International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-8 Issue-6, August, 2019

Implementing ASBP: A Novel framework for Sanitizing Android Apps.

Akshay Bhardwaj, A J Singh

Abstract*: Paper The advent of Android as a mobile OS has revolutionized the way we perceive life today. It has permeated now from handhelds to becoming an inclusive and persistently increasing member of our day to day lives. We are increasingly becoming dependent on technology in our everyday lives, insomuch that it is impossible to envision the world without it. Android is at the forefront of this revolution. It has practically rendered the Personal computer useless and has shown the world a new way of transacting on the go using our smartphones and tablets. However, with all the facilities, there comes the shadow of insecurity and data privacy compromise. This is an area which requires deeper introspection and research because it is required to ensure seamless and hassle-free operations in our day to day activities. The current research tries to delve deep into the innards of Android to come up with some plausible solutions to assuage somewhat the wounds that thee intentional leaks of the operating system can give. In this paper, in the successive sections, we propose a framework which would help us in removing the identified vulnerabilities that afflict Android, namely third-party apps and vendor customized ones. We would also propose a set of guidelines for improving the Permissions model. We call our framework, "The ASBP Framework." (App Sanitization and Better Permissions).

Keywords:Android,ASBP,framework,Malicious apps,Permissions,security

I. INTRODUCTION

Mobile technologies have transformed every facet of life. Google's Android is the most well-known cell phone Platform, running on 52.5% of all cell phones, and with more than 10 billion applications downloaded from the Market. Android takes an open-publish approach to application dissemination, in which any application is installed on any telephone. To help address security concerns, Android ensures access to delicate resources, including the Internet, GPS, and telephony with consents. At the point when an application is introduced, the consents it solicits appear to the client, who then chooses whether to continue with the installation. No extra authorizations might be gained when an application runs. While Android authorisations give a vital level of security, more accessible permissions are given than should be expected. For instance, the Amazon shopping application must get full Internet authorization, enabling the application to send and get information from any webpage on the Internet, not simply www.amazon.com. This authorization permits applications to associate with neighbourhood attachments on the telephone also, which prompted to an as of late promoted security opening whereby any application with Internet permission could get to definite framework sign on HTC Android telephones.In this paper, in the successive sections, we propose a framework which would help us in: Removing the identified vulnerabilities that afflict Android, namely third-party apps and vendor customized ones. We would also propose a set of guidelines for improving the Permissions model.
We call our framework, "The ASBP Framework." (App Sanitization and Better Permissions).

II. CURRENT SCENARIO

Dangerous permissions

These cover areas at which the program wants resources or data which demand an individual's private info, or could potentially impact the user has stored data or the operation of different apps.Special permissions
There are two or three permissions that do not behave as dangerous and normal permissions. SYSTEM_ALERT_WINDOW along with WRITE_SETTINGS are especially sensitive, so most programs should not utilize them. If an app needs one of these permissions, then it has to declare the permission from the manifest, and ship an intention asking the consumer's authorization.

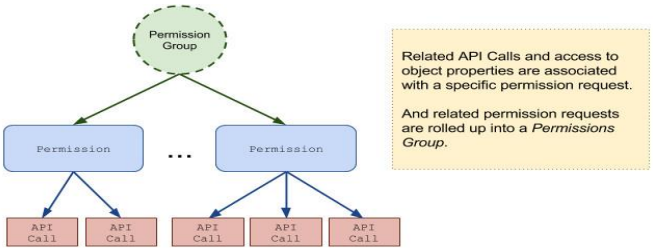


Fig 2.1 Current permissions model

III. SET OF GUIDELINES FOR THE PERMISSION MODEL IN ANDROID

After doing extensive and exhaustive research, [1].[2],[3],[4],[5],[6],[7], we have concluded that Android permission scheme/model right since its inception has been undergoing transformations and is continuously changed depending upon the user suggestions and the developer's suggestions and the platform owner's business decisions.

Revised Manuscript Received on October 30, 2019.
* Correspondence Author
Akshay Bhardwaj*, Department of Computer Science, HP University, Summerhill, Shimla 171005,India. Tel: +919418482826, e-mail: akshay117@gmail.com
A J Singh, Department of Computer Science, HP University, Summerhill, Shimla 171005,India. Tel: +919418484855, e-mail: aj.hpucs@gmail.com.

Since Android came into existence, we have studied that there have been various kinds of changes in the type and taxonomy of permissions which have been implemented with subsequent revisions. However, the most important and radical change came with the advent of Android 6.0 (Marshmallow). Earlier the users had to accept the permissions all at install time, or the application would not install. Marshmallow changed this by introducing a Runtime based permissions model which allowed the users to choose what permissions the applications that they installed on their mobile devices had access to. This allowed users to have more control over their choices to allow/deny certain permissions. A study regarding this was done by us, which vindicated our stand that Android required Granular permissions. In Android after M permissions have been a welcome step forward and bring a few controls to users. It is like a boxer with a glass jaw, there is still a lot to be done, but it is a trade-off between convenience/simplicity and privacy controllers. These Controls are a win-win -- provide several privacy controls, curb the most nefarious offenders; however, at the same time, softly allow developers to gather more aggregate analytics. The developers gain, the consumers gain something Google gains that the maximum, Now you could state it is sacrificing privacy, to increase security. However, unlike IOS, it can be circumvented.From the preceding paragraph, the gravity of the situation becomes quite apparent. Therefore, we propose a set of guidelines for the permission model of Android:

1. There will be no Permission Groups. Permissions shall be Granted and Revoked individually.
2. INTERNET permission shall be labeled as Dangerous.
3. When an App is running in the background, it shall cease to hold control over any resources that it may have access to.
4. No Pending Intents should be allowed by the system.
5. Third party or user-defined permissions will not be treated at par with the system defined permissions.
6. Each of the dangerous permissions can be and wherever possible should be broken down into further fine-grained ones.
7. In addition to the existing dangerous permissions in Android classification, the following permissions would also be included and labeled as dangerous:

android.permission.ACCESS_LOCATION_EXTRA_COMMANDS
android.permission.ACCESS_NETWORK_STATE
android.permission.ACCESS_WIFI_STATE
android.permission.ACCESS_WIMAX_STATE
android.permission.BROADCAST_STICKY
android.permission.CHANGE_NETWORK_STATE
android.permission.EXPAND_STATUS_BAR
android.permission.FLASHLIGHT
android.permission.GET_ACCOUNTS
android.permission.GET_PACKAGE_SIZE
android.permission.GET_TASKS
android.permission.KILL_BACKGROUND_PROCESSES
android.permission.MODIFY_AUDIO_SETTINGS
android.permission.PERSISTENT_ACTIVITY
android.permission.READ_SYNC_SETTINGS
android.permission.READ_SYNC_STATS
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.REORDER_TASKS
android.permission.RESTART_PACKAGES
android.permission.SET_TIME_ZONE
android.permission.SET_WALLPAPER
android.permission.SET_WALLPAPER_HINTS
android.permission.TRANSMIT_IR
android.permission.VIBRATE
android.permission.WAKE_LOCK
android.permission.WRITE_SETTINGS
android.permission.WRITE_SYNC_SETTINGS
android.permission.WRITE_USER_DICTIONARY
android.permission.BLUETOOTH
android.permission.BLUETOOTH_ADMIN
android.permission.CHANGE_WIFI_MULTICAST_STATE android.permission.CHANGE_WIFI_STATE
android.permission.CHANGE_WIMAX_STATE
android.permission.DISABLE_KEYGUARD
com.android.launcher.permission.INSTALL_SHORTCUT
android.permission.INTERNET
android.permission.NFC
android.permission.READ_INSTALL_SESSIONS
android.permission.READ_PROFILE
android.permission.READ_SOCIAL_STREAM
android.permission.READ_USER_DICTIONARY
com.android.alarm.permission.SET_ALARM
com.android.launcher.permission.UNINSTALL_SHORTCUT android.permission.USE_FINGERPRINT
android.permission.WRITE_PROFILE
android.permission.WRITE_SOCIAL_STREAM

For our discussion permissions model does not only imply the nomenclature and taxonomy of Permissions in Android but also refers to any security aspect which stems from the misuse of these permissions by malicious apps.These general guidelines are proposed after an extensive study of literature that we have done regarding the permissions model of Android as is amply demonstrated in the previous papers. We further propose that these new guidelines/framework would have maximum impact on only dangerous permissions. This is because although our theoretical framework is exhaustive, our literature survey and our research, as demonstrated in the preceding papers put on us a limitation. We shall only limit ourselves to implementing a prototype application only for INTERNET (because we deem it to be the strongest of the candidates for malware attacks).

IV. LOGIC AND THOUGHT BEHIND PROPOSED GUIDELINES

The following is the justification behind the salient points of the Guidelines:

1. In case permission groups are to be retained for the sake of user simplicity and reduction of complexity. Still, the individual app permissions will be given to the specific app that requests it.
2. The other permissions shall remain unaffected and will have to be explicitly asked for. (The concept of permission groups though was visualized as an effort to club similar permissions into a single entity. Unfortunately, it has opened up a new world for malware because when one permission is granted to an application from a group, then the other permissions from that group are implicitly granted to the application)

3. The INTERNET permission has been removed from the list of dangerous permissions and is now normal permission that does not require user intervention and is granted by default. (possibly this is because Google works on an Ad revenue-based model or concept).
4. For example, when an app goes into the background in a cached state, it cannot operate outside of its sandbox.
5. A Pending Intent specifies an action which would happen in the future. You pass a future intent to another application and allow that application to execute that Intent; as if it had the same permissions as your application, whether or not your application exists when the Intent is eventually alled upon. This is a very dangerous situation as the permissions requested through the Intent will remain available in the system even after the original app is finished.
6. Apps can define their custom permissions and request custom permissions from other apps by defining <uses-permission> elements. Android treats these just like system permissions, which is dangerous as there is no way of finding out if the app developer's intentions are true or not.
7. If any apps are found indulging in nefarious activity by misusing the permissions model, then they should be broken down into finer grains and could alternatively be used to provide fake information to the malapp. We propose that all the permissions outlined above can be used to provide for fine-grained versions. This is very much feasible and possible. All it would require would be the creation of a layer/library that would hold the definitions and actions corresponding to the new permissions which would not replace the platform ones but would instead provide a pathway through which apps would interact with the platform ones.
8. As an example, without the prompting, ALL programs can read your Google-registered Gmail address, nearby Wi-Fi networks, currently connected Wi-Fi network, find what other reports have been installed on your apparatus, launching at boot, change your wallpaper/timezone, and more.Even after Android M's new permissions with ALL locked down and there are no alarms to the consumer. This is a big problem.

V. A VIEW AT GRANULARITY OF PERMISSIONS

We thought to classify permissions based on the kind of resource that was either to be protected or accessed.

Category 1: Outside Access Our category contains those Android permissions that enable access, for example, Web access, sending and receiving text messages, and writing and reading external storage. The particular resource naturally parameterizes all these permissions. By way of example, Internet access involves specifying an Internet domain, text messages are sent to a phone number in a certain field code, and card access calls for specifying a directory or file name. Thus, there are two natural methods of fine-grained variations with this category: having a whitelist of allowed resources (domains, domain name, directories, etc.), or even a blacklist of forbidden resources. We could even combine both the methods to achieve an optimum solution. Within our framework, we have selected to focus on access to the Internet, which is pervasive across applications and maybe especially dangerous. We developed a Fine-grained, Black Listing permission InternetURL(Id), which allows network links just to domain d and its Subdomains. Notice that this consent does not remove the requirement target domain d to some degree. However, it will help ensure that app vulnerabilities cannot be tapped to contact it and malicious websites.

Category 2: User Data Our second category contains those Android permissions that access structured user data, such as an individual's calendar, contact list, and accounts information. For these permissions, we can present variations that leverage the structure to provide access to a subset of the info.

Category 3: Sensors Our category contains those permissions that protect access to sensors on the phone, including the camera, GPS receiver, and microphone.

Category 4: Settings Our fourth category contains those permissions that provide access to settings and state information on your telephone. An average of each permission that is such provides access to browse or update unrelated bits of advice. As we see from the preceding discussion, it is quite easy to introduce finer-grained variants of permissions into the sandbox framework that we are proposing. This, however, is one aspect of our framework. The other aspect would include instrumenting the application so that it can make use of the fine graining of permissions that we have just described. Specifying such permissions model would be an exhausting task; hence, for the sake of our discussion, we include only the prominent ones. INTERNET as inferred from the previous paragraph. As is very much understandable from the preceding discussion, we need to condense the permissions into Granular ones, which can be done as follows: By examining the software within our 1100-application sample we identified 11 permissions that may bring about the disclosure of 12 kinds of sensitive information: location, phone_state (granting access to call number & distinctive device ID information types as well as call state),contacts, user account information, camera, mic , browser history & Folders, logs, SMS messages, calendar, along with subscribed feeds. We quantified the prevalence by which applications required each permission by parsing the applications' manifests with the publicly available Android APKtool. We find that 605 applications (55 percent) require the use of one of the resources and access to the Internet, leading to the potential for undesirable disclosure. This essentially means we can have 11 different kinds of permissions which are holding the key to 12 sensitive tools.This is a pointer as to where we should focus next. We shall limit our focus to some of the prevalent malicious activities that could be done by malware. These examples include for our purpose:

1. **Sending SMS's without the user knowing about them**

2. **Accessing potentially unwanted URLs either for advertising or for information leakage.**

At the very root of this framework lies the idea that malware generally infects a system through various entry points such as resources, contexts, intents, etc. So, a one size fits all solution would be very tough to develop.

However, the focal point that we have come to know from preceding and extensive discussion are that Permissions are the basic area which opens access to the system for malware and benign applications alike. This means that if we can somehow implement fine graining into these permissions, we can achieve great success in alleviating this security problem encountered by Android. To this extent, we propose that the coarse-grained permissions model can be modified by including finer graining of permissions through code.Thus, from the preceding discussion, we propose a framework which necessarily is a Two-Tier architecture in which we introduce a middle layer between the application that is to be installed or tested and the operating system, i.e., Android.

VI. THE ASBP (APP SANITIZATION AND BETTER PERMISSIONS) FRAMEWORK

1. The framework essentially will consist of two parts:

a) The middle layer which will hold the policy logic for granular permissions (from now on also referred to as the Granular Access Layer (GAL)).

b) The instrumentation layer where the actual part of installing the policy logic will be carried out by using reverse engineering. (from now on referred to as the Instrumentation Technique (IT)).

2. It is required to find out which permissions can be targeted for granularity and accordingly dealt with. For our purpose of demonstration, we shall be focusing on only INTERNET and the send and receive SMS permissions.
3. The application that is to be tested will not be allowed to play with any area of the operating system.
4. The application will see all interaction with the OS as direct interaction with the native resources through inbuilt code.
5. In reality, all communication between the Application and OS would be carried out through the wrapper, which is GAL.
6. Essentially the framework would act as a self-contained sandbox where the application would be checked and thoroughly vetted for any nefarious intentions.

Operating Assumptions For our discussion, our model would assume that any app which is installed on the Play Store, has been vetted fully and therefore we would take a similar app from a third-party source and then compare that app with its corresponding app in the play store. Then the third-party app would be put into the sandbox/framework and then vetted for any malicious code that has been put into it by nefarious developers. So, necessarily, the Play store app would serve as a reference through which we can infer the policy logic for vetting.In addition to detection of malware, this design would also enable us to implement granular permissions on the app in question, and we would be successful in ascertaining whether the Play Store App needs refinement or not. *Essentially our framework allows us to achieve two different objectives:* We would get a malware-free application that we can blindly install on the device**.**

We can further customize the application so that we can include granularity of permissions in it also.

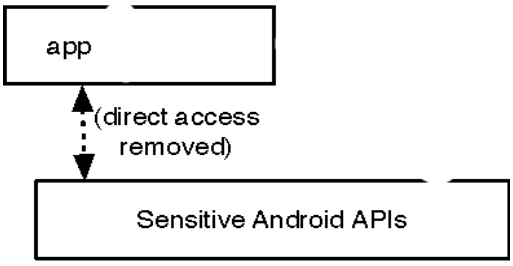


Figure 6.2 GAL architecture

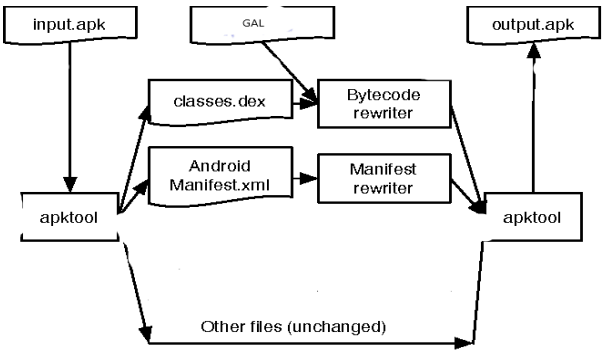


Figure 6.3 IT architecture

As is quite clear from these figures GAL is responsible for providing granular access to sensitive resources whereas IT allows us to customize the original apk which can make use of GAL to provide restricted access to sensitive resources.

VII. IMPLEMENTATION WITH ANALYSIS AND RESULTS

Moving further from the preceding section, in this paper, the implementation of the framework is done in two steps:

1. **Using ASBP to identify android vulnerabilities and removing them.**
   For this purpose, we will focus on apps that contain malicious code which is used for sending unsolicited messages. These represent one kind of vulnerability which we shall focus on removing.
2. **Using ASBP to demonstrate fine graining of permissions with user involvement.**

For this purpose, we have created a functional android browser application which serves as a prototype for enforcing fine-grained policies. We will focus on the INTERNET permission, which is pervasive permission that is automatically granted to apps. This will further strengthen our guidelines that it needs to be labelled as dangerous. By using fine graining, we constrain our browser app to restrict access to particular user chosen sites, so even while the app has INTERNET access, we have fine-grained it through the code in ASBP.As a further practical standpoint, a wrapper (GAL) that will offer permission controlling interface will be introduced into the APK installer file for the apps. We will improve the wrapper that will offer a permission management interface and a patcher (part of IT) that is being added into the APK installer file.

Through this, we will provide a little private area to apps taken from untrusted sources. They will not allow the app to play with the rest of the device that stores your confidential information. Android devices can contact the internet, make online bank transmissions, handle social networks, etc. Using Androgaurd, we can find the risk of the apk file and similarities of apk file and patched apk file. We will analyze the security level of android application before and after patching.

A. System Overview

First, we present a complete overview of ASBP. The step-by-step process is as follows:

1. We take an application from a third party store or Play store.
2. In case it is from a third party store, we will check for vulnerabilities using Androguard.
3. If both files match, then there are no vulnerabilities. We can proceed to step 7.
4. In case there are vulnerabilities we shall check for them by using our IT tool.
5. We shall specifically look for unauthorized SMS sending code and for checking unfettered internet access.
6. After the process is complete, we can again check using Androguard whether there still exist some vulnerabilities.
7. We can check for the possibility of granularity.
8. If the possibility exists, provide for granular mechanism.
9. Stop.

As is clear from the previously outlined steps that we will accomplish both our objectives. We shall be able to identify vulnerabilities, and we shall be able to improve the permissions model too.

Instrumentation Technique: The first part of the design requires creating an instrumentation tool which can manipulate the APK and insert the files that will intercept the application's behavior at run-time. A step-by-step process of achieving this is outlined below.

1. Take our prototype application or any real-world apk file.
2. Disassemble the APK using apktool into assembly code .smali files.
3. Also, check the manifest file of the apk.
4. The manifest will show us the permissions.
5. The smali code now has to be checked for discrepancies.
6. Inject our custom code which will override the calls to the inbuilt classes with calls to our defined classes.
7. Reassemble the APK with apktool.
8. Sign APK with jarsigner using a key provided by ourselves so it will be accepted at load time.
9. Install signed APK onto the device, using adb tool.

Instead of using steps 7,8,9, we can accomplish all the code by using a patcher that we have developed for our prototype in a single step to save time.

Run-time Flow

The code as classes which will be utilized for guiding into the broken down APK is the key factor behind catching and effectively taking care of undesired calls at run-time. To deal with this, two documents will be embedded which contain the code for IT and GAL. The IT class is the main point of contact when the startActivity is called and handles the extraction of data to go to the GAL. The IT should wait until GAL has evaluated the data and given its report. On the off chance that the activity is non malicious, IT should continue the application, and if the activity beneficiary has been blacklisted, it should obstruct the activity. The GAL is called from IT either because IT has been required the first run through and has Policy design information to send to GAL, or it requires the GAL to assess conceivably unsafe information. If Policy design information is passed in, the GAL should store the data in a blacklist that it can utilize later for checking. However, if IT is requesting that the GAL check approach, it should utilize the blacklist information to decide the danger level, alongside examining whether the activity is conceivably a URL or SMS. The Policy document is not embedded during APK instrumentation. Rather, it will be made by IT during run-time. The IT will initially check if the Policy design record as of now exists, on the off chance that it doesn't, it will make one, but if it does exist, it will attach new data to it. The Policy setup will be utilized to store blacklisted numbers that can be recovered each time the application runs. The organization of the policy document will be straightforward with the goal that it might be perused rapidly from the IT on start-up. After both the IT and GAL have worked, each class will at that point be dismantled utilizing APKTool so that the. smali variant of each record might be embedded and recompiled with the adjusted APK.

Now we shall further proceed with the help of following use cases which are discussed in the next section.

Use Case 1: Identifying Android Vulnerabilities and Rectification with ASBP

A.**Design Requirements** We defined the following set of requirements based upon our findings from the literature review.

- Modified applications that do not violate policies should function as expected.
- Instrumentation of the application should rely purely on having access to the APK, and the source code would primarily be unavailable.

B. Policies

In order to evaluate our system, a set of policies must be outlined, which will be used to ensure our system correctly intercepts the desired activities. The following policy has been chosen as it addresses operations that are commonly used without permission by malware.

- Prevent applications from sending SMS messages to unauthorized numbers.

C. Design and Implementation Decisions

The following key design and implementation decisions were made in order to satisfy the design requirements.

- Hooking/Inserting Methods at Run-time: The most effective approach appears to be an actual modification of the application before it is installed on the device. As stated in the requirements deploying new firmware onto the Android device is not a desirable option. Our design is based upon modifying the code of the application, ***specifically in wrapping the methods by overriding which we presume can be the entry location for malicious behavior***. This design had its origins in the observation that many methods we were thinking of intercepting made calls to the same class. This involved the breakdown and disassembly of the APK, inserting the wrapper class, and changing class paths where appropriate to direct the actual class to the modified custom class, before returning to the normal execution of the task.

With this approach, we change the target to point to the customized class. Our class contains custom policy check methods which are specified by the user.

This is a simpler approach than others because our bytecode modification does not make changes to existing code classes instead provides a workaround hence does not take anything away from the original.

D. Actual Experiment

The choice of a dataset is important because it reflects real-world applications. Choosing a poor dataset may yield to better results (for instance if the code within applications is always small) but nonuseful results since they are non-applicable on most real-world applications. We apply the whole experimental protocol on a set of 50 Android applications randomly selected among the top 50000 applications from the Android market. They span various domains such as finance, games, communications, multimedia, system, or news.

These apps have been downloaded from third-party stores like AppBrain, SlideMe, and ApkPure.

Out of the dataset mentioned above, we chose to focus on five main apps, namely calendar, notes, birthday free, ACR, and contacts, because of two main reasons:

1. They are mostly prebuilt into every phone so are susceptible to vendor customized code.
2. They are also most prevalent on third-party stores as per our requirement.

The results so obtained are explained in the concluding section of this paper.

Use Case 2: Granular Permissions with ASBP

A. Design Requirements

- Users of the application should be conferred with on decisions of whether to permit or deny methods intercepted by policies.
- The design should make no assumptions about the user's technical knowledge other than the fact that they can download and install Android applications. The requirement stems from the fact that users of all backgrounds own and operate Android devices. It, therefore, is potentially isolating to user groups if they are assumed to have high technical knowledge.

B. Policies

- Restricting the application from making connections via a web browser. A user should be made aware when an application is attempting connections to a particular domain and should be given the option to permit or deny a URL.

C. Design and Implementation Decisions

- User Interface: One of the design requirements outlines that there should be no assumptions made about the user's technical knowledge and given that disassembling and reassembling applications recruits the help of apktool, keytool and jarsigner it seems sensible to implement a user interface to complement the hooking methods.
- Policy Configuration File: To allow fine-grained policies, it is ideal to give users a choice about an URL whether they trust the recipient or not. We store the policy information within a file accessible only to the modified application. To achieve this, the IT will read from the Policy configuration file when first started or create a Policy configuration file if one does not already exist.

D. Run Time Flow

1. The IT will come into action when an "original" class calls the startActivity method.
2. If it is the first time, the IT has been called it should first read in the Policy configuration file, so that it may extract the blacklisted defined by the user.
3. The IT extracts information about the activity intercepted and pass the activity data over to GAL.
4. The GAL checks the activity data passed in.
5. The activity will then be passed into a policy specific method, where the data is inspected minutely.
6. GAL will compare the received information with the created blacklist.
7. After the GAL has performed its checks, it will return a policy report to the IT.
8. The IT ses the information from the GAL.If there are no threats found normal operation resumes.
9. If the GAL had returned information indicating the URL has already been blacklisted, application is resumed after blocking the malicious activity.
10. If the URL is unknown, it is assumed ASBP could not find any pre-existing url's that matched the Policy configuration file.
11. If the user does choose to blacklist the URL from step 6, the GAL is notified of the changes, and the user preferences are persistently stored.

E. Actual Experiment

To explain granular permissions,we have developed a test app named browser wherein the flow of events outlined in the preceding paragraphs takes place. This prototype tester app assumes that the app for granularization is free from any malicious code and just needs a little fine-tuning in order to increase its security.

The results so obtained are explained in the concluding section of this paper.

Results of ASBP Malapp Detection and Rectification

Malicious Apk	Fuzzy Risk Value (%)	Similarity (%) with Original Apk	Difference (%) between Original & Malicious Apk
iCalendar	92	99.98	0.022
Notes	92.4	99.965	0.035
Birthday Free	91.6	99.948	0.052
ACR	88.45	99.946	0.054
Contacts	90.23	99.925	0.075

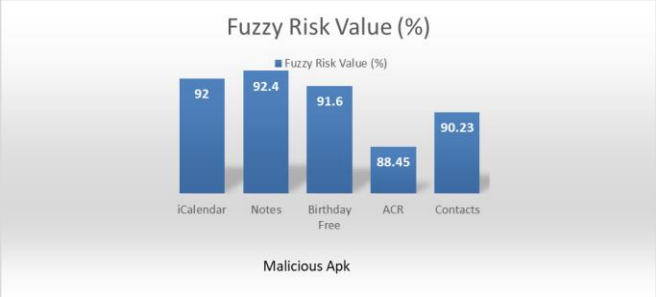


Fig 7.1 Fuzzy Risk Values

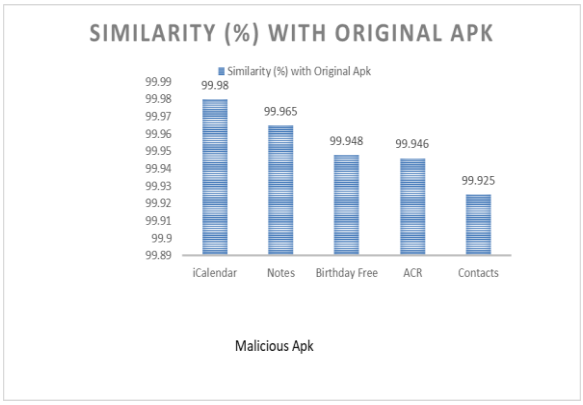


Fig 7.2 Similarity With Original APK

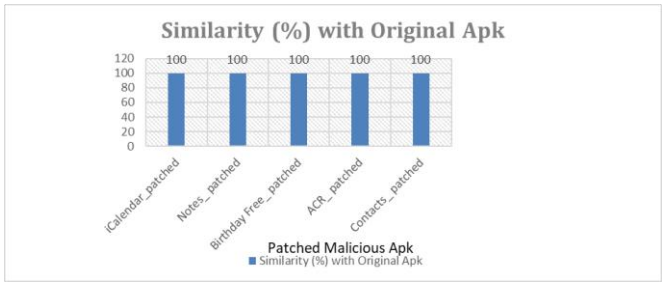


Fig 7.4 After ASBP Difference reduced to zero

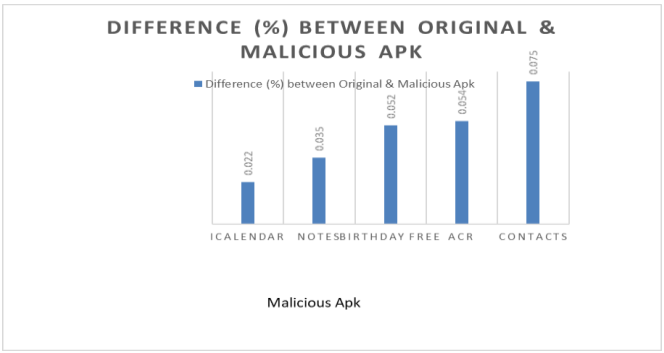


Fig 7.3 Difference between original and Malicious apk

Patched Malicious Apk	Similarity (%) with Original Apk	Difference After Patching (%)
iCalendar_patched	100	0.0
Notes_ patched	100	0.0
Birthday Free_ patched	100	0.0
ACR_ patched	100	0.0
Contacts_ patched	100	0.0

Results of ASBP Granular Permissions

Company	Device	Version	Build #	Memory
Samsung	Galaxy On7	5.1.1 [lollipop]	LMY47XG600F YXXUIAPD5	3gb
Sony	Xperia Tipo Dual	6.0.4 [Ice-cream Sandwitch]	11.0.A.6.8	3gb
Lenovo	A369i	6.2.2 [jellybean]	ROW_S111_140522	3gb

Lenovo	K3 Note	5.1.1 [Lollipop]	VIBEUI_V2.5_1512_ 5.495.1_ST_K50_T5	2gb
Samsung	Galaxy J1Duos	6.6.4 [kitkat]	KTU84P.J100HD DU0APB4	2gb
Sony	Xperia SL	6.0.4 [Ice-cream sandwitch]	6.1.A.2.45	2gb
Motorola	Moto G1	5.0.2 [lollipop]	LXB22.46-28	2gb

Fig 7.5 Experimental setup for granular permissions

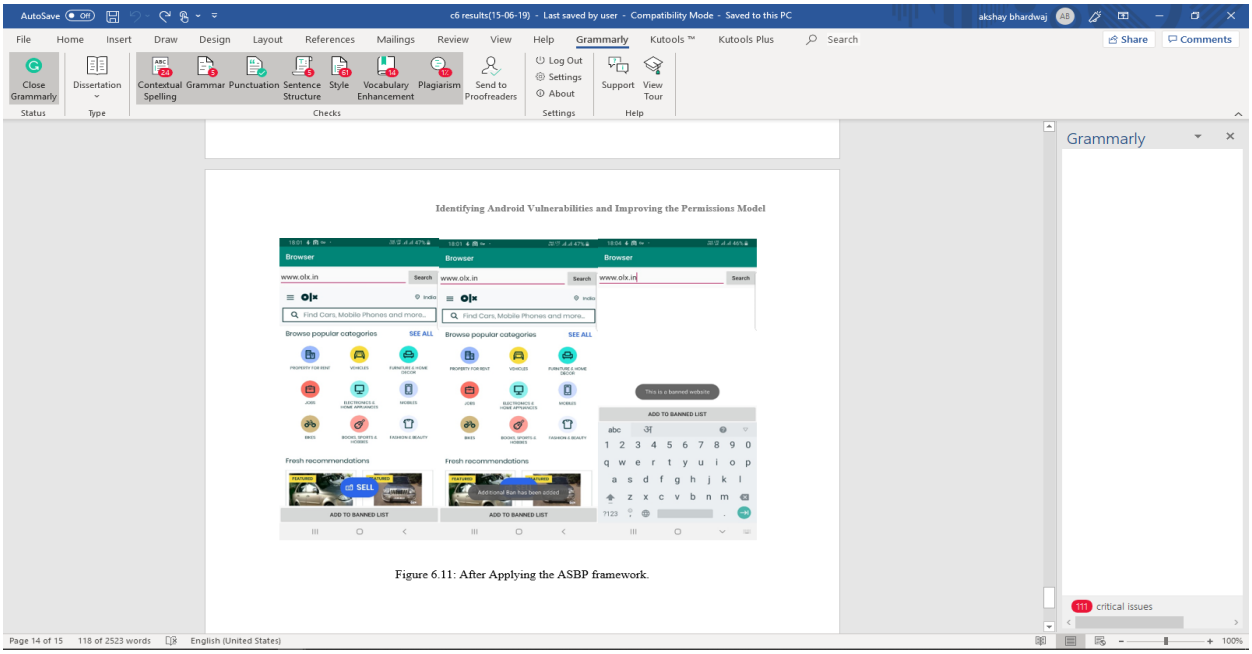
We tested that policies were enforced correctly against misbehaving or malicious applications. This functional testing was implemented by creating our test application.

Web Browser Policy Enforcement: Our policy enforcement testing mechanism will test the web browser activity. The testing performed will be

ASBP will be testing intents to check whether a valid URL is included in the intent information.

An application has been created of which the user will automatically be taken to a certain website when a button is clicked, which ASBP should intercept. The application will be installed fresh into the device with a dataset of around 60 URLs.

Web Browser Policy Enforcement Results. The resulting actions from ASBP can be seen in the following figures:



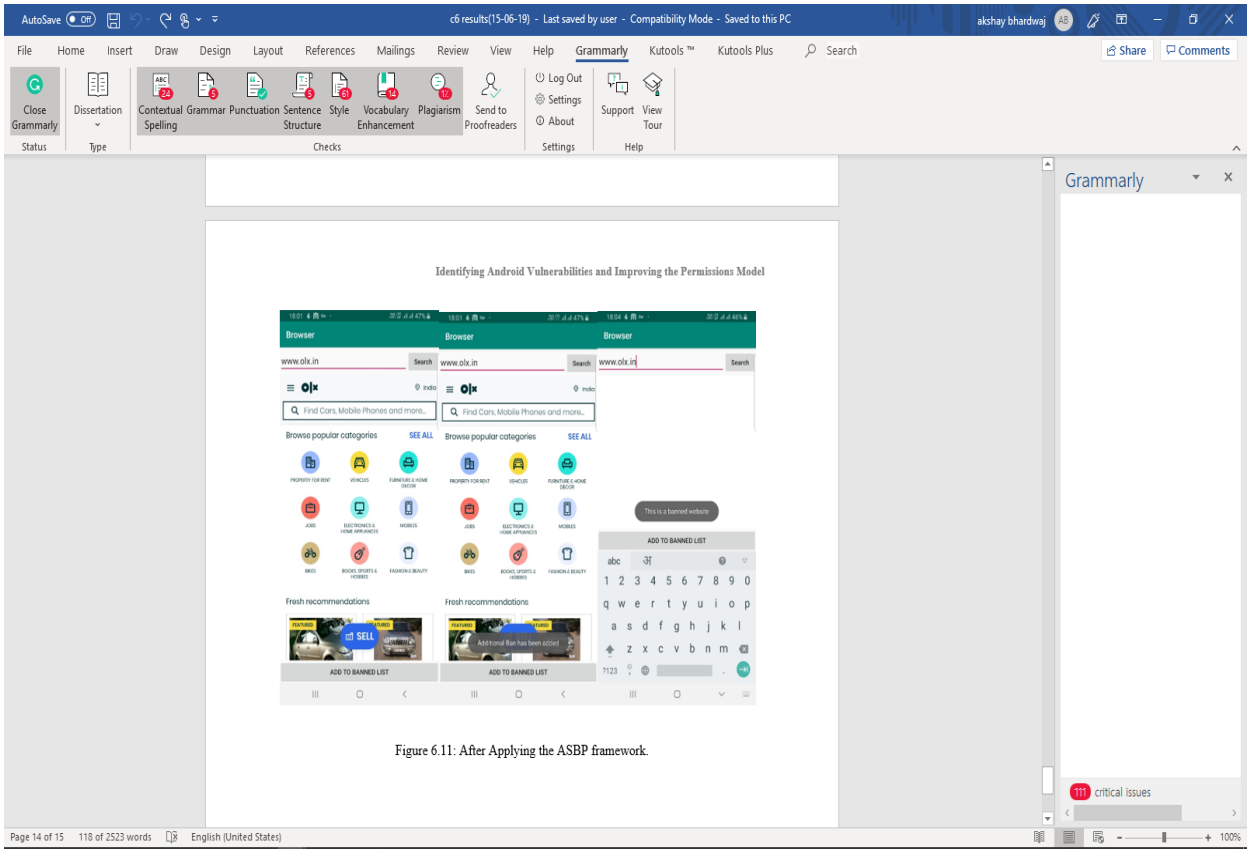


Fig. 7.6 Before and After ASBP Granularity Introduction

VIII. CONCLUSION

We aim to fix the majority of challenges that Android encounters by providing a simple and effective technology tool. Our Most Important contributions are that: Where policies ensuring security and protecting privacy can be enforced, we have built a method to repackage arbitrary APKs which could be malicious.We provide a means of protecting users from the software without making any adjustments to the underlying Android architecture. This makes ASBP a desirable solution.ASBP can be a robust technology that was tested on many versions of Android. ASBP has low overhead and, unlike other approaches, is mobile over different OS versions.

REFERENCES

1. Bartel, A., Klein, J., Monperrus, M. and Le Traon, Y. (2014). Static Analysis for Extracting Permission Checks of a Large Scale Framework: The Challenges and Solutions for Analyzing Android. IEEE Transactions on Software Engineering, 40(6), pp.617-632.
2. Li, W., Jiang, G.,(2015).Detecting Malware for Android Platform: An SVM based approach. 2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing.
3. Idrees, F., Muttukrishnan, R. (2014).Investigating the Android Intents and Permissions for Malware detection. 978-1-4799-5041-6/14/$31.00 ©2014 IEEE.
4. Zhou, Y., Jiang, X,.(2012). Dissecting Android Malware: Characterization and Evolution,2012 IEEE Symposium on Security and Privacy
5. Fan, Y. and Xu, N. (2015). The Analysis of Android Malware Behaviors. International Journal of Security and Its Applications, 9(3), pp.335-346.
6. Tchakounte, F. (2014). Permission-based Malware Detection Mechanisms on Android: Analysis and Perspectives. Journal of Computer Science and Software Application. 1. 63-77.
7. Enck, W., Ongtang, M. and McDaniel, P. (2009). Understanding Android Security. IEEE Security & Privacy Magazine, 7(1), pp.50-57.

AUTHORS PROFILE

Er. Akshay Bhardwaj, is an Asstt Prof. at UIIT Deptt. In Himachal Pradesh University Shimla.He is working here since 2004.He is pursuing his PhD. in computer science from HPU after completing his BTech and MTech in Information Technology. His areas of interests are Distributed systems (Networks), Operating systems,Mobile security and forensics.. He is on the academic bodies of many universities. He can be reached at akshay117@gmail.com and Mobile: 09418482826.

Dr. A. J. Singh, is a Professor in the Department of Computer Science in Himachal Pradesh University Shimla. He has been in this Department since 1992. He has obtained his Bachelor of Engineering degree in Computer Technology from National Institute of Technology (MANIT) Bhopal, Master of Science in Distributed Information Systems from University of East London (UK) under British Government ODASS Scholarship and Ph. D. degree Himachal Pradesh University Shimla. He worked on deputation to Royal Government of Bhutan (Education Division) under Colombo Plan for three years. He has published more than 50 research papers, supervised six Ph. D. students, 9 students doing Phd under his supervision and guided many M. Tech. Dissertations. His areas of interests are Distributed systems (Networks and DBMS), and ICT for Development. He is on the academic bodies of many universities. He can be reached at aj_singh_69@yahoo.co.uk and Mobile: 09418484855.