*Journal of Universal Computer Science, vol. 22, no. 4 (2016), 459-474*
*submitted: 1/10/15, accepted: 30/3/16, appeared: 1/4/16* © *J.UCS*

# Analysis of Permission-based Security in Android through Policy Expert, Developer, and End User Perspectives

**Ajay Kumar Jha**
(School of Computer Science & Engineering, Kyungpook National University
Daegu, Republic of Korea
ajaykjha123@yahoo.com)

**Woo Jin Lee**
(School of Computer Science & Engineering, Kyungpook National University
Daegu, Republic of Korea
woojin@knu.ac.kr, corresponding author)

**Abstract:** Being one of the major operating system in smartphone industry, security in Android is paramount importance to end users. Android applications are published through Google Play Store which is an official marketplace for Android. If we have to define the current security policy implemented by Google Play Store for publishing Android applications in one sentence then we can write it as "*all are suspect but innocent until proven guilty*." It means an application does not have to go through rigorous security review to be accepted for publication. It is assumed that all the applications are benign which does not mean it will remain so in future. If any application is found doing suspicious activities then the application will be categorized as malicious and it will be removed from the Play Store. Though filtering of malicious applications is performed at Play Store, some malicious applications escape the filtering process. Thus, it becomes necessary to take strong security measures at other levels. Security in Android can be enforced at system and application levels. At system level Android uses sandboxing technique while at application level it uses permission. In this paper, we analyze the permission-based security implemented in Android through three different perspectives – policy expert, developer, and end user.

**Keywords:** Android, permission, security, analysis, privacy, policy
**Category:** D.4.6

## 1 Introduction

There is a popular proverb "*You can be successful and have enemies or you can be unsuccessful and have friends*." The Android policy matches with the first part of the proverb. The success of Android is largely due to its openness. There are free tools and techniques available for developers to easily create applications. Created applications can utilize the services provided by a large number of third-party applications. Also, created applications can be easily published without going through rigorous censorship. Addition to all of these, Android allows developers to publish applications through third-party store. Though openness is a key element behind success of Android, it invites large numbers of enemies.

According to threat reports by F-Secure in 2014 [FSecure, 14], Android continues to be the favoured target for the majority of mobile malware. The previous reports by the same organization also show that Android alone accounts for 97% of all mobile malware but interestingly only 0.1% of those malware belongs to the official Google Play Store. Even if we take account of only Google Play Store then this 0.1% malware pose high security threat to millions of benign applications available in the Play Store which is around 1.6 million as of July 2015 [Statista, 15]. So, the question is how benign applications can be secured with so many malicious applications roaming around.

In Android, security measures can be taken at various stages. Play Store can take some security measures to filter out malicious applications. Though it's a non-trivial task, Google Play Store is doing this successfully in some extent which is evident from the fact that it has very less malicious applications (0.1%) in comparison to third-party stores. For those malicious applications which escape from the filtering process at Play Store, some measures can be taken at system level to isolate the applications. Android platform successfully uses the sandboxing technique to isolate applications. Under some circumstances such as inter-application communication, the isolation rule does not apply. So finally and most importantly, security measures can be taken at application level for protecting benign application from malicious application in case of contact between them. Application level security measure is the main discussion point of this paper.

In Android, stakeholders for implementing security at application level can be divided into three groups: policy experts, developers, and end users. Policy experts take decisions on security policy and procedure to implement that policy. Developers are primarily responsible for implementing the security policy. End users play key role in executing the security policy.

In this paper, we provide insight into permission-based security in Android through three different perspectives: policy expert, developer, and end user. Not only we provide insight but also we analyse existing research works in each categories. Through this work we are exploring answers to some important questions. What are the skills required by stakeholders for effectively implementing security in Android? How to improve the security in Android applications? Where to concentrate resources for security improvement? We could find only one related work [Tan, 15] which provides survey and taxonomy of Android security. The work performs the taxonomy based on deployment stages of Android application. In contrast to this work, our work focuses only on permission-based security and performs perspective-based analysis. The main goal of our work is not to provide survey on Android security or malware. Our work performs analysis on weakness and limitations of Android application level security. It also suggests and analyses possible solutions available in the research community. Preliminary version of this work has been published as a poster paper [Jha, 15].

Rest of the paper is organized as follows. We briefly describe how the security is enforced in Android in [Section 2]. It mainly focuses on the application level security. In [Section 3, 4, and 5] we analyse the permission-based security with the help of existing research works through policy expert, developer, and end user perspectives respectively. In [Section 6] we discuss the outcome of this work and possible answers

to the aforementioned security questions. We also discuss recent policy changes in Android security. Finally in [Section 7] we conclude the paper.

## 2 Security in Android

Android imposes security at two different levels: system and application. It uses sandboxing approach at system level. In this approach an Android application is executed in virtually isolated environment which means the resources required to execute the application, for example Dalvik Virtual Machine, file system, memory, etc., are granted exclusively. This approach largely prevents benign application from getting harmed by malicious application.

When it comes to the application level, an Android application sometimes need to make hole in the isolation wall created by sandboxing technique. An Android application can access the shared sensitive resources through API and also it can communicate with other Android applications through inter-application communication mechanism. Sharing resources and inter-application communication creates two different kinds of holes in the isolation wall and most of the malicious applications exploit these two kinds of holes to perform malicious activities on benign applications. So, despite the existence of sandboxing technique at system level, Android needs protection at application level. For that purpose it uses permission-based security at application level.

In Android, permission can be thought of as a label which is placed on the sensitive object. If any application has desire to access that sensitive object then the application has to first acquire the label. In Android, there are two types of sensitive objects. The first one is shared system resources or system applications such as contact address, camera, etc. This type of resources are protected by system defined permission. For example, if any application wants to read the contact address then it has to declare "READ_CONTACTS" permission and user has to grant that permission to the application during installation. Second type of sensitive object is application's component. Android applications are built using components and developer can expose a component for third-party application use. The exposed components are highly vulnerable against attacks so they need to be protected. In this case, developer has to define and declare a permission label for the exposed component. Any application willing to access that exposed component must acquire the permission declared by the component. For example, if a component c of an application A defines and declares "MY_PERMISSION" permission then to access the component c other application, for example B, has to acquire the same permission "MY_PERMISSION."

In Android, permission has protection level. There are four kinds of protection levels: normal, dangerous, signature, and signatureorsystem. Normal level permissions are automatically granted to applications since these permissions do not pose serious security threats. Dangerous level permission requires user approval that is user must grant all the dangerous permissions during application installation time. Signature level permissions are only granted to those applications which are signed with the same certificate. SignatureOrSystem level permission are granted to only those applications which are in the system image or signed with the same certificate. More details about Android security can be found in [Enck, 09] and [Shabtai, 10].

## 3 Security Analysis Through Policy Expert Perspective

Core security policy implemented in Android application is not a new technique. It originates from very old technique called Mandatory Access Control (MAC). In this technique a label is assigned to an object and subject must acquire the label to access the object. In Android specific term the label is called permission. Permission-based security is very simple and highly effective if it is implemented properly. With the MAC type core security policy, there are two distinguishing Android specific variations which are mostly discussed in research community. First one is granting permission at installation time and second one is the way the permission is granted that is "all or none."

If an application needs to access some shared system resources or third-party application's component then it must include all the permissions defined by those resources and components in the Android manifest file. At application installation time all the permissions are displayed to an end user and the end user must grant all the permission to complete the installation. Once the installation is complete, the application is free to use any of those resources and components without any restrictions. So, the question is what is the problem with this approach? The main problem is that the developer loses the control of his own application. For example, a malicious application defines the permission for using a component of benign application. A user grants all the permission to malicious application without knowing the nature of the application. Now, the developer of benign application cannot stop malicious application from performing harmful act. To mitigate the problem researchers have suggested to extend the current install-time policy as well as include some run-time policy.

Saint [Ongtang, 12] adds signature and application configuration based policies at install-time whereas in run-time it includes signature, application configuration, and context based policies. In signature-based policy an application grants or denies the permission by default based on the signature of requesting application. Application configuration policy controls permission assignment based on the permission requested and version of requesting application. Context-based policy controls the run-time interaction between applications based on the context such as location, time, battery level, etc. The main idea behind Saint is to strengthen the security of an application by giving it control over which application to communicate with and under what circumstances. It definitely serves its purpose but not without compromising one key business model of Android that is openness. Saint can seriously jeopardize collaborative model (applications utilizing services of other applications openly) if all the developers start restricting their applications use by third-party applications.

CRePE [Conti, 11] only provides context-based run-time policy. In contrast to Saint's context-based run-time policy where policy is written by developer, CRePE's policy is written by user. Saint also provides an option for user to override its policy. It seems much better practice to give final control to user but the major question is whether the user will be interested in either writing or deciding policy.

Another Android specific security policy is "all or none." The main problem with this policy is that the user cannot selectively grant the permission. A user has to either grant all the permission or abort the installation. Also, user cannot revoke the

granted permission unless the user uninstall the application. Apex [Nauman, 10] extends the install-time policy. It allows the user to selectively grant the permission. It also allows the user to make some run-time constraint on the permission. The main problem is that the paper does not mention how an application would perform after denying some of the permissions. Similar to CRePE, it also transfer the burden of writing policy to end users.

## 4 Security Analysis Through Developer Perspective

An Android application can be at security risk under two circumstances: when communicating with third-party applications and when accessing shared system resources. The inter-application communication can happen in both direction that is an application can access component of third-party application and vice-versa. In the former case, the application can leak sensitive data to third-party application while in later case third-party application can perform harmful act on the application. While there is some-level of permission-based protection for later case, there is hardly any protection mechanism defined for leaking sensitive data. Developers must take security measures on their own but most of the time developers are pre-occupied with delivering the product in time so they fail to take any security measures. In this direction, research community is mostly focusing on the common mistakes made by developer and how those mistakes can be mitigated.

A developer can expose an application's component for third-party use. This feature is a key element in Android success. The exposed component must be protected with permission but the developer may accidently leave the component unprotected. In this circumstance, malicious application can easily perform harmful act on the application through exposed component. Researchers have proposed some tools such as COMDROID [Chin, 11] and ICCMATT [Kumar, 15] to assist developer in finding out exposed and unprotected components. These tools can not only find out the exposed components but also find out other vulnerabilities in inter-component communication (ICC). In Android, intra- and inter-application communication are commonly referred as inter-component communication. Analysing ICC is extremely important for asserting security in Android. In addition to aforementioned tools, developer can use some other tools like Epicc [Octeau, 13] and IC3 [Octeau, 15] for general purpose security analysis of ICC. Epicc maps ICC among applications. By doing so, it helps in finding security risk communications. For example, if an application communicates with another application implicitly then that communication can be intercepted by malicious applications. Epicc can find all the applications including malicious applications which can intercept a particular communication thus exposes the security risk communications. IC3 is more precise version of Epicc. In addition to security vulnerabilities, the exposed components can cause reliability issues. In Android, most of the ICC happens through Intent which is a message passing technique. An application can send a purposely constructed intent to the exposed component causing the application to crash. Maji et al. [Maji, 2012] studied this behaviour and found several components vulnerable.

Even if a developer protects exposed components with permission, there is a situation called privilege escalation attack [Davi, 11] [Felt, 11a] in which malicious act can be easily performed on the application. In privilege escalation attack, an under

privileged application uses the privileged application for accessing sensitive resources. It is also referred as confused deputy attack because the attacker uses the privileged application as deputy to access sensitive resource. Felt et al. has proposed IPC Inspection technique [Felt, 11a] to prevent such attack. Their core idea is to reduce the privilege of a recipient application to the intersection of the recipient's and requester's application's permission at run-time. Covert [Bagheri, 15] performs compositional analysis on a composite formal specification of applications to detect privilege escalation attack. Its idea is to perform compositional analysis whenever a new application is installed by a user. It does not provide any defence mechanism. To prevent privilege escalation attack, changes should be made at policy level such as Saint's configuration based policy [Ongtang, 12] can prevent such attack more effectively. Though a developer does not have any option to prevent such attack, it is strongly advised not to expose components unnecessarily.

To access sensitive shared resources or components of third-party applications, a developer has to define permissions in the manifest file manually. Developers sometimes provide more permissions than required by the application [Felt, 11b] which can ultimately lead into some security problems. First, end users may feel suspicious about the application and abort the installation. Second, the application will be more likely to become accessory (deputy) for privilege escalation attack. Some tools such as Stowaway [Felt, 11b] and PScout [Au, 12] can solve the problem by finding out the permissions which are not required by the application. We believe that the problem should be solved more effectively at policy level by making the permission entry procedure automated.

The most important security problem which has been left unattended by policy maker is privacy leak. Privacy leak happens when the sensitive data outflows to third-party applications or system. There is absolutely no direct protection against sensitive data leak in Android application at policy level. Most of the time privacy leaks are performed by malicious applications which is deliberately done by a developer. Sometimes a developer may accidently leave the application in a leak state which can seriously harm the reputation of the application. In the latter case, the sole responsibility lies on the developer to protect the privacy leak. Large number of researchers are working on detecting privacy leak and they have produced some prominent tools such as TaintDroid [Enck, 14] and Flowdroid [Arzt, 14]. Most of the currently available privacy leak detection tools work on taint propagation mechanism in which the propagation of tainted source is monitored for leak. TaintDroid and Flowdroid are complementary to each other in the sense that they perform dynamic and static taint propagation respectively.

One challenge, which still remains unsolved, is to effectively detect privacy leak in ICC. TaintDroid performs the taint propagation in ICC which occurs only through Binder interface. Both TaintDroid and Flowdroid fail to perform taint propagation in Intent based ICC. The main reason behind failure is that the Intent breaks the data flow path. In other word, we can say that it is difficult to map the data flow path when Intent is involved. Intents are classified into explicit and implicit intents based on whether it defines target component name or not. While it's little bit easier to map the path from one component to another component in case of explicit intent, it's extremely difficult to map precisely in implicit intent. Both Epicc and ICCMATT tools exactly perform this mapping task but they don't perform privacy leak detection.

IccTA [Li, 2015] solves the privacy leak detection problem in ICC. It leverages the existing tools Epicc and Flowdroid to perform taint propagation in ICC.

Enck et al. [Enck, 11] performs source code analysis on large numbers of android application to access the security properties in general. They report several critical security findings which are useful for android developers.

## 5 Security Analysis Through End User Perspective

Security policy maker of Android has handed key roles to an end user. An end user has the sole responsibility of making final security decision. A developer implements permission inside the application and during application installation all the permissions are shown to end users. Based on those displayed permissions, a user has to decide whether the application is benign or malicious. At a first glance it seems huge policy mistake to let the user decide application's nature by merely viewing permissions used by the application but it is not a mistake. The key point is that the user is not the sole stakeholder. User is merely a part of the whole process. It's like water treatment plant. Water gets filtered at several layers and finally reaches to user and user has sole responsibility to either drink the water or throw it just by scanning through bare eyes. The ultimate goal is to filter the application at several layers. Including end user as a security layer to filter applications seems good decision but the major question is whether the layer is effective in its job. Simply adding several security layers won't do anything unless those layers are performing their tasks. In this direction, researchers have done surveys which we will discuss here.

Felt et al. performed a survey [Felt, 12] to mainly evaluate user's attention and comprehension towards permission. Through survey they checked whether the end user pay any attention to Android permission before installing an application. They also checked whether the user understands how those permissions correspond to an application privilege. They observed very low percentage of attention. Most of the users are completely unaware of permission. Even the users who are aware of permission did not pay any attention. They also observed very low percentage of comprehension. The users who noticed permission during installation performed better in comprehension than the other users. They conclude that the majority of Android users do not pay attention to or understand permission warnings. We observed some interesting facts in their work. First, they found that the user who installs application from Google Play Store has significantly more understanding of permission than the user who installs from third-party store. Second, they found that the users are not aware of security implications. For example, in one case, user did not know that the SMS can be sent by an application without the user consent. Given these two facts, we believe that the users will pay more attention to permission if they are aware of security implications.

Kelley et al. performed a smaller scale survey [Kelley, 12] than the Felt et al in [Felt, 12]. The objective of the survey was same that is to find out whether end users read and understand the permission. They also studied how end users perceive the security implications. In study they found that although the end users generally read permission, most of them don't understand the permission. They also found that the most of the users are not aware of security implications.

Both the surveys in [Felt, 12] and [Kelley, 12] clearly indicate that end users are

currently not highly effective in their share of work as one of the security stakeholder in Android. Given the short history of Android and limited awareness about security implications, survey's results do not seem highly surprising. The survey results do not indicate that the current security policy for end user is a complete failure. As Felt et al. mention that it is currently neither a failure nor a success because some of the end users (20% of laboratory participants) in their survey read and understand the permission. The percentage is likely to increase with time and awareness. As we have discussed before, the ultimate goal is to add another layer of security as end user. So, whatever end users do to take security measures is going to add in the security measures already taken at other layers by other stakeholders.

Lin et al. in [Lin, 2012] studied the user's mental models of mobile application privacy through crowdsourcing. They studied the expectations of end users in terms of permission required by an application according to its functionalities. They also studied whether the end users can correlate between functions of an application and its permission requirement during install-time. If they can't correlate then what are their reactions against the permission used by the application. Out of 100 most downloaded applications at that time, they found 18 applications in which user has less than 20% expectation of a particular permission. It means those 18 applications have at least one permission which is highly unexpected by the user. Obviously, the users could not explain the reason behind the unexpected permission used by those 18 applications and they reacted negatively. Most importantly, comfort level of users declined against those applications. Lin et al. concluded that users feel more comfortable when they are informed of the reasons behind sensitive resources use. Currently, Android does not support the mechanism of informing reasons behind sensitive resource use. Even if, in future, Android supports this mechanism then it will be very difficult to implement correctly. It is unlikely that the developer of malicious application will display the correct reasons behind the sensitive resource use. At the same time, it will be difficult if not impossible for Play Store to assert the stated reasons. Certainly, the benign application will get benefit from this mechanism but it seems non-trivial task. Lin et al., in the same work, proposed a solution. Their solution provides a privacy summary interface which, in addition to displaying permission, displays the user perception about the sensitive resources use. The value of user perception is derived through crowdsourcing. The solution is similar to current review system included in Android, except that privacy user summary focuses only on privacy. As mentioned in [Felt, 12] and [Kelley, 12], the users are currently looking at review for privacy decisions too.

## 6 Discussion

In this section we will discuss three important questions. First, what are the skills required by stakeholders for effectively implementing security in Android. Second, how to improve the security in Android applications? And third, where the resources need to be concentrated for improving security? In addition to above questions, we will discuss recent changes made by Android in permission policy.

Policy experts must understand which business model works for them. For example, they cannot compromise on openness so the security and openness must symbiotically coexist in Android. They also need to thoroughly understand the

advantages and disadvantages of the technique which will be used for security implementation. Finally they need to understand the user's adaptability and friendliness of the technique being used. Another key stakeholder is developers who actually implement the security in the application. Developers must understand the security policy and procedure thoroughly. More importantly, they also must understand the implications of not implementing the security properly. Finally, we have end users as a key stakeholder. All the hard work done by policy expert and developer can be compromised if the user does not understand the procedure of securely using the application. End users must understand the security use policy and its implications. We have come up with an inside-out model for skills required by three different stakeholders which is shown in [Fig. 1].

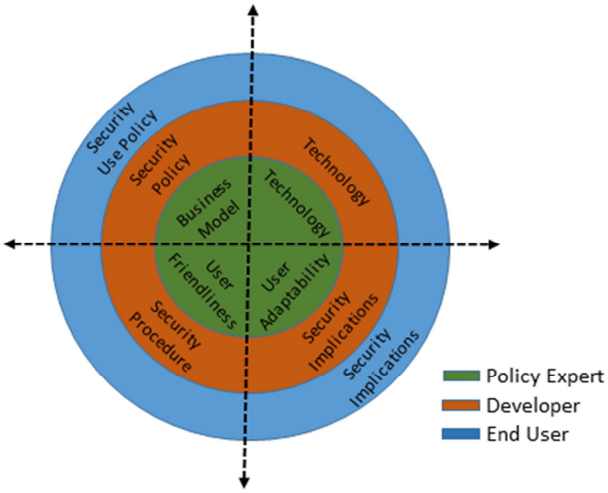


*Figure 1: Inside-out model for skills required by security stakeholders*

There are many ways to improve security in Android applications. There should be clear separation of responsibility among policy expert, developer, and end user. For example, writing policy should be left to policy maker rather than to developer or end user because developer and end user are not often security experts. Developer should be given short-term training about the security measures and its implications. Most importantly, developer should be furnished with automated tools for security analysis. From surveys in [Felt, 12] and [Kelley, 12], it is clear that resource-centric coarse-grained permission model is confusing for end users. End users will be more likely to understand feature-centric fine-grained permission. For example, most of the users in those surveys understood "READ_CONTACTS" permission. All the surveys indicate that the end users are not aware of security implications so they should be informed about the security implications.

Resources can be assigned based on long-term and short-term security goals. Making changes in policy is highly complex and non-trivial task. Policy changes also comes with certain business risks. It does not mean that the policy should not be changed at all. There should be some level of changes in policy according to new

challenges. Policy change is a long-term goal and resources should be assigned accordingly. Most of the resources should be focused on developing automated tools which can assist developer. As we have mentioned before, developer is less likely to take the security measure on their own but if they are provided with automated tools then they would be likely to take the required security measures.

In line with the some of the suggestions made by researchers in their works, Google has made some changes in the permission policy in Android. Mainly, the changes have been made to help the end users in understanding what an application will have access to. As we have already discussed, end users are currently finding it hard to understand the permission. User interface (permission screen) for Dropcam application is shown in [Fig. 2]. Left one is old user interface while right one is new user interface. As we can see in the figure, permissions in the old categories have been rearranged into new and less number of groups. When user taps on the group, it displays a short description about the group and the list of permissions in that group. Most importantly, permissions are not displayed in technical terms as used to be but they are now displayed in more general terms. These changes are good for end users but there are some changes which got strong reactions from developer as well as user communities.

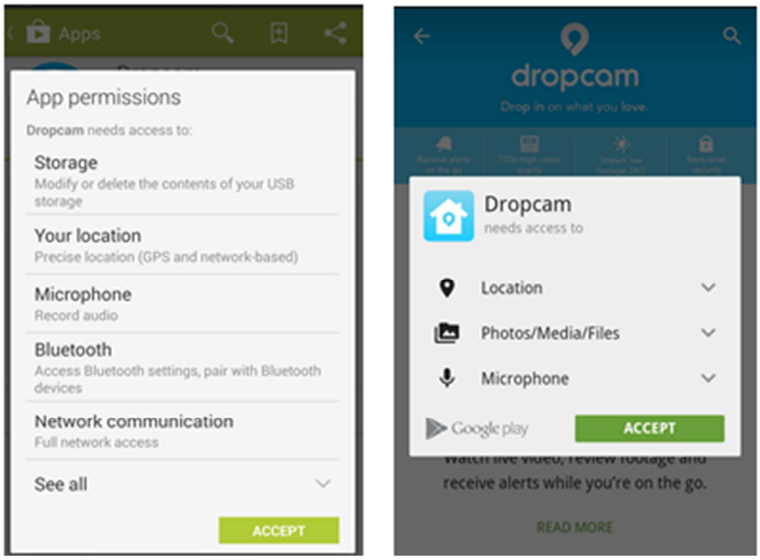


*Figure 2: Permission screens for Dropcam. Left - old screen. Right - new screen.*

The permissions which are requested by most of the applications (common permissions) such as Internet has been placed in the group named *Other* which is not shown to users during installation. Some of the permissions included in *Other* group are very sensitive. For example, Dropcam includes a permission which can prevent phone from sleeping. Malicious applications can exploit this behaviour especially against novice users. Though Google Play Store has provided an option to check complete list of permission including other by going to developer section and

selecting permission details, novice users may not do so due to primary focus on installation. Having discussed the vulnerabilities, we need to focus on the fact. The surveys in [Felt, 12] and [Kelley, 12] indicate that the novice users are not paying much attention to the permission so in terms of ground reality it seems correct policy decision.

While shrinking the number of groups, large number of related permissions have been included in a group. For example SMS group has permission to receive, read, edit, and send text message including some other permissions. New policy allows an application to update its permission set without the consent of user if the new permissions belongs to the groups which have been granted by user during first-time installation. For example, an application initially displays only read text message permission and user grants that permission. Now, if that application wants to add more permission from SMS group such as edit or send text message then it can do so without informing the user. This can cause some serious security problems like malicious applications can send text message to premium rate numbers. Sanders et al. discusses the implications of new auto-update policy in [Sanders, 15]. Though Google has provided an option to user for disallowing the auto-update feature, the whole mechanism seems to be risky proposition for end users.

Google has recently announced one major change in Android permission-based security. It has abolished the install-time permission starting from Android M SDK. Android will no longer display the permission screen as shown in [Fig. 2] during application installation. Instead, it will display permission dialog during runtime. When an application tries to access any sensitive resource which requires permission, Android will display a dialog with that permission prompting user to allow or deny the permission. For example, left screen in [Fig. 3] shows the install-time permission screen displayed by Hangout application whereas right screen shows the runtime permission dialog displayed by the same application.

Unlike install-time permission where a user has to make security decision without knowing context, runtime permission allows the user to make security decision based on the real context. For example, SMS permission dialog will be displayed to the user only when application sends SMS. In this case, the user will have good understanding on why the application is requesting the permission. Runtime permission will certainly improve the security and privacy issues. For example, it will prevent or at least control the security vulnerability where malicious applications used to send SMS or make phone calls to premium rate numbers silently. This vulnerability is one of the major contributor in financial loss in Android devices. However, one major concern in runtime permission is user fatigue. It is well known fact that a user develops fatigue in interaction with the interface. In the process user makes decisions which are not well thought. This ultimately can lead into runtime permission which is ineffective in curbing security and privacy issues in Android applications.

Google has carefully engineered the permission dialog display system which can somewhat prevent the user fatigue. When an application accesses a resource which requires permission for the first time, Android displays a runtime permission dialog without the “don’t ask again” checkbox. If the user allows the permission then subsequent request to the same resource will not display any permission dialog. If the user deny the permission then the subsequent request to the same resource will

display the permission dialog containing the checkbox as shown in the right screen of [Fig. 3]. If the user selects the checkbox and then deny the permission again then the subsequent request to the same resource will not display any permission dialog. However, the user will be able to revoke the decision by going through settings.

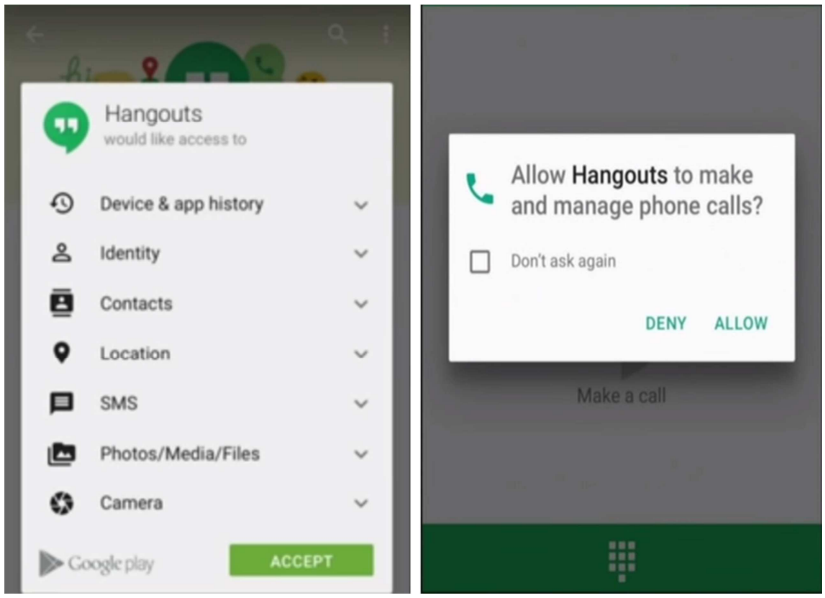


*Figure 3: Install-time and runtime permission screens.*

Legacy applications which target Android SDK older than M will still be displaying permission screen at install-time but unlike existing install-time permission solution where a user has no option to selectively grant the permission, a user has now option to perform selection on permissions. This feature is available not only for legacy applications but also for new applications which target Android M SDK. For legacy applications running on Android M SDK device, user still has to grant all the permissions at install-time but after installing the application user can selectively turn off and turn on the permissions of that application. To perform this task user has two options. User can open the permission screen of any particular application through settings and then can turn off or turn on any permission displayed by the application as shown in left screen of [Fig. 4]. Another options is to open the vertical view. In this case, a user can view all the installed applications which are using a particular permission. For example, right screen of [Fig. 4] shows all the installed applications which are using Camera permission. User can turn off or turn on Camera permission from any installed applications using this screen.

User experience is one of the major concern when it comes to selectively denying permissions. If a user deny a permission requested by the application then the user will no longer avail the feature related to that permission. Instead the user will be served with empty state or some warning. Viewing empty state or warning is certainly not a pleasant experience when the primary focus is on performing a certain task. The

situation can ultimately lead into rejection of the application by users. Developers can avoid this situation in some of the cases. Instead of directly using permissions, certain tasks can be performed by using Intents. For example, Intent can be used to capture an image through camera. So, if full control of camera is not required by an application then the best option is to use Intent.

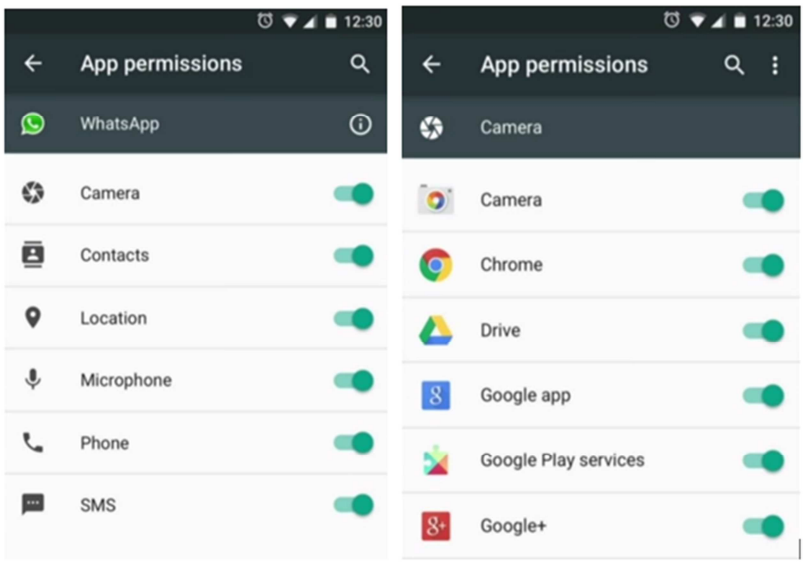


*Figure 4: Permission screens after application installation.*

Though runtime permission system seems better option at least from existing research works, only future will tell how much impact it will create in curbing security and privacy issues in Android applications. Also, it will be interesting to know the impact of user fatigue and user experience on runtime permission system.

## 7 Conclusions

In this paper, we have analysed the permission-based security in Android through three different perspectives: policy expert, developer, and end user. We have mostly analysed and discussed the major issues or weakness which came across several studies of permission-based security. We also analysed and discussed the major changes in permission-based security in Android. There are limited research works on Android security policy. Changing policy and implementing new policies are not trivial tasks. Organizations are often reluctant to do that due to business risks associated with policy change. Most of the works on policy level advocate that the run-time fine-grained policy should be included in the current install-time coarse-grained policy. In line with these research works, Android permission has been shifted from install-time to runtime. Research works on policy also focus on handing policy making power to developer or user which needs to be carefully discussed more

because current research works suggest that neither developer nor user has much understanding about Android security. Developers are the ones who should be encouraged to develop secure applications. At developer's level, security can be implemented effectively and efficiently with minimum cost. Developers can be encouraged by facilitating them with various automated security analysis tools. There are many research works on preventing and detecting malicious attack but still there is a lack of practical tools which can be used by developers to produce secure applications. More works need to be done on developing practical automated security analysis tools. End users as a last layer of security is a good concept but the layer must be effective in its tasks. Currently, end users have very little understanding about permission-based security and its implications. End users need to be educated about the applications security and its implications to get them effective as a last layer of security. Google is currently experimenting with Android application level security. We firmly believe that the experiment will not succeed in using end users as one of the security layer unless the users are completely aware of security and its implications which is currently too far from its target. Shifting from install time to run time permission seems good move at least from existing research works perspective but it will be too early to come to any conclusion.

**Acknowledgements**

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. NRF-2014R1A1A2058733), the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the C-ITRC (Convergence Information Technology Research Center) (IITP-2015-H8601-15-1002) supervised by the IITP (Institute for Information & communications Technology Promotion), and IITP grant funded by the Korea government (MSIP) [No. 10041145, Self-Organized Software platform (SoSp) for Welfare Devices].

## References

[Arzt, 14] Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., ... & McDaniel, P.: "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps"; In ACM SIGPLAN Notices 49, 6 (2014), 259-269.

[Au, 12] Au, K. W. Y., Zhou, Y. F., Huang, Z., & Lie, D.: "Pscout: analyzing the android permission specification"; In Proceedings of the 2012 ACM conference on Computer and communications security, 217-228.

[Bagheri, 15] Bagheri, H., Sadeghi, A., Garcia, J., & Malek, S.: "Covert: Compositional analysis of android inter-app permission leakage"; IEEE transactions on Software Engineering 41, 9 (2015), 866-886.

[Chin, 11] Chin, E., Felt, A. P., Greenwood, K., & Wagner, D.: "Analyzing inter-application communication in Android"; In Proceedings of the 9th international conference on Mobile systems, applications, and services (2011), 239-252.

[Conti, 11] Conti, M., Nguyen, V. T. N., & Crispo, B.: CRePE: "Context-related policy enforcement for Android"; In Information Security (2011), 331-345. Springer Berlin Heidelberg.

[Davi, 11] Davi, L., Dmitrienko, A., Sadeghi, A. R., & Winandy, M.: "Privilege escalation attacks on android"; In Information Security (2011), 346-360. Springer Berlin Heidelberg.

[Enck, 09] Enck, W., Ongtang, M., & McDaniel, P.: "Understanding android security"; IEEE security & privacy, 1 (2009), 50-57.

[Enck, 11] Enck, W., Octeau, D., McDaniel, P., & Chaudhuri, S.: "A Study of Android Application Security"; In USENIX security symposium, 2 (2011), 2.

[Enck, 14] Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B. G., Cox, L. P., ... & Sheth, A. N.: "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones"; ACM Transactions on Computer Systems (TOCS), 32, 2 (2014), 5.

[Felt, 11a] Felt, A. P., Wang, H. J., Moshchuk, A., Hanna, S., & Chin, E.: "Permission Re-Delegation: Attacks and Defenses"; In USENIX Security Symposium, 2011.

[Felt, 11b] Felt, A. P., Chin, E., Hanna, S., Song, D., & Wagner, D.: "Android permissions demystified"; In Proceedings of the 18th ACM conference on Computer and communications security (2011), 627-638.

[Felt, 12] Felt, A. P., Ha, E., Egelman, S., Haney, A., Chin, E., & Wagner, D.: "Android permissions: User attention, comprehension, and behavior"; In Proceedings of the Eighth Symposium on Usable Privacy and Security (2012), 3.

[FSecure, 14] Whitepapers, https://www.f-secure.com/en/web/labs_global/whitepapers, 2014.

[Jha, 15] Jha, A. K., Lee, S., & Lee, W. J.: "Permission-based security in android application: from policy expert to end user"; In Proceedings of the 2015 Conference on research in adaptive and convergent systems (2015), 319-320.

[Kelley, 12] Kelley, P. G., Consolvo, S., Cranor, L. F., Jung, J., Sadeh, N., & Wetherall, D.: "A conundrum of permissions: installing applications on an android smartphone"; In Financial Cryptography and Data Security (2012), 68-79. Springer Berlin Heidelberg.

[Kumar, 15] Kumar, A., Lee, S., & Lee, W. J.: "Modeling and test case generation of inter-component communication in android"; In Proceedings of 2nd ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft) (2015), 113-116.

[Li, 2015] Li, L., Bartel, A., Bissyande, T. F. D. A., Klein, J., Le Traon, Y., Arzt, S., ... & McDaniel, P.: "IccTA: detecting inter-component privacy leaks in android apps"; In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE 2015).

[Lin, 2012] Lin, J., Amini, S., Hong, J. I., Sadeh, N., Lindqvist, J., & Zhang, J.: "Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing"; In Proceedings of the 2012 ACM Conference on Ubiquitous Computing, 501-510.

[Maji, 2012] Maji, A. K., Arshad, F., Bagchi, S., & Rellermeyer, J. S.: "An empirical study of the robustness of inter-component communication in Android"; In Proceedings of 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2012, 1-12.

[Nauman, 10] Nauman, M., Khan, S., & Zhang, X.: "Apex: extending android permission model and enforcement with user-defined runtime constraints"; In Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security (2010), 328-332.

[Octeau, 13] Octeau, D., McDaniel, P., Jha, S., Bartel, A., Bodden, E., Klein, J., & Le Traon, Y.: "Effective inter-component communication mapping in android with epicc: An essential step towards holistic security analysis"; In USENIX Security 2013.

[Octeau, 15]Octeau, D., Luchaup, D., Dering, M., Jha, S., & McDaniel, P.: "Composite constant propagation: Application to android inter-component communication analysis"; In Proceedings of the 37th International Conference on Software Engineering (ICSE), 2015.

[Ongtang, 12] Ongtang, M., McLaughlin, S., Enck, W., & McDaniel, P.: "Semantically rich application-centric security in Android"; Security and Communication Networks, 5, 6 (2012), 658-673.

[Sanders, 15] Sanders, C., Shah, A., & Zhang, S.: "Comprehensive Analysis of the Android Google Play's Auto-update Policy"; In Information Security Practice and Experience (2015), 365-377. Springer International Publishing.

[Statista, 15] Number of apps available in leading app stores, http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/ 2015.

[Shabtai, 10] Shabtai, A., Fledel, Y., Kanonov, U., Elovici, Y., Dolev, S., & Glezer, C.: "Google android: A comprehensive security assessment"; IEEE Security & Privacy 2 (2010), 35-44.

[Tan, 15] Tan, D. J., Chua, T. W., & Thing, V. L.: "Securing Android: A Survey, Taxonomy, and Challenges"; ACM Computing Surveys (CSUR), 47, 4 (2015), 58.