in each subject's personal experience with development on the Android platform.

With the exception for one individual, all of the participants reported using a smartphone in their everyday life. As explained in Chapter 2 this is a result of the ubiquity of these devices in people's lives. Development of an application for a smartphone is different than development on a standard computer system. The type of smartphone each participant used was deemed irrelevant as the main concern is understanding on a general level of how someone interacts with the device.

4.1.2.2 User Case Study

Using the SADM requires that a developer build an application for the Android environment. By using this methodology, a developer is able to build any application he or she wants by providing the ability to determine which permissions should be used before development. This allows for any application to be built using this methodology. To control the test groups, an application was built by the author before testing. This application was used to determine a set of requirements given to each participant. The resulting application's Manifest.xml was used to determine the appropriate permissions the participants should use. This same application was given to each participant to keep a control over permissions expected compared to permissions delivered.

Each participant was provided with a detailed explanation of the SADM. The researcher provided the visual model and an explanation of the methodology. The requirements provided the vision of the end application and the SADM provided the structure on how to build it. The participants followed the researcher's model to develop the application. Once the model and requirements were given to the participants and development began, the researcher had no more interaction with the participants until the task was completed. Each participant was then given a survey and briefly interviewed by the researcher. The survey and the participant's resulting permission list was returned to the researcher for analysis. The survey answers provided the researcher with how the participants used the methodology to generate their final permission list. The final permission list was compared to the expected list generated by researcher. Section 4.2 explains how the participants used this model and how their final list compared to the one the researcher expected.

4.2 Experiment Results

After the participants completed their tasks, a survey was filled out and returned to the researcher along with the final list of permissions generated for the application. By using the SADM methodology, each participant would generate a list of permissions. This list is then compared to the list generated by the researcher. The goal is to have the participants achieve the same permission list as expected by the researcher. Having the end permission list by itself is not enough. This list has to have been generator by using the methodology. The surveys allow for each participant to respond to how he or she used the methodology to build the application.

By looking at the final permission list and the information provided by the post-test surveys, the researcher determined how the participants used the model to build the application. The surveys also provide information for how useful the methodology is for application building in general. The SADM is a development model and can only be effective if it is useful. In addition to testing the model usage for generating the final permissions, it is also of interest to the researcher to inquire into the overall usefulness of following this methodology for development.

The rest of this section details the results for the usage of the model for permission building and the usefulness of the model as a development model.

4.2.1 Permission Generation Results

Will following the SADM methodology result in a minimal number of permissions set for an Android application? This is the question attempting to be answered by this study. As previously said ”minimal” refers to the number of permissions needed so that every functional requirement is addressed. In this study each participant was given requirements (as specified in section 4.1.1) for an application to be built. The application is an Android based application that allows users on two different devices to send messages to each other and share files with one another. Each participant built the application following the development strategies of the proposed methodology. The participants were then given a post-test survey to fill out. This survey allowed the participants to report on their usage of the model and how effective it was at guiding the permission gathering for the application. The survey was returned with a list of the resulting permissions designed to the researcher.

The iterative refactoring process of the model is the main part of determining the resulting permissions. The combination of resulting permissions was taken into account with how many times each participant needed to iterate through and redesign his or her permissions. Question 9 (Q9) of the survey asks the participants if any time was spent reevaluating the application's final permission list with the one that they built before development. Each participant reported that he or she had to refactor at least one time to get their final permission list. Each participant's permission list was compared to the researcher's expected list and every participant reported had a match. This researcher defines a match as a participant's final permission list exactly matching the expected permission list generated by the researcher prior to running the experiment. Table 4.5 summarizes these results.

Table 4.5

Permission Results Summary

Subject	Final permissions compared to expected permissions	Number of times reevaluating permissions
1	Match	1
2	Match	3
3	Match	1
4	Match	1
5	Match	2
6	Match	1
7	Match	2
8	Match	1
9	Match	3
10	Match	1
11	Match	1

Most of the participants only had to reevaluate one time. Each participant reported that before development, a list of permissions was determined to satisfy the requirements. Upon reaching the reevaluation phase, each participant was able to remove permissions not in use and design a new set of permissions for the application. Each participant reported their beginning permission list along with their final permission list. Table 4.6 shows the comparison of beginning and ending permissions.

Table 4.6

Initial versus Final Permission List

Subject	Number of Initial Permissions	Number of Final Permissions	Number of times reevaluating permissions
1	10	5	1
2	10	5	3
3	10	5	1
4	10	5	1
5	9	5	2
6	11	5	1
7	10	5	2
8	9	5	1
9	8	5	3
10	10	5	1
11	10	5	1

Every participant reported having a higher number of permissions determined prior to application development. After spending as few times as one reevaluation process, the number of permissions was dropped to five total.

Question 11 on the survey asks the participants to determine if every permission implemented in the application corresponded to functionality given by the requirements. Every participant reported that each permission is corresponded to at least one functionality of the application. This can be seen by the fact that each participant started with more permissions than necessary and through refactoring got them trimmed to a fewer number.

Questions 5 and 12 on the survey both address the issue of using the SADM methodology to get these permissions. As reported above, each participant was able to trim down the total permissions being used while going through the development process. These two questions allow for the participants to report on how effective the model was at providing them the way to get the final permission list. Tables 4.7 and 4.8 summarize the results of these questions.

Table 4.7

Effectiveness for planning permissions

Q5:How effective was SADM at making the development process plan out the permissions used by the application?	
Option	**Fequency/Percentage**
Not effective	0/0%
Slightly effective	0/0%
Somewhat effective	2/18%
Very effective	6/55%
Extremely effective	3/27%

The participants all reported that using the SADM was very helpful and very effective at planning out and determining the permissions of the application. The participants reported

Table 4.8

Planning permissions before implementation

Q12:How much did it help to plan out the permissions before programming began on the application?	
Option	**Fequency/Percentage**
Not helpful	0/0%
Slightly helpful	0/0%
Somewhat helpful	1/9%
Very helpful	9/82%
Extremely helpful	1/9%

that by using the model, they were able to determine what permissions were necessary and reduce the number of permissions open. The resulting applications built during testing ended up with fewer permissions open than if the application was built through with one iteration.

4.2.2 Permission Generation Analysis

Each participant was given the same requirements and details for the same application to build. Using the SADM for development, each participant was successful at building the application. Each participant returned a list of pre-development permissions and a list of final permissions. It has been shown that each participant developed a list of permissions that was unsatisfactory during the building of the application. The participants all ended with a matching set of permissions to what the researcher expected. No participant was able to generate the correct permissions from the start and had to iterate through the permission evaluation process at least once.

The model provided the process by which the participants were able to reevaluate the permission list. The iterative process when this reevaluation occurs gave each participant the opportunity to trim down his or her initial permission list. Most participants had to reevaluate only once. None of the participants reported determining the minimal permission list prior to implementation of the application. Each participant took the general requirements and decomposed them into more specific ones. A list of permissions that could possibly satisfy the requirements was generated. Each participant had to reevaluate the ending permission list compared to the functional requirements.

The SADM was reported as being overall effective with providing the participants with the ability to plan out the permissions. By combining the reported level of effectiveness, the participants felt the SADM was at gathering permissions with the fact that each participant reevaluated their set of permissions at least one time, we can see that the participants did not just guess at the permissions. The participants were able to determine the appropriate permissions by iteratively evaluating the permissions against the functional requirements.

4.2.3 SADM Usage Results

The SADM is a software development model and as such needs to be practical for such usage. No matter how effective it is for refining a list of permissions, the model will not be used if it interferes with the development process. The survey given to the participants allowed each one to rate how effective using the model was in the overall development process. The goal of these questions is to gather insight into how developers use the model for their development. Tables 4.9-4.14 summarize this information.

The participants all were given the requirements of the application. All participants have experience in software development. While actual development on Android was not as experienced by the participants, they all understood how to take requirements and build a final product at the end. Before explanation of the SADM, each participant was given some time with the requirements to figure out a plan of development. Three of the participants responded to the researcher that they had no real plan for the application and would just ”wing it”. Six participants responded with gathering a list of possible permissions and opening them for development. Two responded with using the group permissions instead of individually opening each one.

Table 4.9

Planning strategy before test

Prequestion:What plan do you have for development?	
Synthesised response	**Fequency/Percentage**
”Just Wing It”	3/27%
Gather a master list of possible permissions to open	6/55%
Use Group Permissions	2/18%

The first group of participants had planned on coding the application and enabling what they needed when they needed it. This process lent itself to the possibility of having the fewest possible permissions set for the application. However, there is no way to know certainly that it is the fewest number without some way to reevaluate and rework the permissions. The second group proposed to go through and get one master list of all

permissions that could possibly be used and open them. There was no plan given by these participants to remove permissions later that were not currently being used. This would result in an application with many open permissions many of which would not be used. The last group planned on using the group permissions. The reasoning is that group permissions generalized different permissions together and resulted in fewer permissions being set. As explained earlier, while the developer has to make fewer calls to the group permissions, there are not fewer permissions set. It opens a whole group of permissions even if they are not all being used.

The participants had enough information to make the general decisions for preplanning but none of their strategies could guarantee minimal usage of permissions. After the initial planning, the SADM was explained to each person. Testing began after each participant understood how to follow the model. In the survey, each participant was able to rate how their experience was with using the SADM for development. In question 2, the participants rated how difficult it was to understand and follow the proposed model.

Table 4.10

Understanding and Following the SADM

Q2:How difficult is SADM to understand and follow?	
Option	**Fequency/Percentage**
Not difficult	6/55%
Slightly difficult	5/45%
Somewhat difficult	0/0%
Very difficult	0/0%
Extremely difficult	0/0%

Table 4.10 shows the results for question 2 of the survey. Fifty-five percent of the participants all reported that it was not difficult at all to understand the model. Forty-five percent of the participants reported that it was slightly difficult to understand. The model seemed to be easy to understand and follow for development. By making it easy to understand, it makes it easier for developers to use the model for software application development.

The next question asked how feasible do the participants feel the SADM is for development of Android applications. This is a software development model but it specifically targets the Android platform. Question 3 allowed the participants to address this concern. Table 4.11 shows the responses to this question.

Table 4.11

Feasibility of the SADM

Q3:How feasible is SADM in the development of Android applications?	
Option	**Fequency/Percentage**
Not feasible	0/0%
Slightly feasible	0/0%
Somewhat feasible	1/9%
Very feasible	8/73%
Extremely feasible	2/18%

Mostly the participants felt that it was very feasible to use the model for Android development. Nine percent felt that it was somewhat feasible; 73% felt that it was very feasible; and 18% felt that it was extremely feasible. The ability to use the model for development is

a major concern for any development model. By saying the SADM is very feasible to use for development allows it to be more readily usable by developers. No correlation could be found between how feasible it was to use the proposed model and the level of previous Android experience the participant had. The participant who felt it was somewhat feasible had reported none for previous experience with Android development. In contrast one of the two participants who reported extremely feasible also had reported none for previous experience with Android development as well.

The participants report that the proposed model is feasible to be used in Android development and is easy to understand. All software development models add steps but they cannot be viewed has a hindrance to the process or the model will never be used. Question 4 of the survey address this concern and the results are shown in table Table 4.12.

Table 4.12

Difficulty Using the SADM

Q4:How difficult did SADM make the application development process?	
Option	**Frequency/Percentage**
Not difficult	11/100%
Slightly difficult	0/0%
Somewhat difficult	0/0%
Very difficult	0/0%
Extremely difficult	0/0%

All 100% of the participants reported that it added no difficulty to the development process. Using the SADM was used and did not interfere with the participants' ability to develop the application during testing.

Each participant was asked to give his or her opinion on the strength and weaknesses of proposed model. Question 7 and question 8 ask for this information and are presented in table Table 4.13 and table Table 4.14 respectively. In these tables, each participant's response was condensed into a general concern. Each concern was given a response from one participant to show how the participants responded to the question. While different participants have different responses phrased, similar responses are condensed to a general response.

Table 4.13

Strengths of the SADM

Q7:What strengths does the SADM have for the application development process?		
Response Summary	**Frequency/Percentage**	**Sample Response**
Understanding how to select permissions	3/27%	Knowledge of what certain permissions do
Application built with few permissions	7/64%	The finished app will be more appealing with fewer permissions
No unused permissions	1/9%	Creates secure applications without unnecessary application permissions

Each participant responded with the strength of the SADM being how it handles permission generation. Twenty-seven percent of participants feel that the SADM provides the developer with the ability to understand how to select the permissions. Sixty-four percent of participants reported that the strength of the SADM is applications build using this process results in the fewest permissions needed. Nine percent of the participants felt the strength of the model is that the resulting application had no unused permissions.

Question 8 asked the participants to describe the weakness of the proposed methodology. The results are presented in Table 4.14. Similar to table Table 4.13, each response was group into similar categories and summarized. Each category is presented along with a sample response that belonged to the category.

Table 4.14

Weakness of the SADM

Q8:What weaknesses does the SADM have for the application development process?		
Response Summary	**Frequency/Percentage**	**Sample Response**
none	4/36%	”None observed”
Easy to get distracted from main development	5/46%	”Spent too much time over compensating on permissions that we would later not need”
Intermediate steps should test the permissions.	2/18%	”I think there should be some intermediate processes to insure necessary permissions are present”

The participants responded with a weakness and the responses were grouped together. Thirty-six percent of participants responded with no weaknesses. Forty-six percent of participants felt that it was easy to get too distracted by the permissions and lose focus on building the actual application. Eighteen percent of participants feel that iterating through the process was extraneous. These participants suggested that having more intermediate steps for checking the permissions would work better. Answering these questions provided the researcher with an understanding of how well the SADM works as a development model. The participants all responded positively with respect to using the proposed model for future development of Android applications.

4.2.4 SADM Usage Analysis

Each participant used the SADM to develop an application from requirements to final product. Section 4.2.2 presented the results of the application built using the proposed methodology. While using the model allowed the participants to build applications with minimal permissions, a development model must also be usable for the software development process. In addition to reporting on the permissions generated for the final product, each participant also reported on their usage of the proposed model.

It is of interest to note how difficult it is by software developers to use the SADM and how effective it is at building Android applications. The participants all reported that the model was easy to use and useful in the development process. The SADM provided the guidance the developers needed to understand the permission system of Android and provided the method needed to map functional requirements to permissions. By applying

an iterative approach to requirements mapping, the developers were able to generate a minimal set of permissions the application needed to use. This is the biggest strength reported by the participants the SADM gives developers. The participants all reported using the SADM gives the developer minimal permissions to satisfy the requirements. This is done by understanding how the permissions work and by forcing the developers to spend time mapping requirements to permissions.

The SADM methodology is not immune to problems. The participants reported on the weaknesses of the proposed model. Due to the nature of reworking permissions of an application, the big weakness of the SADM is how much time is spent on just permission mapping. Many of the participants felt it is easy to get lost in permission gathering. The proposed model utilizes an iterative approach by reevaluating the permissions and starting back at the beginning. Iterating through the whole process many times can be time consuming and is a weakness that can be addressed.

4.3 Main Hypothesis and Research Questions Results

The hypothesis of this dissertation is that by following the methodology modeled by the SADM, developers will be able to produce an application with minimal permission being used. For the test, the researcher built an application designed to send messages between devices as well as share files between the two. By iterating through the design, the researcher produced an application with a set of permission, such that removal of any permission would remove functionality from the application. The requirements for this application was given to participants to test the SADM. Every participant was able to gen-

erate a list with the exact same permissions as the one determined by the researcher. Each participant did start with a larger permission list then what they finally ended with. By using the SADM, each participant was able to remove useless permissions until all that remained was the minimal set needed to achieve functionality. Each participant successfully built an application with the predetermined minimal permissions.

The surveys answered provided the researcher with information answering the research questions asked to validate the usefulness of the SADM as a software development model.

4.3.1 Is the model easy to use?

It is reported by the participants that the model was easy to use. For any software development model to be effective it has to be used by the developers. A model that is easy to understand is more likely to be used by developers. Each participant felt that the model was clear and easy to understand. The participants felt that the model was feasible to be used for Android development and felt that it added no more difficulty to the overall process. The model guides the developers in decomposing requirements to a base form that can be used by a certain permission or set of permissions. The participants were able to follow this process and not have it interfere with the development process.

4.3.2 Does the model help with determining what permissions to use ahead of time?

The SADM provided the participants with the methodology to map functional requirements to different permissions. Each participant built a list of permissions at the beginning of development but had to reevaluate their list. Their lists were trimmed down to the appropriate permissions. This worked in context of providing the minimal set of permissions, but

they still had a different set before development. The participants were able to map permissions to requirements. However, once the participants began implementing the application they began to understand how some permissions were not necessary. This resulted in the reevaluation cycle to get a new set of permissions for implementation. The SADM did provide assistance in determining permissions ahead of time even if it was different from the final list.

4.3.3 Does the model result in developer specified minimal permission usage?

The participants all reported the same five permissions needed for the application. These permissions were the same ones determined prior to testing by the researcher. Each participant reported their set of permissions determined prior to implementation and determined upon completion. Every participant needed to iterate through permission acquiring, but they were able to end with the same five permissions expected. These permissions were generated by the participants by using the proposed model and every application tested utilized the same minimal set of permissions.

4.3.4 Can minimal permission usage be built in from the beginning of development?

The SADM did not appear to build the minimal permission base at the very beginning of development. While the SADM does provide the means to determine different permissions based on the functional requirements, it does not provide a way to know exactly how the permissions will be utilized prior to implementation. Most of the participants reported little to no prior experience developing applications on Android with a few reporting higher. The more experienced Android developers were expected to have a better under-

standing what permissions were needed prior to implementation. There was no observable correlation between prior Android experience and the ability to determine permissions at the beginning of development.

4.3.5 Can the model be used by someone with little to no prior experience with application development on Android?

All of the participants had experience with software development in general. Their specific experience with Android development was scattered with most reporting little to none. Each participant reported the ability to easily use the model to develop applications for Android. The model guided their development process for people with no experience with Android development. The SADM provided the mechanism to map the functional requirements to the permissions. It was thought that participants with higher levels of experience with Android would not follow the model as strictly. From the surveys filled out, there is no correlation between the usability of the model between those with high Android experience and low Android experience.

4.3.6 Research Question Summary

The surveys provided answers to the research questions and was discussed above. Table 4.15 summarizes the research questions. These research questions are listed with the expected outcome and the actual outcome.

The table summarizes the research questions of this dissertation. The information was gathered from the survey done post application development. The survey was compared to the resulting list of permissions to answer the questions. The next chapter discusses the

Table 4.15

Research Question Summary

Research Question	**Expected**	**Result compared to expected**	**Survey Question**
Is the model easy to use?	The model will be easy to follow and use for development	Yes	Q2,Q3,Q4,Q6
Does the model help with determining what permissions to use ahead of time?	The model will provide a way to determine permissions at the beginning of development	Yes	Q5,Q12
Does the model result in developer specified minimal permission usage?	Use of the SADM will result in minimal permissions	Yes	Q11
Can minimal permission usage be built in from the beginning of development?	Through proper decomposition of requirements the minimal set of permissions can be determined at the beginning of development	Every participant had to redesign permissions at least once	Q12, Q9, Comparing initial list to final list
Can the model be used by someone with little to no prior experience with application development on Android?	The model can be used by anyone regardless of prior Android experience	Yes	Q1, Q4
Does the amount of previous experience have an affect on usage of the proposed model?	Developers with less experience will have an easier time using the model	Every participant had easy time with model	Q1, Q4

conclusions drawn from this experiment as well as future work where the weaknesses of the SADM are addressed.

CHAPTER 5

CONCLUSION

This chapter presents conclusions based on the analysis of the resulting information gathered from the experiment. Contributions of the defined model along with an avenue for future research are discussed in this chapter.

5.1 Contribution

The Secure Android Development Model (SADM) proposed in this dissertation provides a methodology to developers that restricts the usage of permissions in Android applications. Review of the literature revealed that attempts at Android security focused on protecting devices from applications installed. These applications are both intentionally malicious and non-malicious. The non-malicious applications tend to be faulty and poorly designed allowing for exploitation by other applications. The proposed model attempts to solve the problem of poorly designed applications being exploited by removing exploitation avenues during development.

After development of the proposed model, the researcher began by conducting a preliminary case study for verification of the model. This was accomplished by examining twenty different applications from the Android app store all of which had similar functionality. The applications chosen were a group of file sharing and messaging apps. From these

apps, the researcher gathered a list of commonly used permissions and classified them into used ones and unused ones. After classification of permissions, the researcher began development of an application which allowed for messaging and file sharing using the same permissions used by these other applications. Following the model, the researcher was able to reduce the permissions down to a select group. Removal of any permission would break functionality of the application.

The created application became the control for participants testing the model. Upon completion and verification, the researcher gathered 11 volunteers and provided the requirements for the application. The participants were then given the development model and a lecture to its meaning and workings. The participants were given a survey upon completion of the application. Each participant provide a list of initial permissions and final permissions along with the completed survey to the researcher. It was shown that by using the development model and iterating through it at least once, each participant was able to build the application with the same set of final permissions as the researcher. Analysis of the model itself was performed from the information provided by the surveys to validate the usefulness of the SADM as a software development model.

The SADM contributes to the field of information security because it is a development model focused on minimizing permission usage. While plugins have been built to assist developers in selecting permissions[32], they are insufficient in providing the necessary capabilities of the designer to reevaluate which permissions are needed. Developers relied on guessing what permissions they would need for development. This led to grabbing many different permissions if there was a small chance it would be needed. Utilizing the SADM

allowed the participants to develop the application and work through which permissions are necessary and eliminating the ones that are not.

5.2 Publication Plan

The following paper from this work have been submitted for peer review: Refereed Conference:

C. Ivancic and D. Dampier, ”A Developer’s Guide to Android Security: Building Security in Your Apps”. To be reviewed at the 24th International Conference on Software Engineering and Data Engineering (SEDE) 2015.

The following paper is planned to be submitted for journal publication Refereed Journal:

C. Ivancic and D. Dampier, ”Secure Android Development: A Methodology for Building Security into Your Android Applications”. Will be submitted to Pervasive Mobile Computing special issue on Mobile Security, Privacy and Forensics.

5.3 Future Research

Through testing it was discovered that the developers can spend much of their time reevaluating the permissions. The iteration process takes the developers back through the development cycle starting at the beginning with mapping requirements to permissions. This resulted in some participants feeling that too much time was spent cycling through the model many times. One potential solution to this problem is to modify the model implementation to allow for more rapid redesigning of the permissions during the implementation phase. Research can be done by modifying the existing model implementation

and running tests again attempting to determine how much less time is spent by not having to repeat the whole process again.

The proposed methodology is implemented as a model designed specifically for the Android platform. Each smartphone OS has its own implementation of permissions and application interaction. The idea of using permissions to allow for applications to communicate is used in many different; however, it is different for each OS. Research can be done to build upon the existing model to allow for a more generalized model. The idea of designing the model to be more generalized is to allow for a standard methodology to be used by developer of any mobile device OS.

For this dissertation, each participant was asked for his or her previous experience with Android development. No significant difference was observed from the results of those with high experience in Android programming versus those with low to none. For the most part, the participants report a low to no level of previous experience programming apps for Android. A test can be designed to test groups of developers with high experience with Android and test groups with low level of Android programming. A comparison of these results will give more insight as to the effectiveness of the model given a previous level of Android experience.

REFERENCES

[1] "Android API Overview, Android documentation ,", `http://developer.android.com/preview/api-overview.html`, 2014, Accessed: 22 May 2014.

[2] H. Banuri, M. Alam, S. Khan, J. Manzoor, B. Ali, Y. Khan, M. Yaseen, M. N. Tahir, T. Ali, Q. Alam, et al., "An Android runtime security policy enforcement framework," *Personal and Ubiquitous Computing*, vol. 16, no. 6, 2012, pp. 631–641.

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

[4] D. Barrera and P. Van Oorschot, "Secure software installation on smartphones," *IEEE Security & Privacy*, vol. 9, no. 3, 2011, pp. 42–48.

[5] M. Becher, F. C. Freiling, J. Hoffmann, T. Holz, S. Uellenbeck, and C. Wolf, "Mobile security catching up? revealing the nuts and bolts of the security of mobile devices," *Security and Privacy (SP), 2011 IEEE Symposium on*. IEEE, 2011, pp. 96–111.

[6] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.-R. Sadeghi, and B. Shastry, "Poster: the quest for security against privilege escalation attacks on android," *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 741–744.

[7] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for android," *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 2011, pp. 15–26.

[8] P. S. S. Council, "PCI Mobile Payment Acceptance Security Guidelines for Developers," *PCI Mobile Payment Acceptance Security Guidelines, version 1.0*, 2012.

[9] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy, "Privilege escalation attacks on android," *Information Security*, Springer, 2011, pp. 346–360.

[10] W. Enck, "Defending users against smartphone apps: Techniques and future directions," *Information Systems Security*, Springer, 2011, pp. 49–70.

[11] W. Enck, M. Ongtang, P. D. McDaniel, et al., "Understanding Android Security.," *IEEE security & privacy*, vol. 7, no. 1, 2009, pp. 50–57.

[12] E. Erturk, "A case study in open source software security and privacy: Android adware," *Internet Security (WorldCIS), 2012 World Congress on*. IEEE, 2012, pp. 189–191.

[13] H. Estler, M. Nordio, C. A. Furia, B. Meyer, and J. Schneider, "Agile vs. structured distributed software development: A case study," *Global Software Engineering (ICGSE), 2012 IEEE Seventh International Conference on*. IEEE, 2012, pp. 11–20.

[14] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 2011, pp. 3–14.

[15] A. P. Fuchs, A. Chaudhuri, and J. S. Foster, "SCanDroid: Automated Security Certification of Android," 2009.

[16] A. Goode, "Managing mobile security: How are we doing?," *Network Security*, vol. 2010, no. 2, 2010, pp. 12–15.

[17] C. T. Hager and S. F. Midkiff, "An analysis of Bluetooth security vulnerabilities," *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*. IEEE, 2003, vol. 3, pp. 1825–1831.

[18] A. A. Janes and G. Succi, "The dark side of agile software development," *Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software*. ACM, 2012, pp. 215–228.

[19] W. Jansen and K. Scarfone, "Guidelines on cell phone and PDA security," *NIST Special Publication*, vol. 800, 2008, p. 124.

[20] S. Liaskos, S. A. McIlraith, S. Sohrabi, and J. Mylopoulos, "Representing and reasoning about preferences in requirements engineering," *Requirements Engineering*, vol. 16, no. 3, 2011, pp. 227–249.

[21] C. Marforio, A. Francillon, S. Capkun, S. Capkun, and S. Capkun, *Application collusion attack on the permission-based security model and its implications for modern smartphone systems*, Department of Computer Science, ETH Zurich, 2011.

[22] C. Miller, "Mobile attacks and defense," *Security & Privacy, IEEE*, vol. 9, no. 4, 2011, pp. 68–70.

[23] M. Ongtang, K. Butler, and P. McDaniel, "Porscha: Policy oriented secure content handling in Android," *Proceedings of the 26th Annual Computer Security Applications Conference*. ACM, 2010, pp. 221–230.

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

[25] P. Ragunath, S. Velmourougan, P. Davachelvan, S. Kayalvizhi, and R. Ravimohan, "Evolving a new model (SDLC Model-2010) for software development life cycle (SDLC)," *International Journal of Computer Science and Network Security*, vol. 10, no. 1, 2010, pp. 112–119.

[26] B. Reed, "A Brief History of Smartphones," *PC World (2010)*.

[27] M. Rogowsky, "More Than Half Of Us Have Smartphones, Giving Apple And Google Much To Smile About ,", http://www.forbes.com/sites/markrogowsky/2013/06/06/more-than-half-of-us- have-smartphones-giving-apple-and-google-much-to-smile-about/, 2014, Accessed: 10 August 2014.

[28] N. B. Ruparelia, "Software development lifecycle models," *ACM SIGSOFT Software Engineering Notes*, vol. 35, no. 3, 2010, pp. 8–13.

[29] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "Andromaly: a behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems*, vol. 38, no. 1, 2012, pp. 161–190.

[30] W. Shin, S. Kwak, S. Kiyomoto, K. Fukushima, and T. Tanaka, "A small but non-negligible flaw in the Android permission scheme," *Policies for Distributed Systems and Networks (POLICY), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 107–110.

[31] J. Srinivasan and R. Agila, "Software Development Life Cycle Model Incorporated with Clemency Brass," *In International Journal of Innovative Research in Advanced Engineering (IJIRAE)*, vol. 1, no. 4, 2014.

[32] T. Vidas, N. Christin, and L. Cranor, "Curbing android permission creep," *Proceedings of the Web*, 2011, vol. 2.

[33] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "Malicious android applications in the enterprise: What do they do and how do we fix it?," *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*. IEEE, 2012, pp. 251–254.

APPENDIX A

PARTICIPANT DEMOGRAPHICS

Subject ID _____

Demographic Information

(Complete each question and mark the circle that most appropriately matches where applicable)

1. **What is your gender?** ○ Male ○ Female

2. **What is your age?** _____

3. **What is your occupation?** _____

4. **What is the highest level of education you have completed?**

Some High School	High School Graduate	Some College	AA/AS
○	○	○	○

Bachelor	Some Graduate School	Masters	Ph.D./M.D./J.D.	Postdoctorate
○	○	○	○	○

5. **What is your major field of study?** *(Where applicable)*

_____

6. **What is your level of prior computer experience?**

No Experience		Novice			Expert
0	1	2	3	4	5
○	○	○	○	○	○

7. **What is your level of prior programming experience?**

No Experience		Novice			Expert
0	1	2	3	4	5
○	○	○	○	○	○

8. **Do you use a smartphone in everyday life?** No ○ Yes ○

9. **What is your level of prior Android app development?**

APPENDIX B

PARTICIPANT INFORMED CONSENT FORM

Mississippi State University
Informed Consent Form for Participation in Research

Title of Research Study: Secure Android Development Model

Researchers: Christopher Ivancic, MSU Dr. David A. Dampier, MSU

Purpose
The purpose of this research is to determine how effective this development model is in guiding developers in building applications for the Android operating system.

Procedures
If you decide to participate in this study, you will be asked to complete a demographics questionnaire and surveys concerning your prior experience with development models and software construction. You will be given a development model to use for building applications for Android. You will use it to guide your software development process. You will then be given a survey about the software development process you used.

Risks or Discomforts
There are no major physical discomforts involved in this study. Risks are minimal and do not exceed those of normal office work. Please tell us if you are having trouble with any task or if you need additional rest and the investigator will be happy to accommodate you in any way possible. If you feel any discomfort, please tell the person assisting you immediately.

Benefits
Although there is no benefit directly to you for participating, this study will help to further our understanding of application development modeling. As a participant, you will design and build an application for the Android platform using a development model to help guide you in the process.

Confidentiality
All of your responses will be kept strictly confidential. To protect the confidentiality of this information, we will assign your data and a code number that will only be known to the members of the research project. All of the information which you provide us today will be marked with the code number, not your name. All information will be stored in a computer for analysis using only your code number for identification. The information collected during the study will be used solely for the purposes of understanding and evaluating secure application development. No indication of your individual answers to questions will be given to anyone. We want you to be completely confident that you may feel free to answer all questions without concern that it may affect you in any way.
Please note that these records will be held by a state entity and therefore are subject to disclosure if required by law. Research information may be shared with the MSU Institutional ***Review Board (IRB) and the Office for Human Research Protections (OHRP).***

Questions
If you have any questions about this research project, please feel free to contact the investigator, Christopher Ivancic, (662) 325-3049, cpi4@msstate.edu, HPCC, Room 201. You may also contact the Faculty Advisor, David A. Dampier, Ph.D., (662) 325-0779, dampier@cse.msstate.edu, HPC2, Room A129.

For questions regarding your rights as a research participant, or to discuss problems, express concerns or complaints, request information, or offer input, please feel free to contact the MSU

Regulatory Compliance Office by phone at 662-325-3994, by e-mail at irb@research.msstate.edu, or on the web at http://orc.msstate.edu/humansubjects/participant/.

Voluntary Participation

Please understand that your **participation is voluntary.** Your **refusal to participate will involve no penalty or loss** of benefits to which you are otherwise entitled. You **may discontinue your participation** at any time without penalty or loss of benefits.

Statement of Consent

Please take all the time you need to read through this document and decide whether you would like to participate in this research study.

If you agree to participate in this research study, please sign below. You will be given a copy of this form for your records.

_____	_____
Participant Signature	Subject ID
_____	_____
Participant Name (printed)	Date
_____	_____
Investigator Signature	Date

APPENDIX C

PARTICIPANT POST-SURVEY QUESTIONS

Q1	What is your level of experience with application development on the Android system? a. No experience (0-1 years) b. Little experience (1 – 2 years) c. More experience (2 – 3 years) d. Very experienced (3+ years)
Q2	How difficult is SADM to understand and follow? a. Not difficult b. Slightly difficult c. Somewhat difficult d. Very difficult e. Extremely difficult
Q3	How feasible is SADM in the development of Android applications? a. Not feasible b. Slightly feasible c. Somewhat feasible d. Very feasible e. Extremely feasible
Q4	How difficult did SADM make the application development process? a. Not difficult b. Slightly difficult c. Somewhat difficult d. Very difficult e. Extremely difficult
Q5	How effective was SADM at making the development process plan out the permissions used by the application? a. Not effective b. Slightly effective c. Somewhat effective d. Very effective e. Extremely effective
Q6	Does the SADM add to the development process of Android applications? If so, what does it add?
Q7	What strengths does the SADM to the application development process?

Q8	What weaknesses does the SADM have to the application development process?
Q9	During development was there any time spent in reevaluating the permissions used in the application? If so, how many times did reevaluation occur?
Q10	Did you have to manipulate any of the phases in the model during development? If so which ones and how did you manipulate it?
Q11	Did use of the SADM result in an application where every permission has an associated functional requirement? (Were there any permissions implemented that did not have any functionality?)
Q12	How much did it help to plan out the permissions before programming began on the application? a. Not helpful b. Slightly helpful c. Somewhat helpful d. Very helpful e. Extremely helpful
Q13	If there was any point in the process you would change, what would it be and why?

Subject ID _____

Post Survey of the Secure Android Development Model (SADM)

APPENDIX D

GENERATED MANIFEST FILE

```
<?xml version="1.0" encoding="UTF-8"?>
-<manifest package="caseTest.androidfiletransfer"
    xmlns:android="http://schemas.android.com/apk/res/android
    ">
        <uses-sdk android:minSdkVersion="14"/>
        <uses-permission android:name="android.permission.
            ACCESS_WIFI_STATE"/>
        <uses-permission android:name="android.permission.
            CHANGE_WIFI_STATE"/>
        <uses-permission android:name="android.permission.
            INTERNET"/>
        <uses-permission android:name="android.permission.
            READ_EXTERNAL_STORAGE"/>
        <uses-permission android:name="android.permission.
            WRITE_EXTERNAL_STORAGE"/>
        -<application android:theme="@style/AppTheme"
            android:label="@string/app_name" android:icon="
            @drawable/ic_launcher" android:allowBackup="true"
            >
                -<activity android:name="caseTest.
                    androidfiletransfer.MainActivity"
```

```
                android:label="@string/app_name"
                android:screenOrientation="portrait">
                    -<intent-filter>
                        <action android:name="
                            android.intent.action.
                            MAIN" />
                        <category android:name="
                            android.intent.category.
                            LAUNCHER" />
                    </intent-filter>
                </activity>
<activity android:name="caseTest.androidfiletransfer.
    BroadcastActivity" android:label="@string/
    title_activity_broadcast" android:screenOrientation="
    portrait"> </activity>
<activity android:name="caseTest.androidfiletransfer.
    ServerActivity" android:label="@string/
    title_activity_server" android:screenOrientation="
    portrait"> </activity>
<activity android:name="caseTest.androidfiletransfer.
    ClientActivity" android:label="@string/
```

```
        title_activity_client" android:screenOrientation="
        portrait"> </activity>
            </application>
</manifest>
```